UNIVERSITAS Dinamika

ALGORITMA PEMROGRAMAN
Pertemuan XII

SORTING

Oleh
Achmad Arrosyidi

Universitas Dinamika

MATERI KULIAH

1. Bubble Sort
2. Selection Sort
3. Merge Sort
4. Quick Sort
5. Insertion Sort
6. Shell Sort
7. Radix Sort

UNIVERSITAS Dinamika

TUJUAN PEMBELAJARAN

Umum:

- ✓ Mahasiswa dapat menguraikan logika berbagai jenis Algoritma Pengurutan (Sorting).

Khusus:

- ✓ Mahasiswa dapat menerapkan logika berbagai jenis Algoritma Pengurutan (Sorting).

1. BUBBLE SORT

BUBBLE SORT

An Animated Example

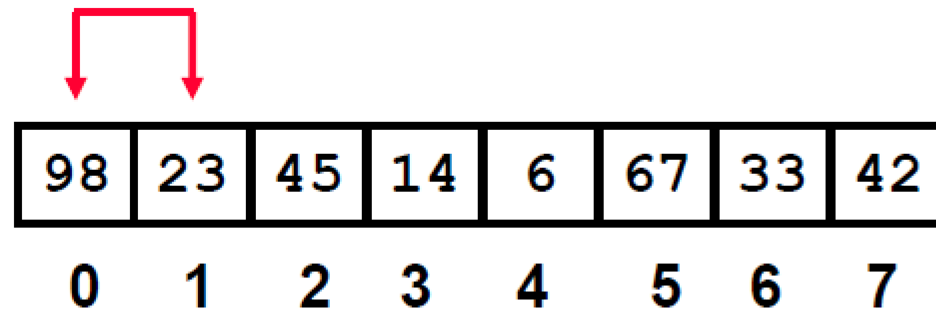
N	8
pos_akhir	6
index	

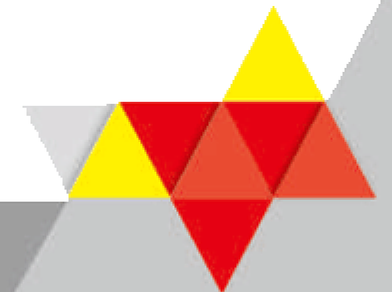
did_swap	true

98	23	45	14	6	67	33	42
0	1	2	3	4	5	6	7

An Animated Example

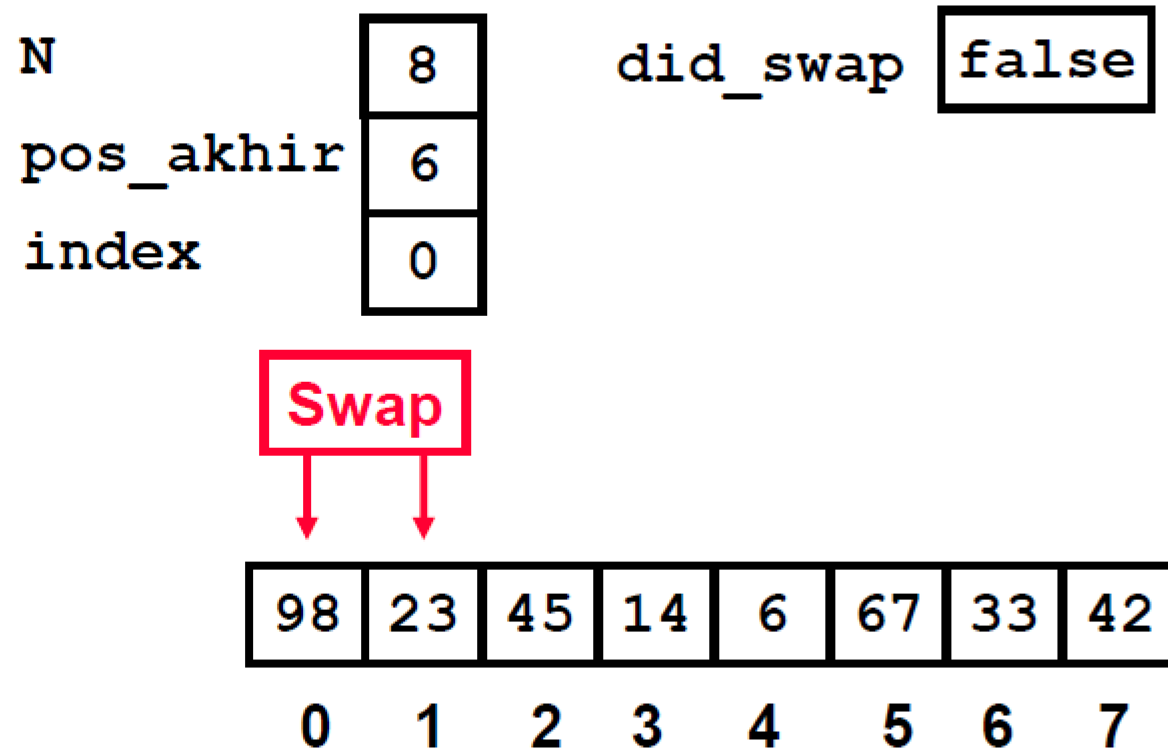
N	8	did_swap	false
pos_akhir	6		
index	0		

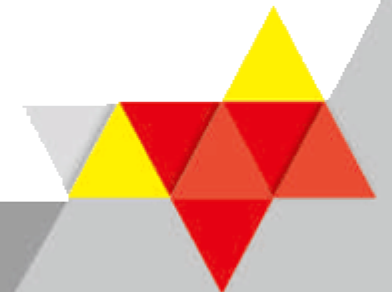




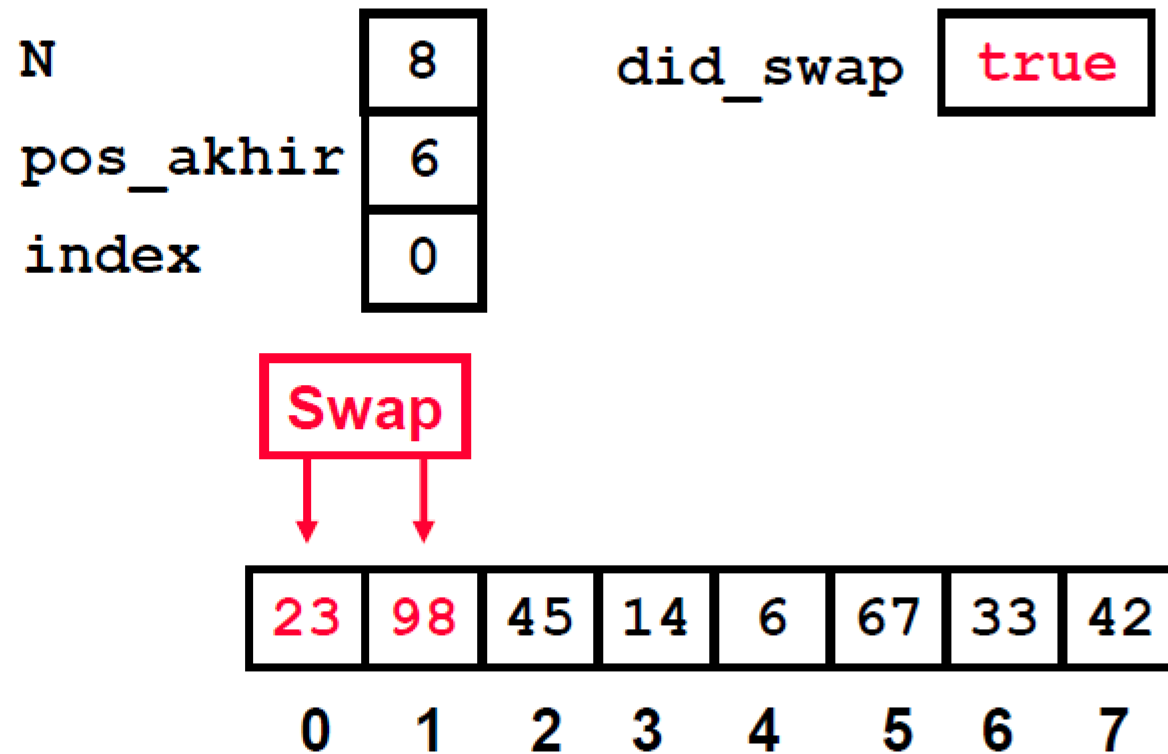
BUBBLE SORT

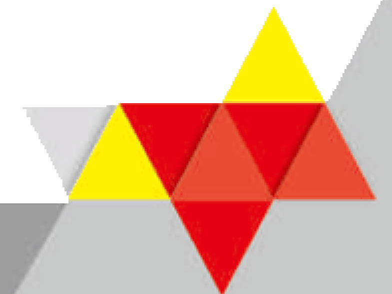
An Animated Example





An Animated Example

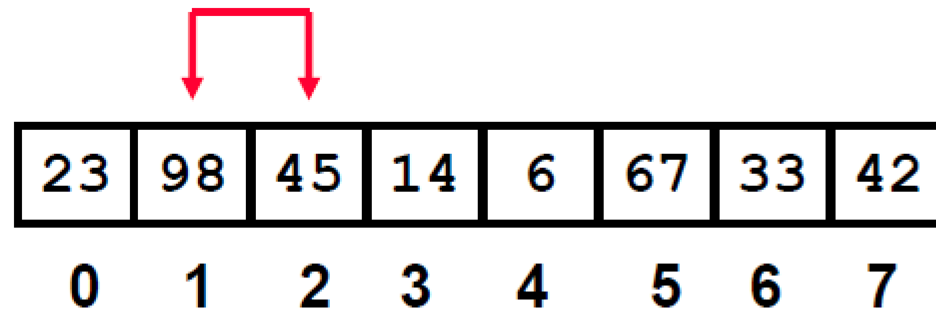


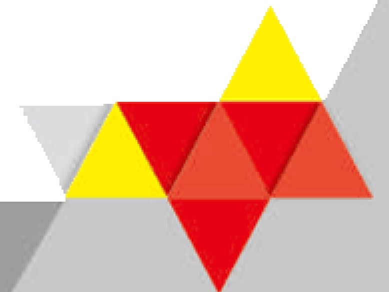


An Animated Example

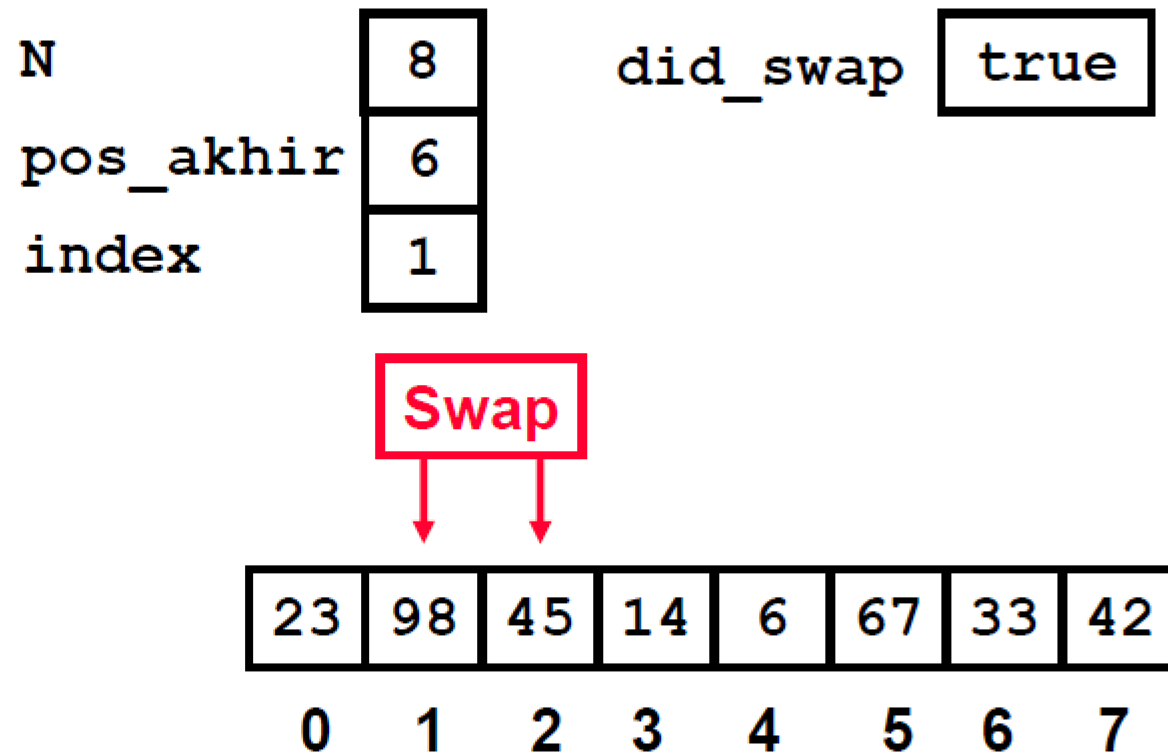
N	8
pos_akhir	6
index	1

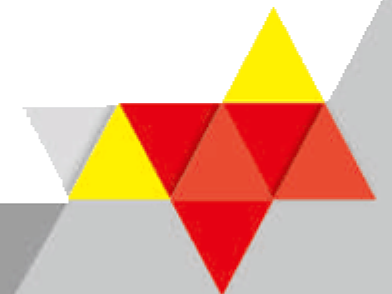
did_swap	true





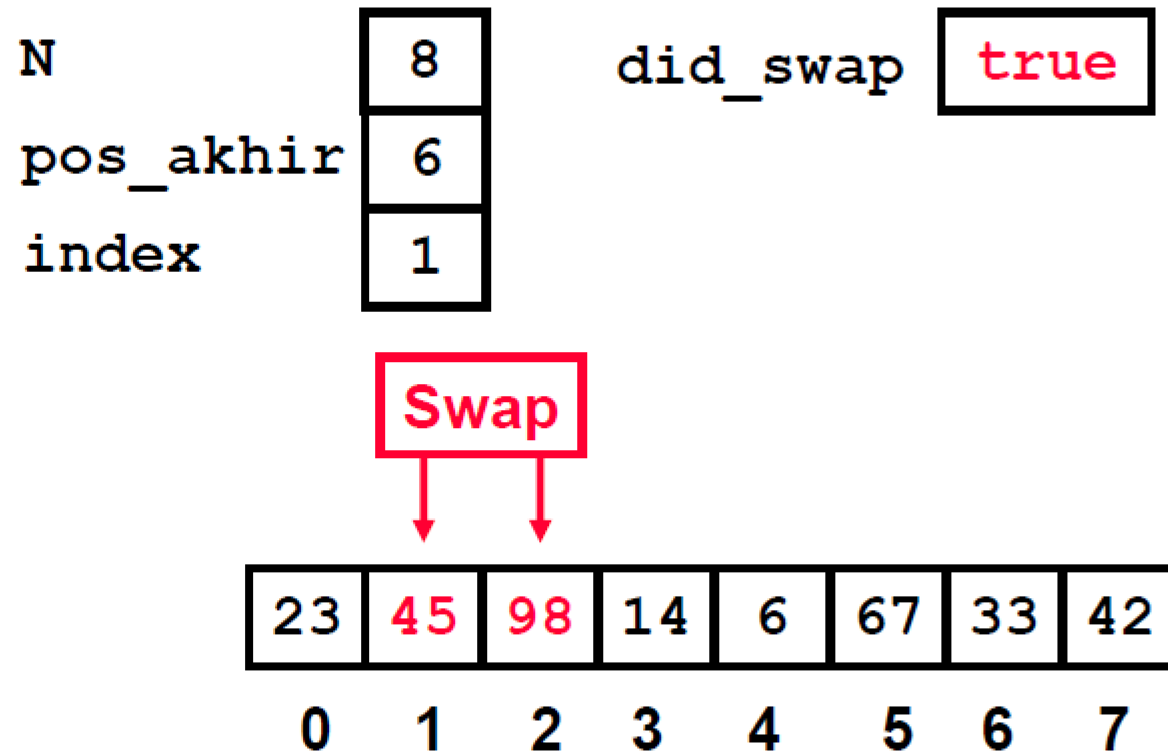
An Animated Example

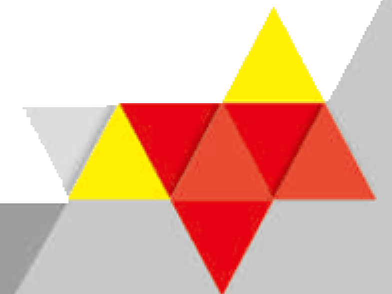




BUBBLE SORT

An Animated Example

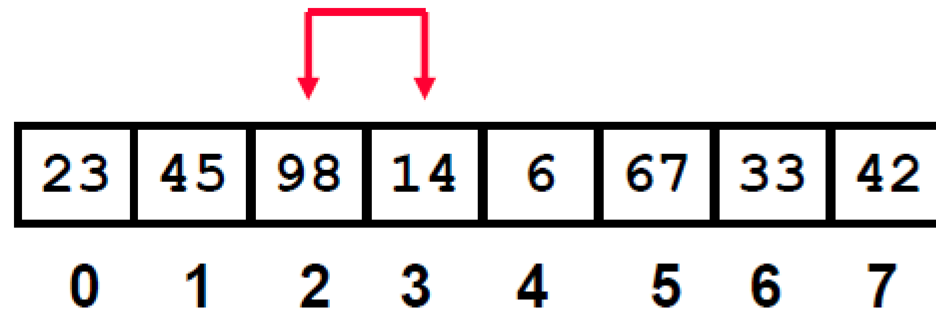


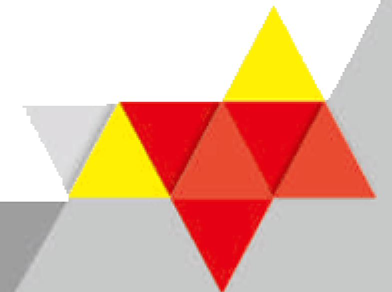


An Animated Example

N	8
pos_akhir	6
index	2

did_swap	true

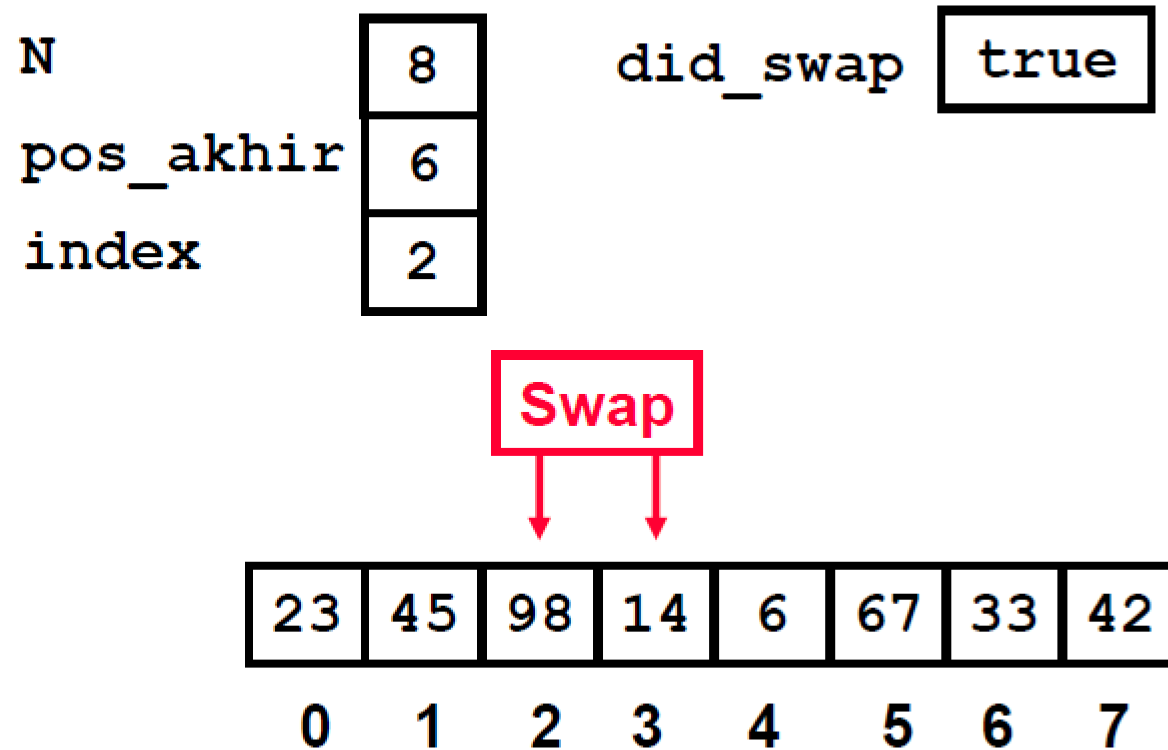


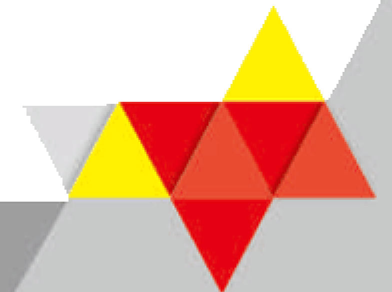


UNIVERSITAS Dinamika

BUBBLE SORT

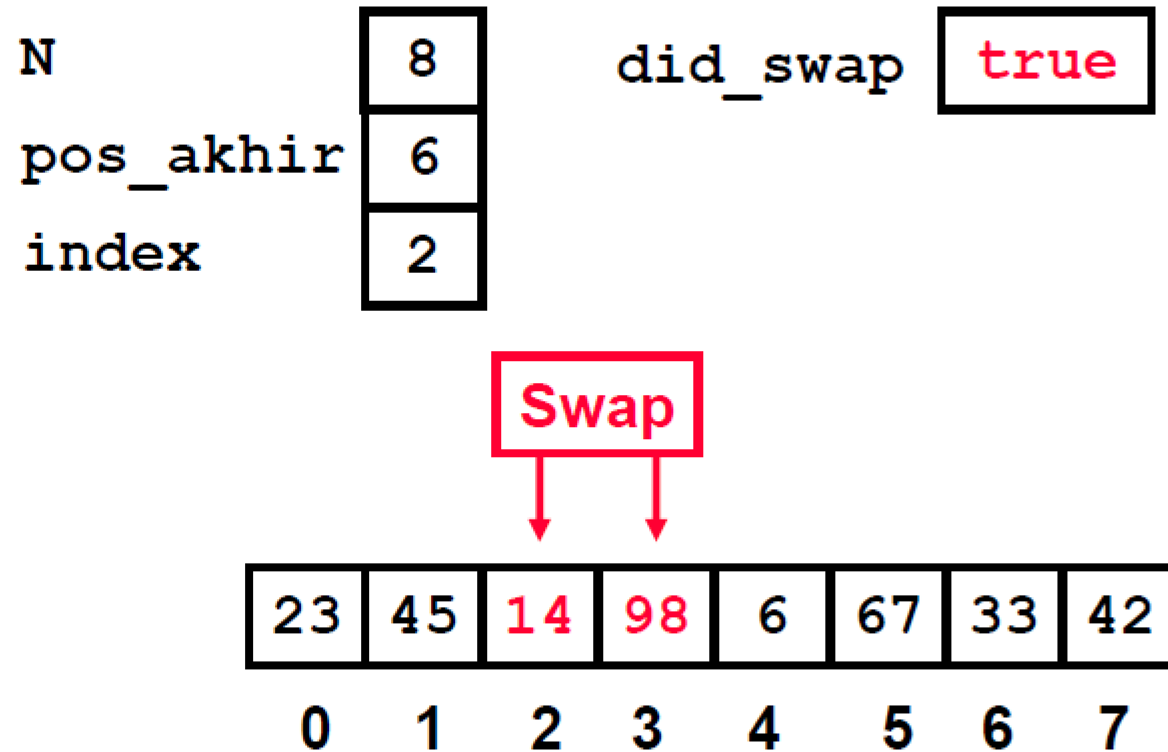
An Animated Example

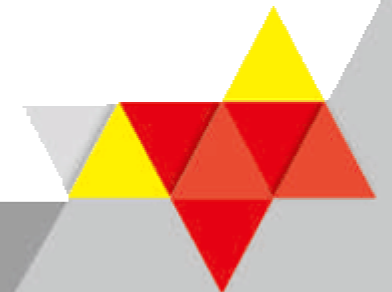




BUBBLE SORT

An Animated Example

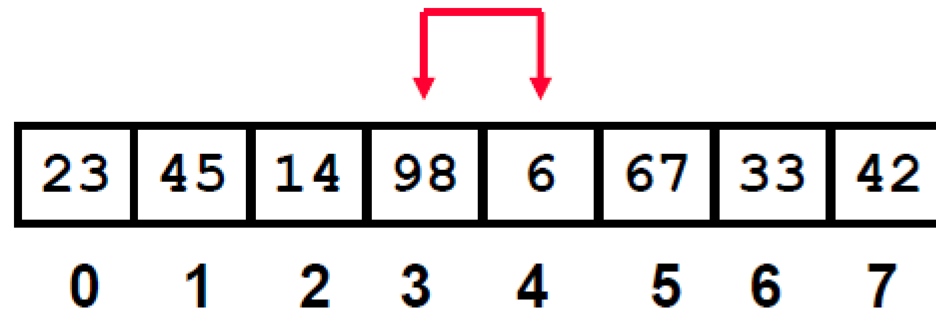


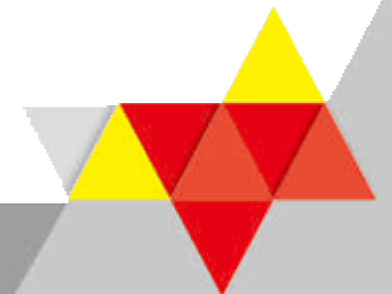


An Animated Example

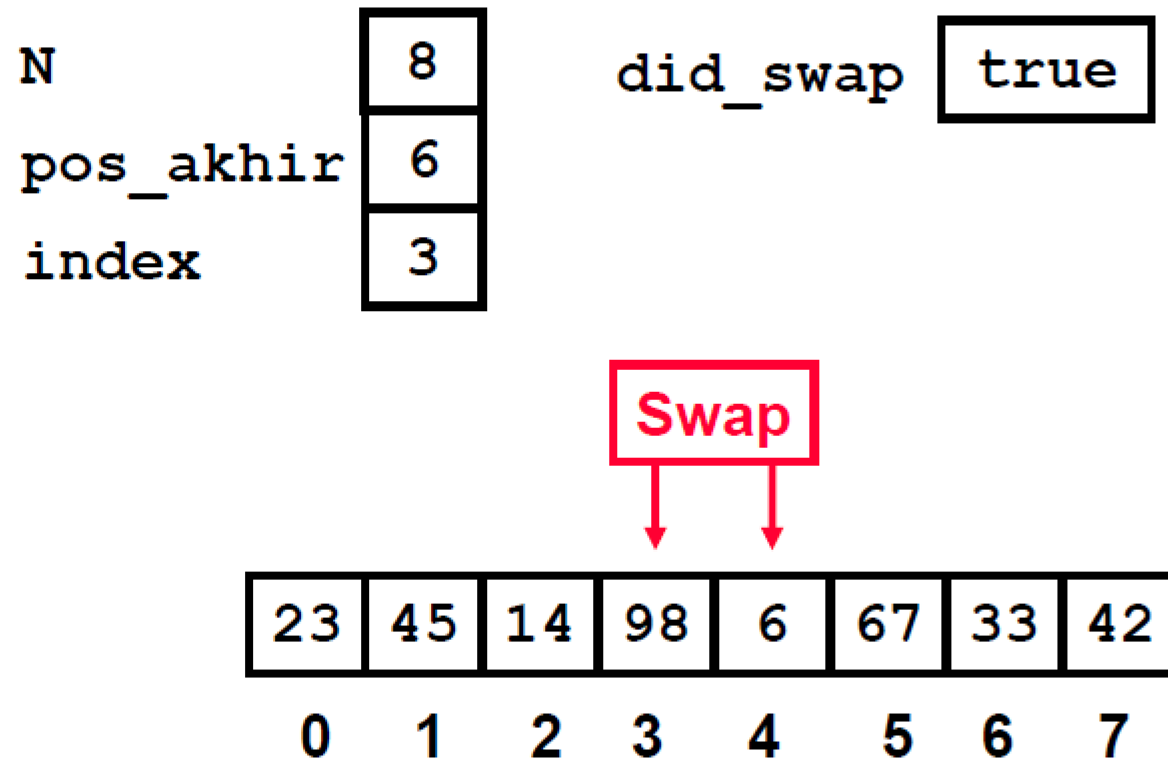
N	8
pos_akhir	6
index	3

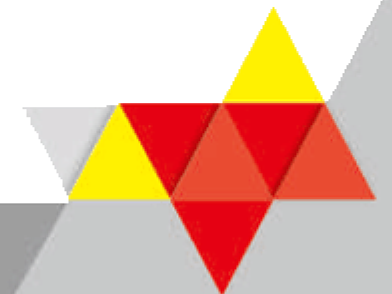
did_swap	true



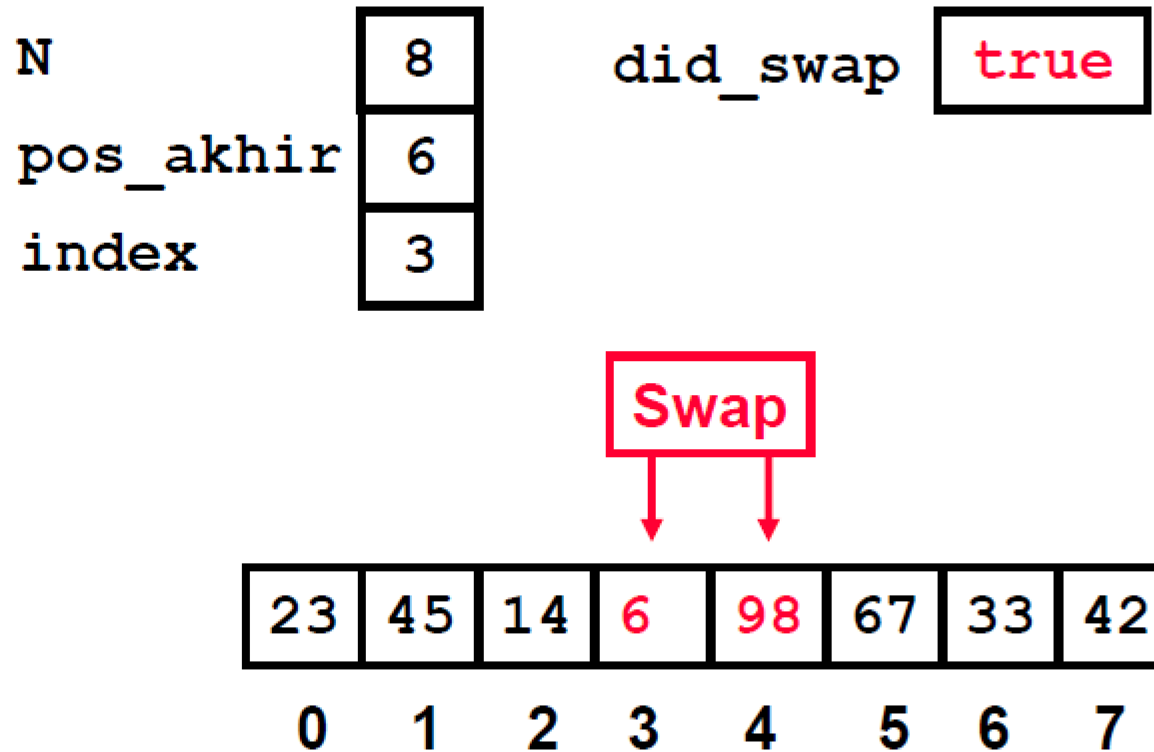


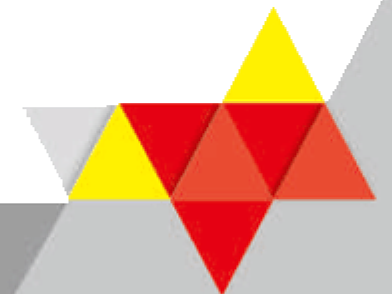
An Animated Example





An Animated Example





Universitas Dinamika

BUBBLE SORT

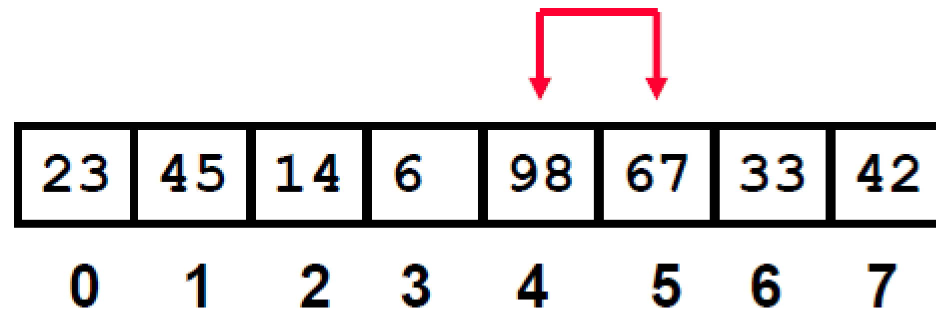
An Animated Example

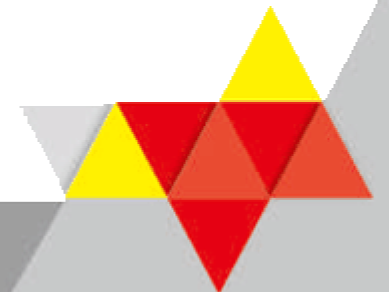
N 8

pos_akhir 6

index 4

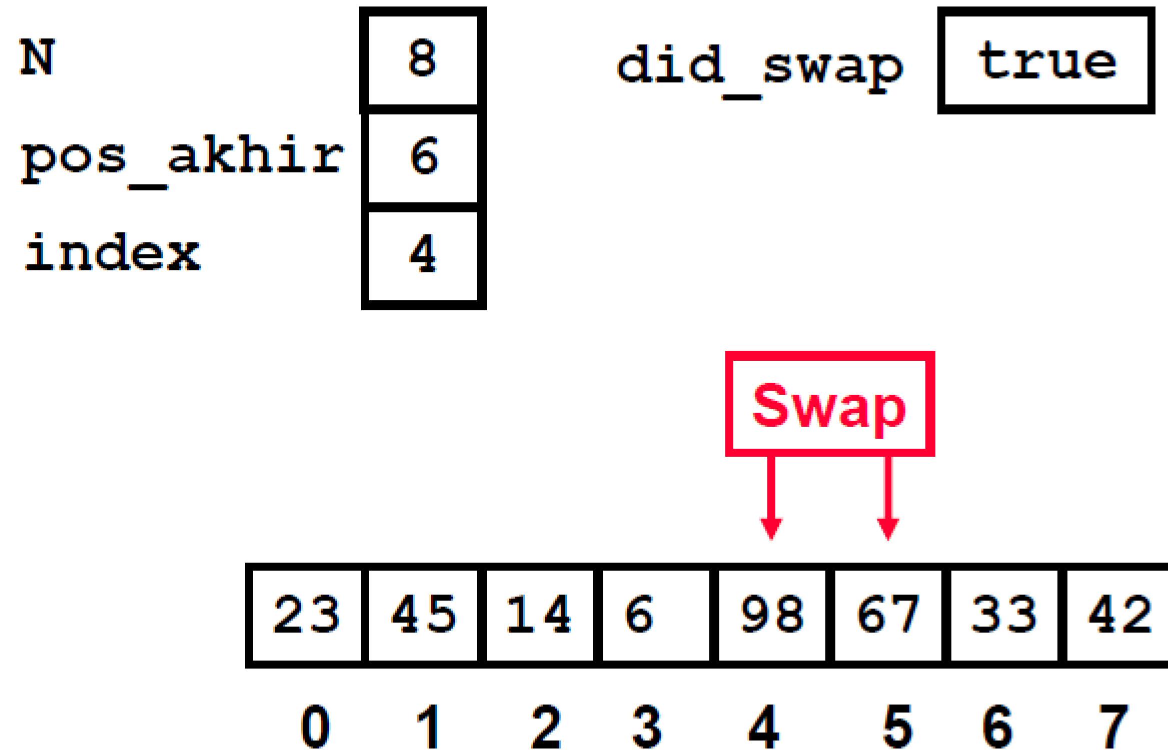
did_swap true

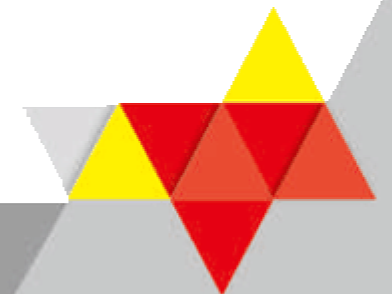




BUBBLE SORT

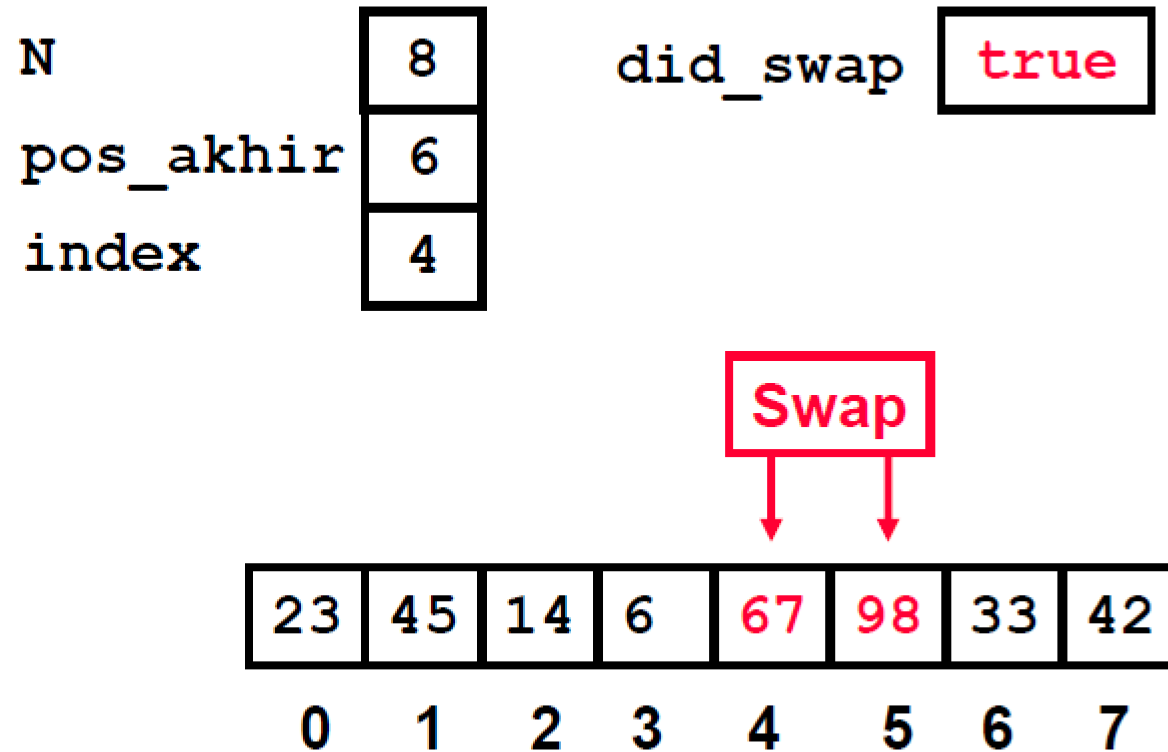
An Animated Example

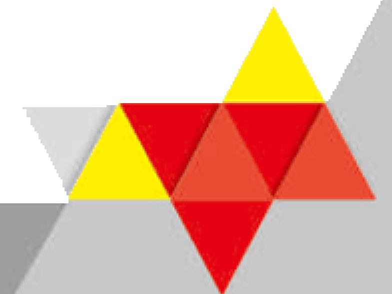




BUBBLE SORT

An Animated Example

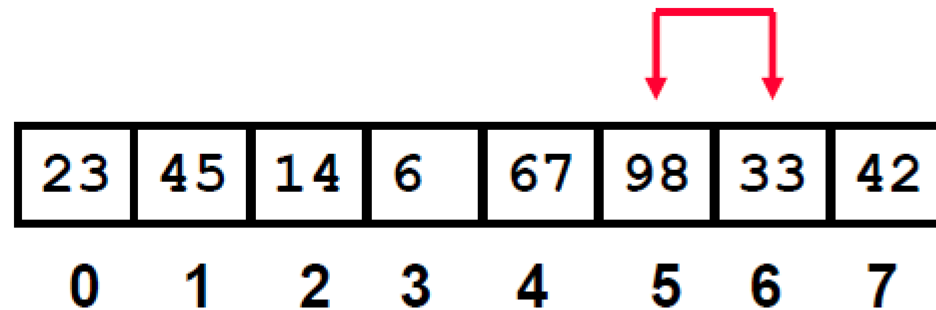


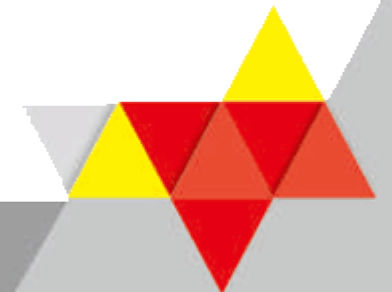


An Animated Example

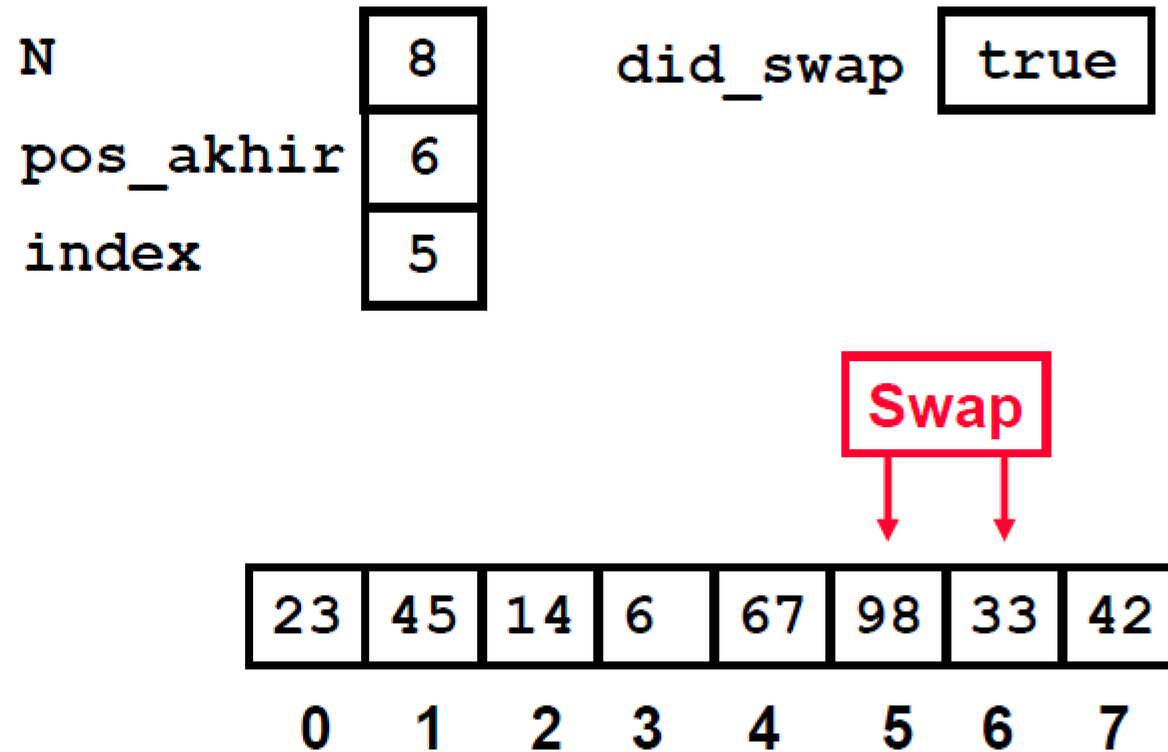
N	8
pos_akhir	6
index	5

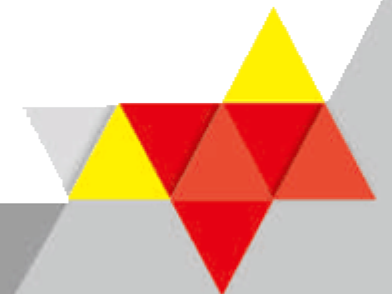
did_swap	true



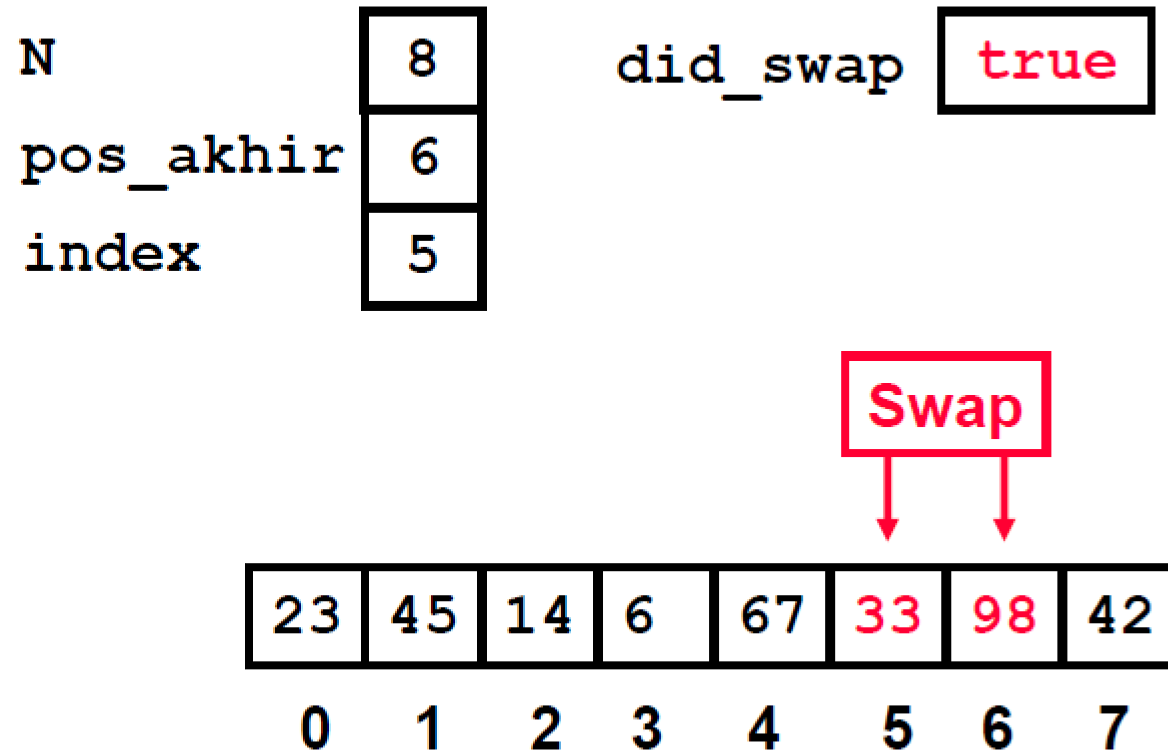


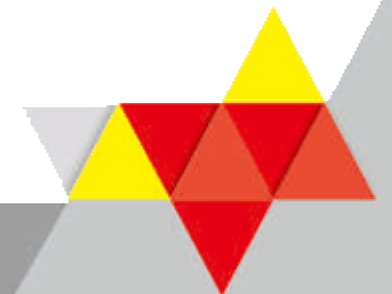
An Animated Example





An Animated Example



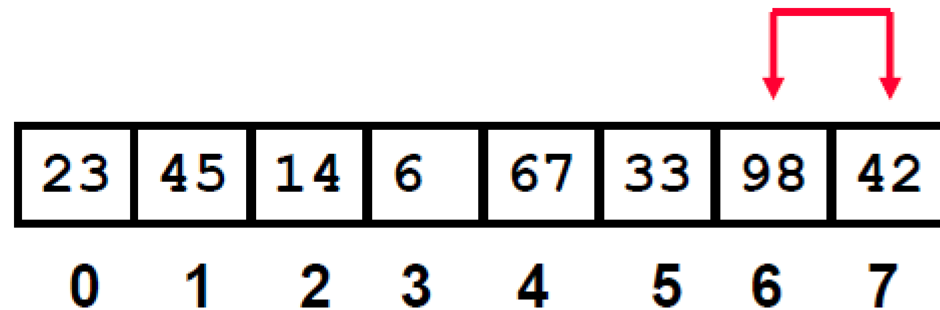


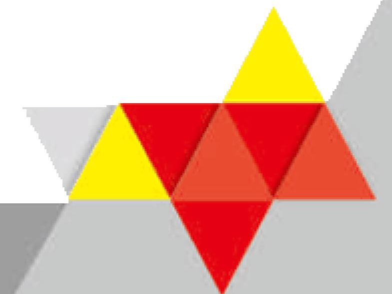
BUBBLE SORT

An Animated Example

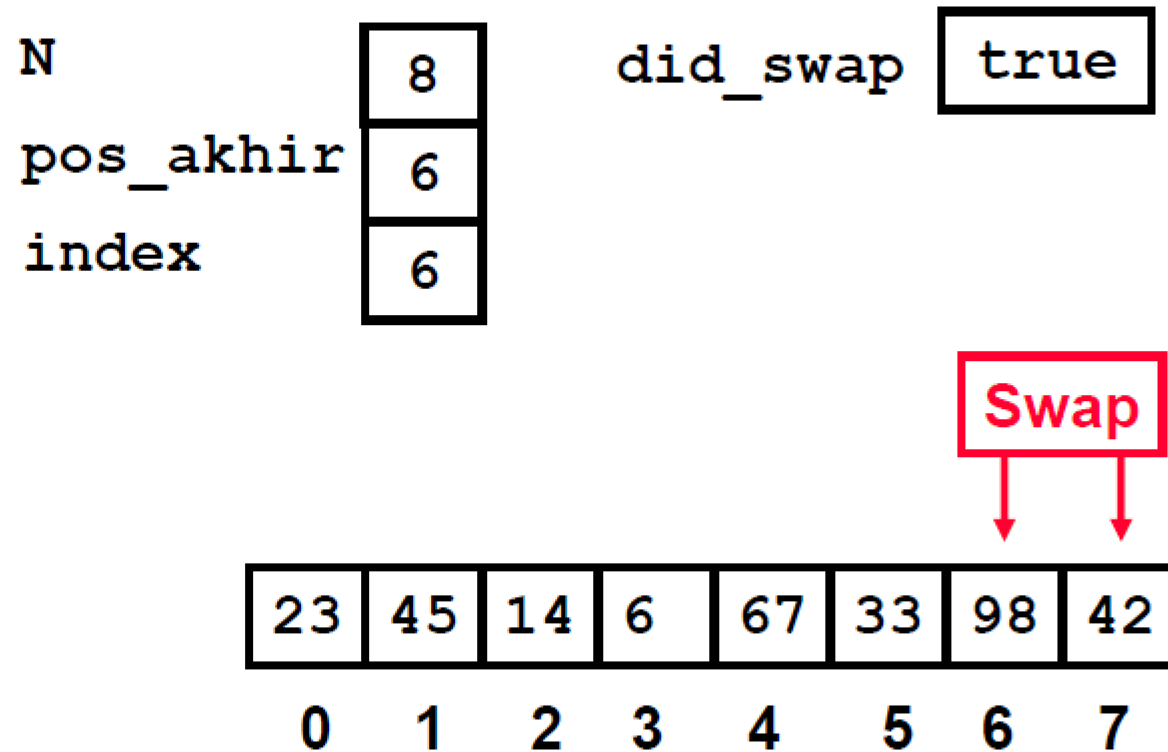
N	8
pos_akhir	6
index	6

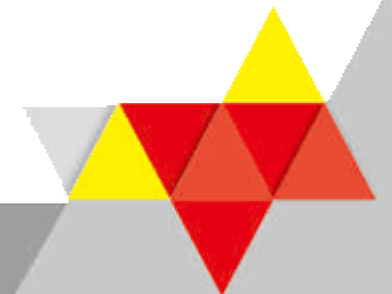
did_swap: true





An Animated Example

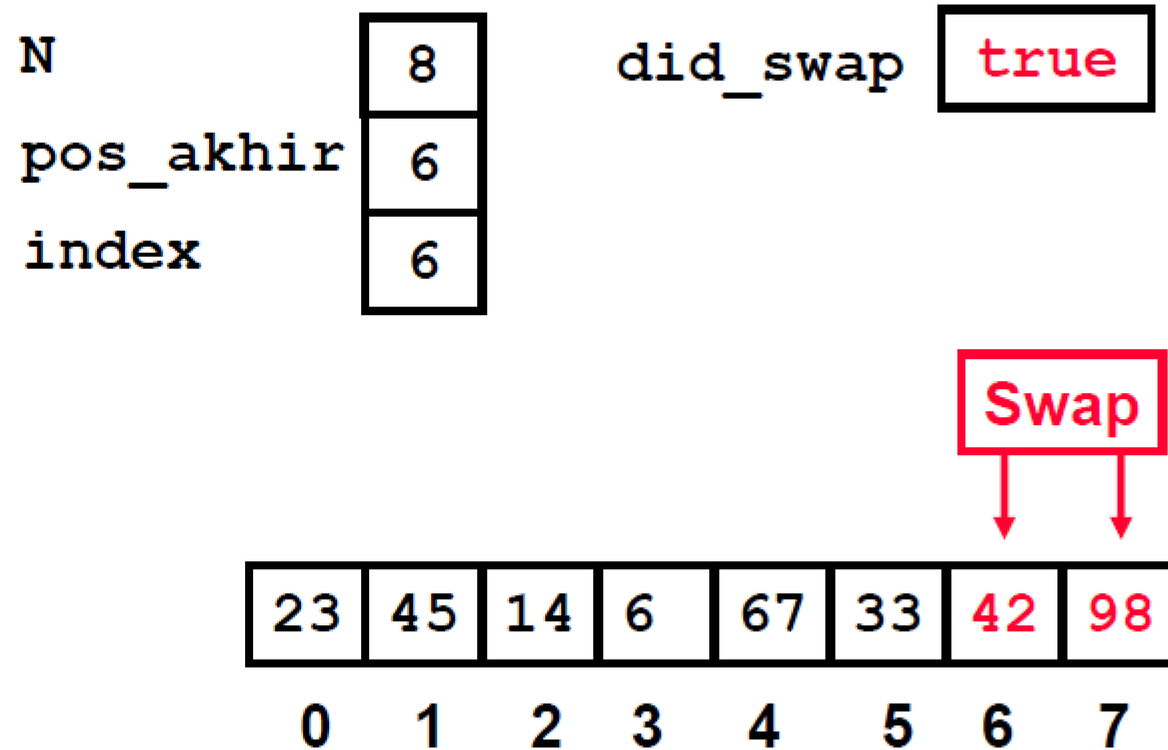


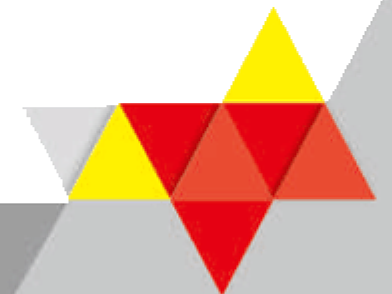


Universitas Dinamika

BUBBLE SORT

An Animated Example





UNIVERSITAS Dinamika

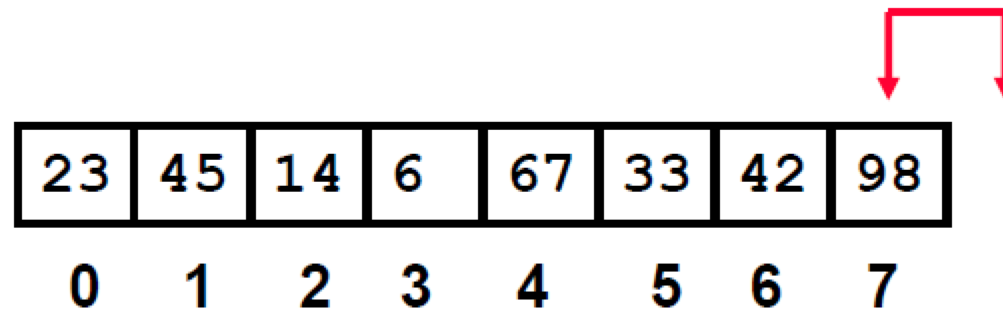
BUBBLE SORT

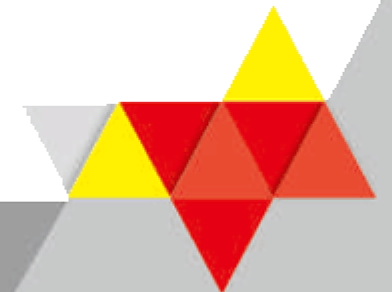
After First Pass of Outer Loop

N	8
pos_akhir	6
index	7

did_swap: true

Finished first "Bubble Up"





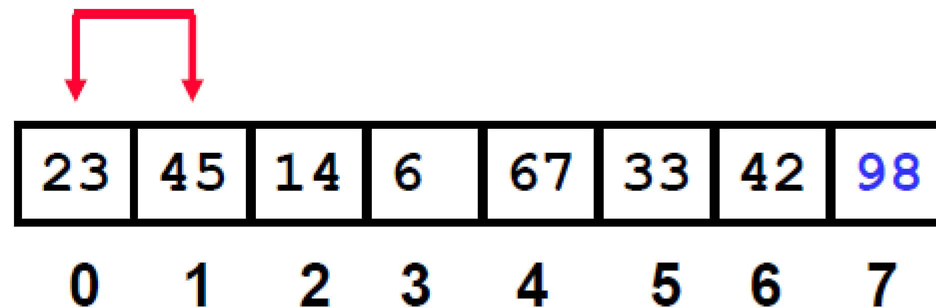
The Second “Bubble Up”

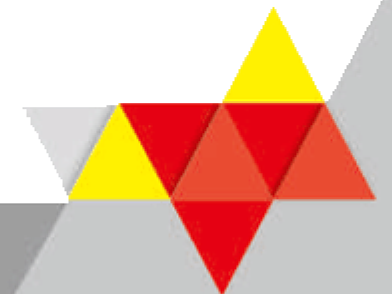
N 8

pos_akhir 5

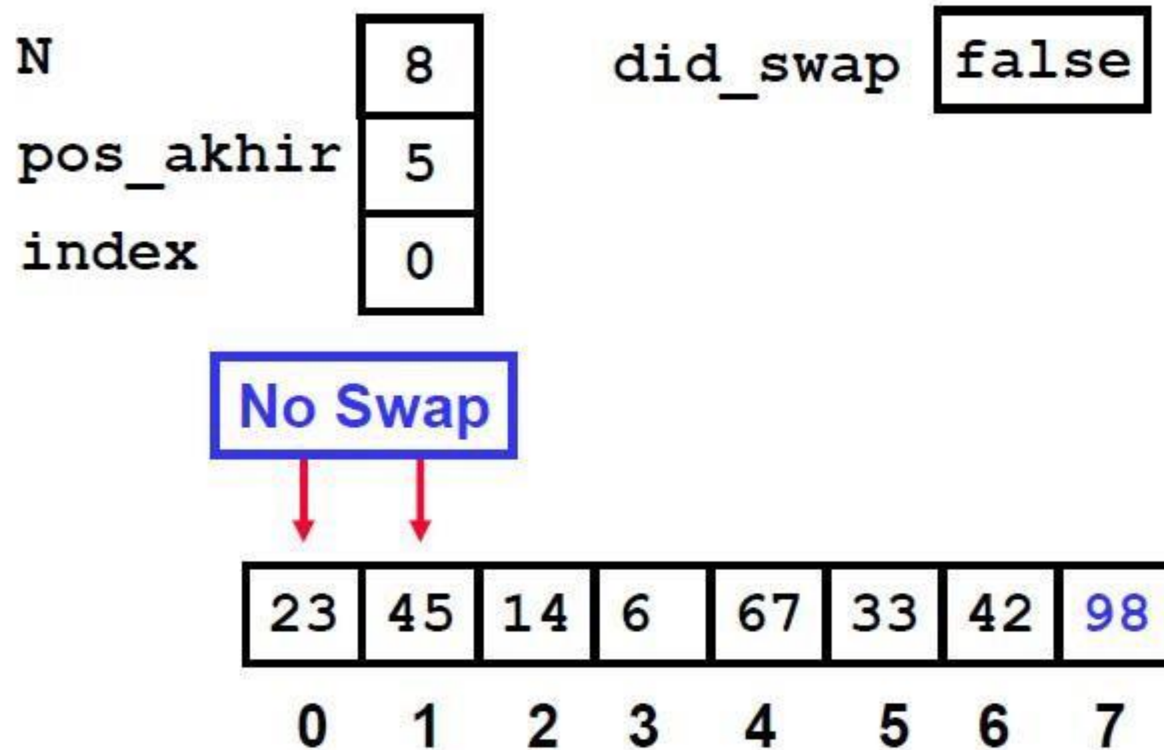
index 0

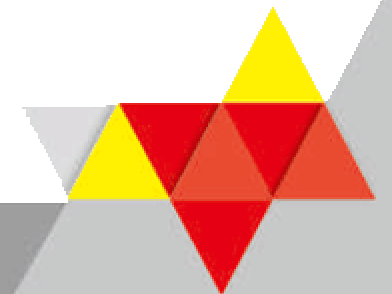
did_swap false





The Second "Bubble Up"

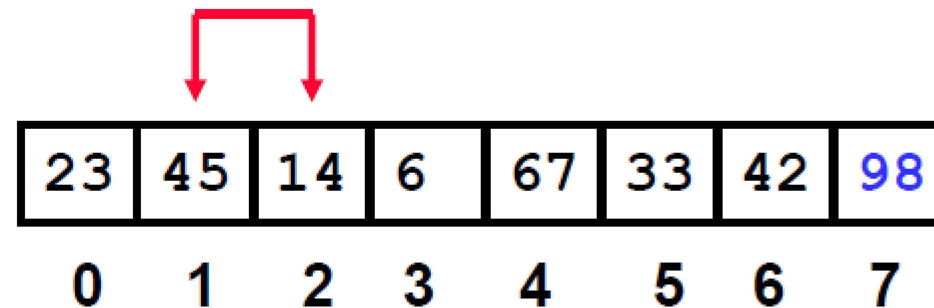


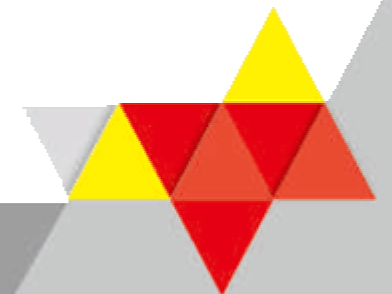


The Second “Bubble Up”

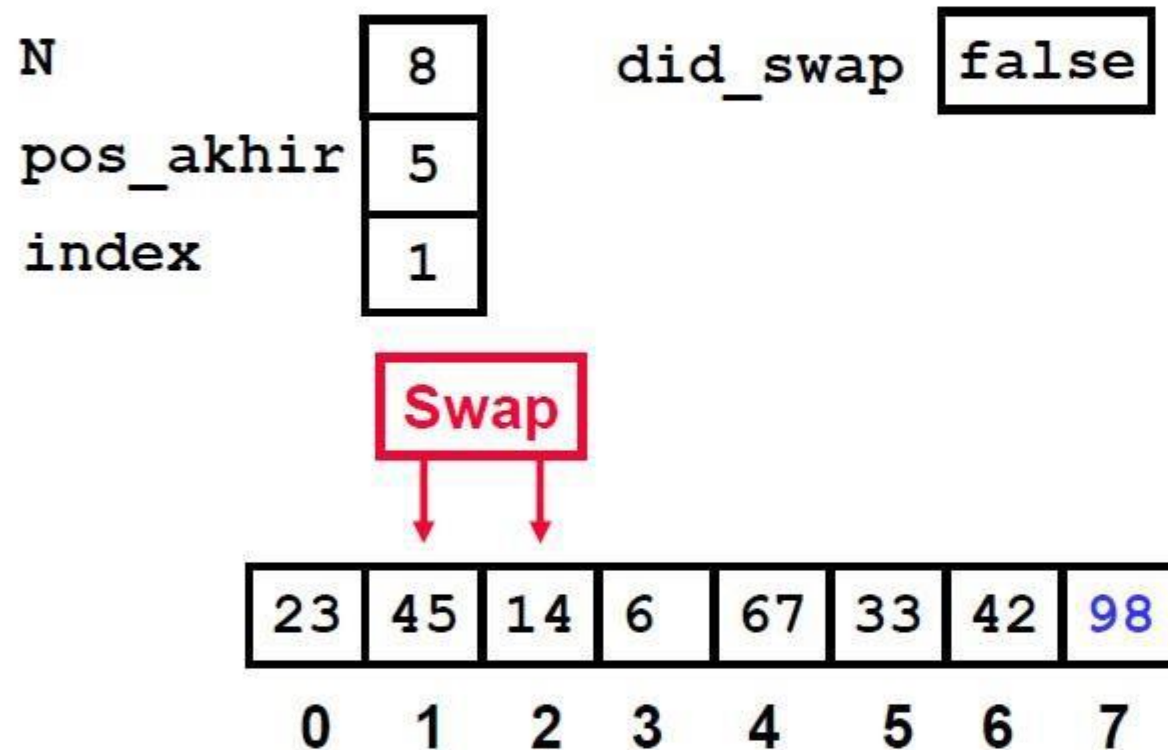
N	8
pos_akhir	5
index	1

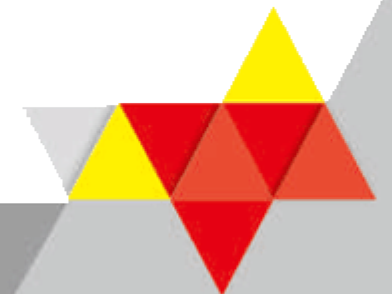
did_swap	false



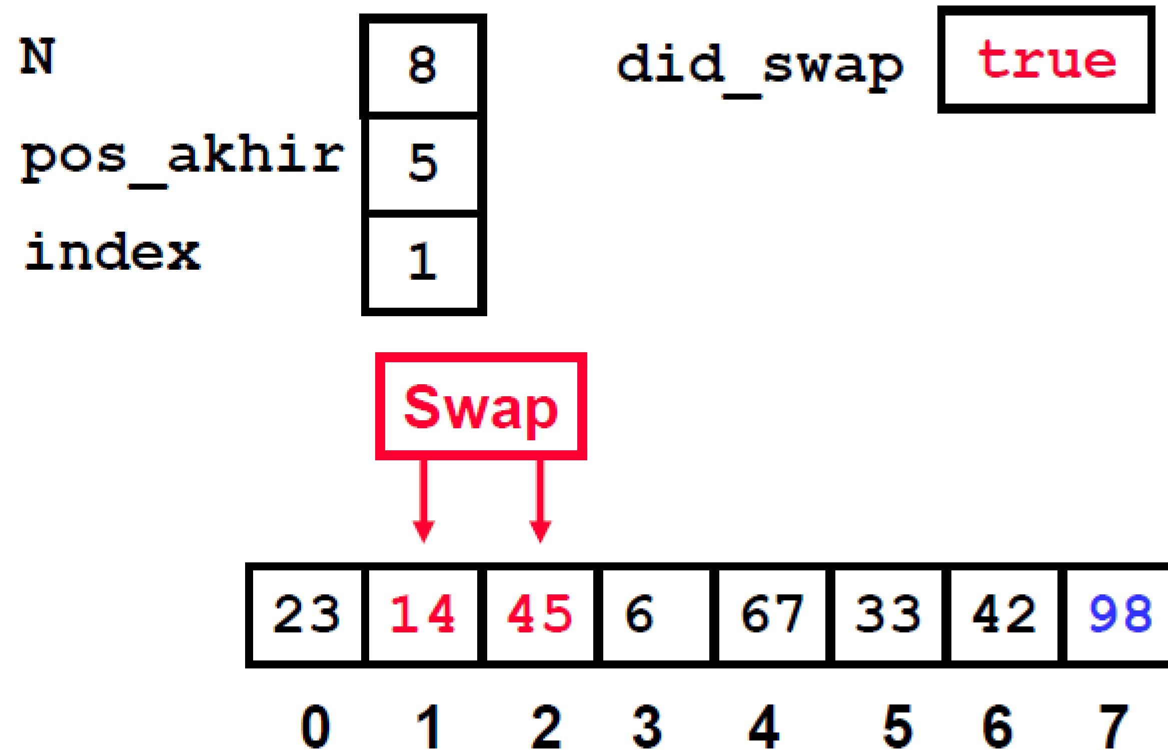


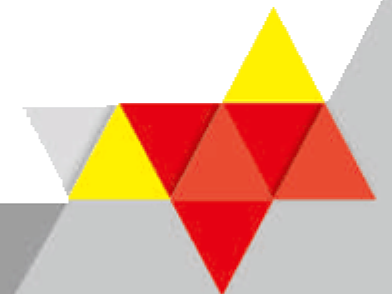
The Second "Bubble Up"





The Second "Bubble Up"





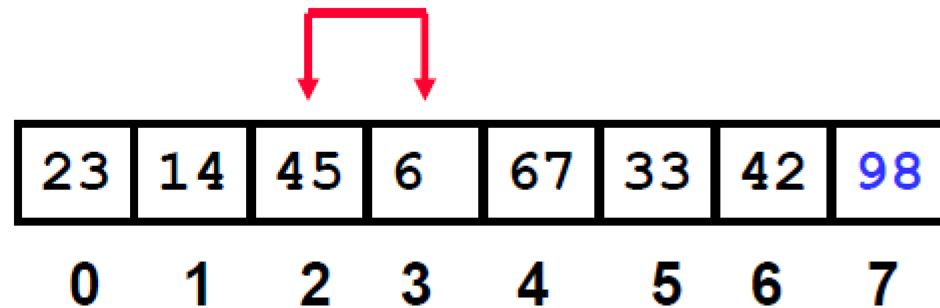
The Second "Bubble Up"

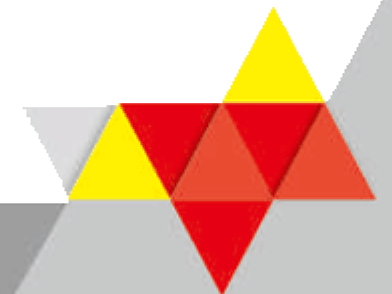
N 8

pos_akhir 5

index 2

did_swap true

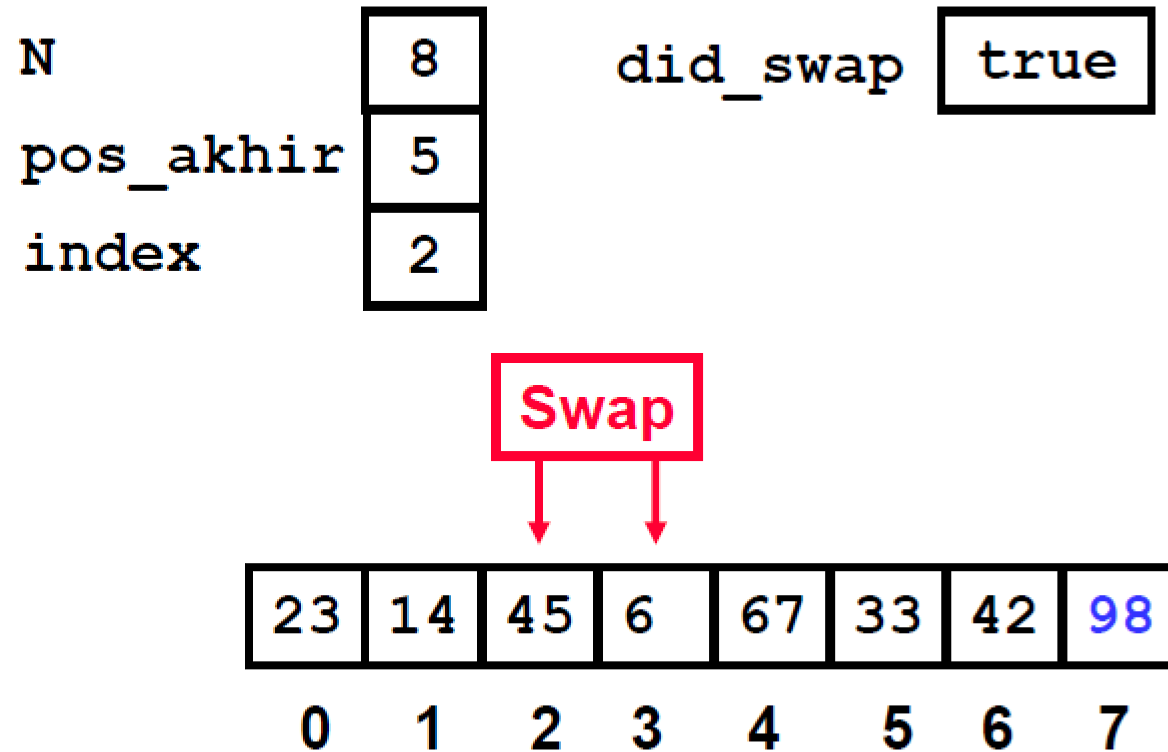


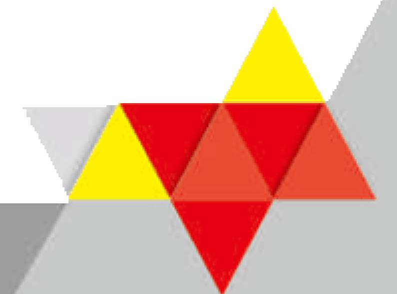


UNIVERSITAS Dinamika

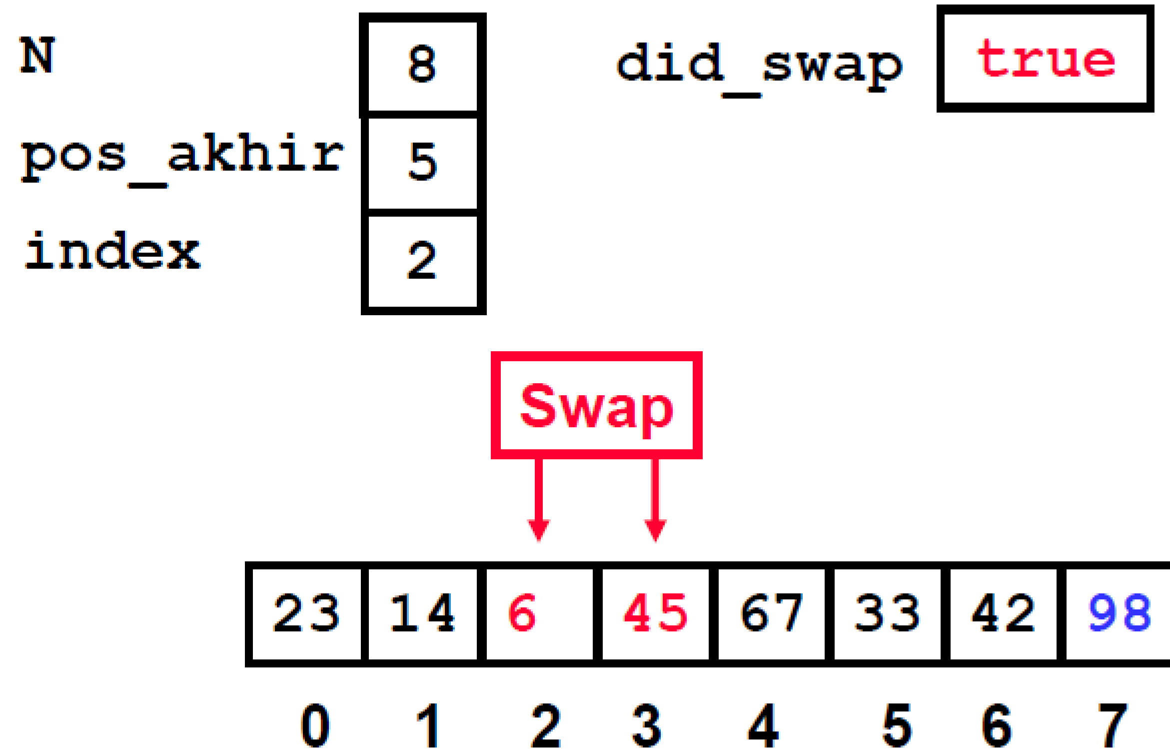
BUBBLE SORT

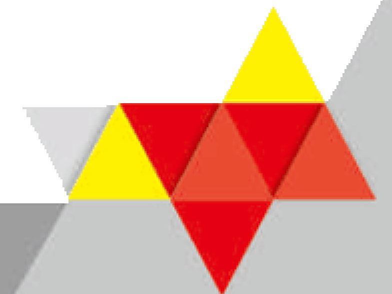
The Second "Bubble Up"





The Second "Bubble Up"

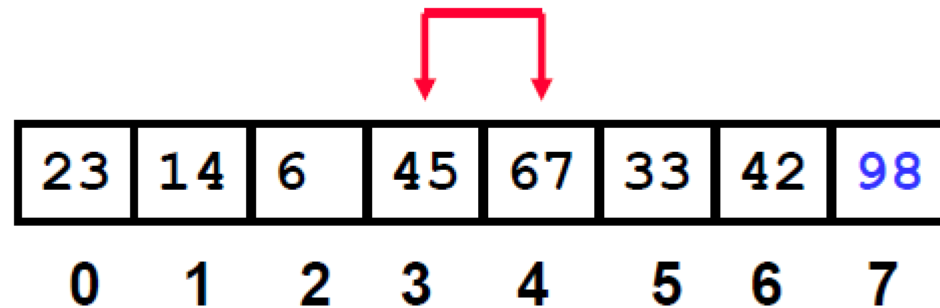


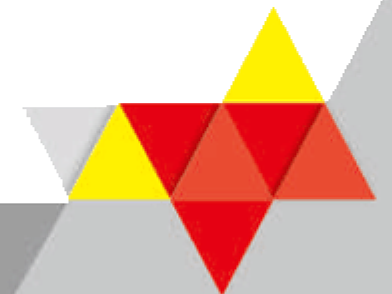


The Second “Bubble Up”

N	8
pos_akhir	5
index	3

did_swap	true





UNIVERSITAS Dinamika

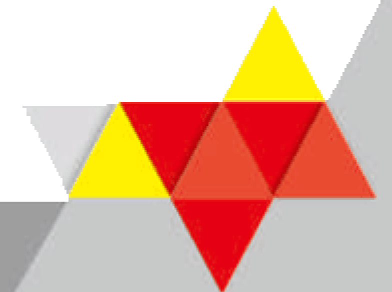
The Second “Bubble Up”

N	8
pos_akhir	5
index	3

did_swap: true

No Swap

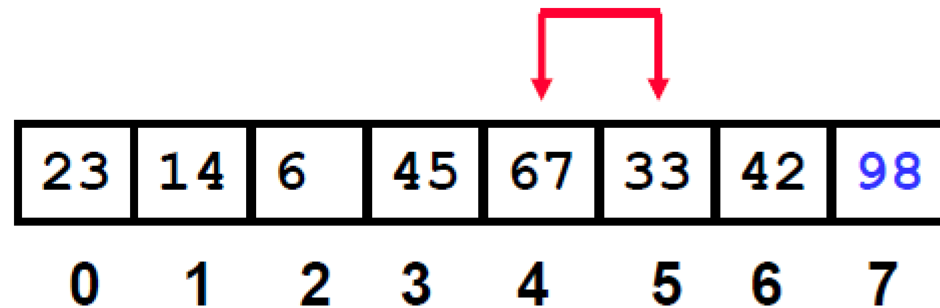
23	14	6	45	67	33	42	98
0	1	2	3	4	5	6	7

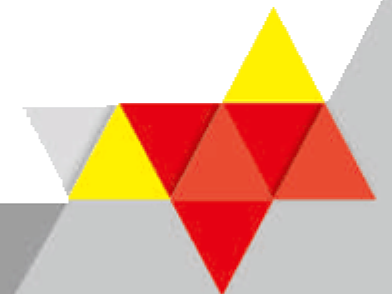


The Second “Bubble Up”

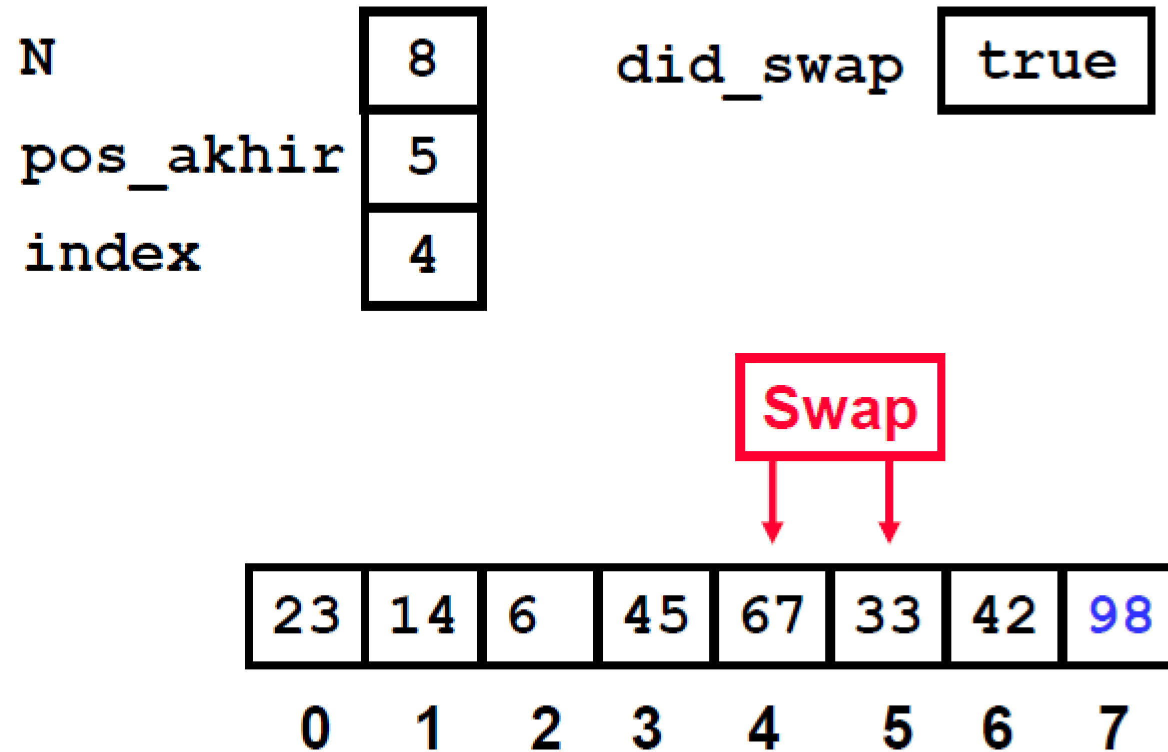
N	8
pos_akhir	5
index	4

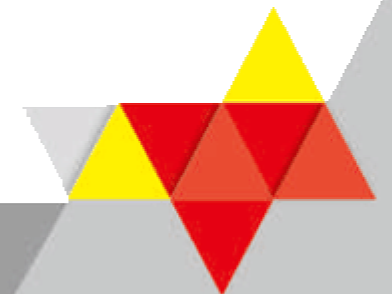
did_swap: true





The Second “Bubble Up”

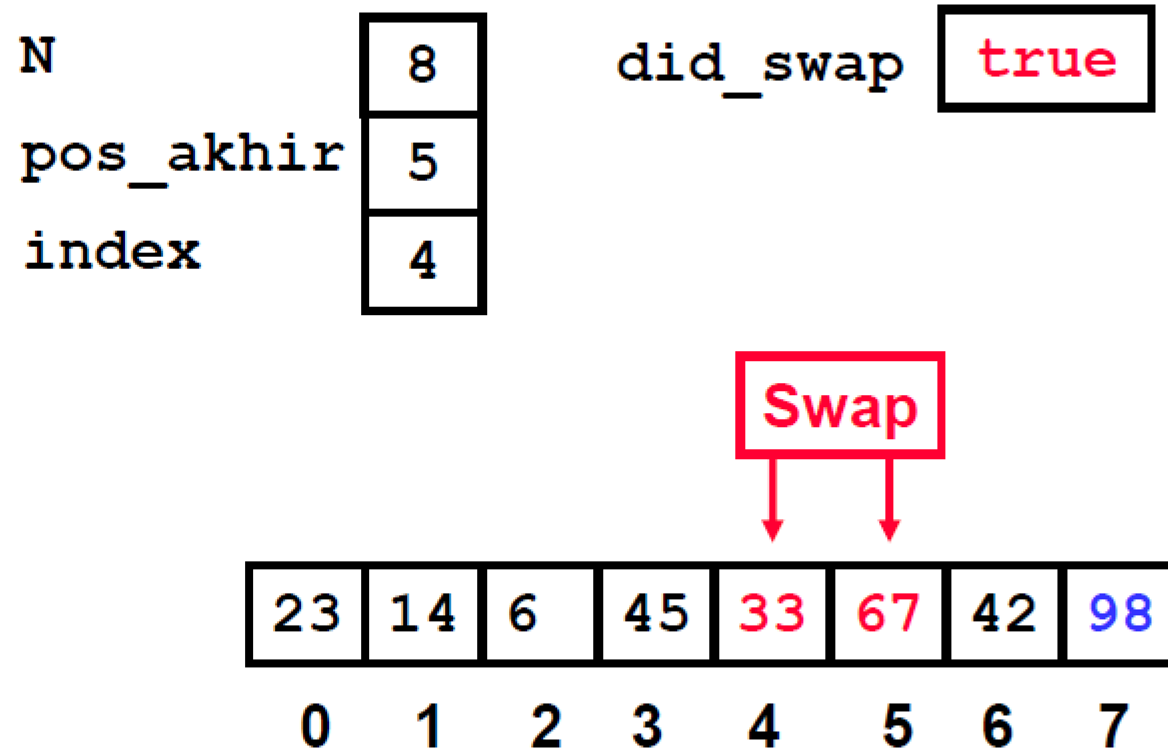


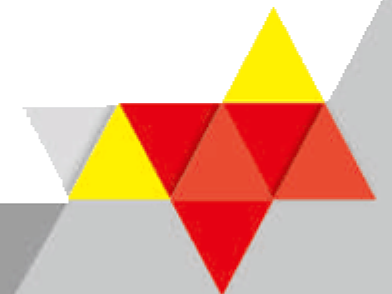


Dinamika UNIVERSITAS

BUBBLE SORT

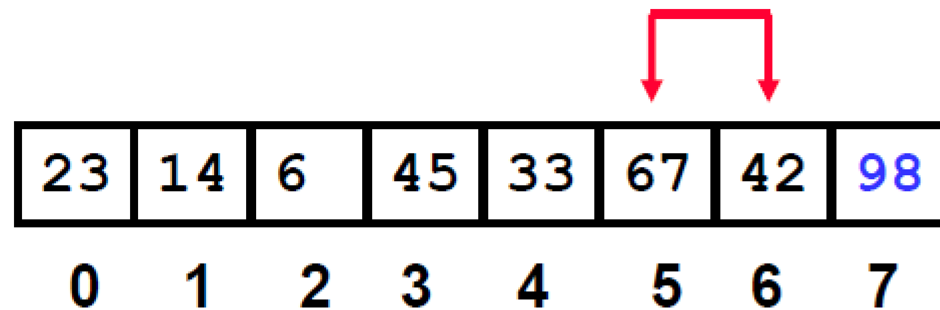
The Second "Bubble Up"

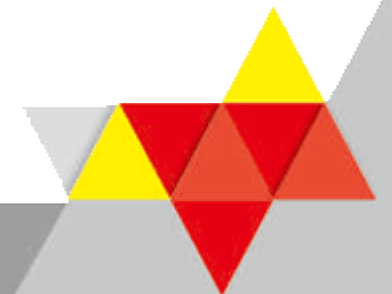




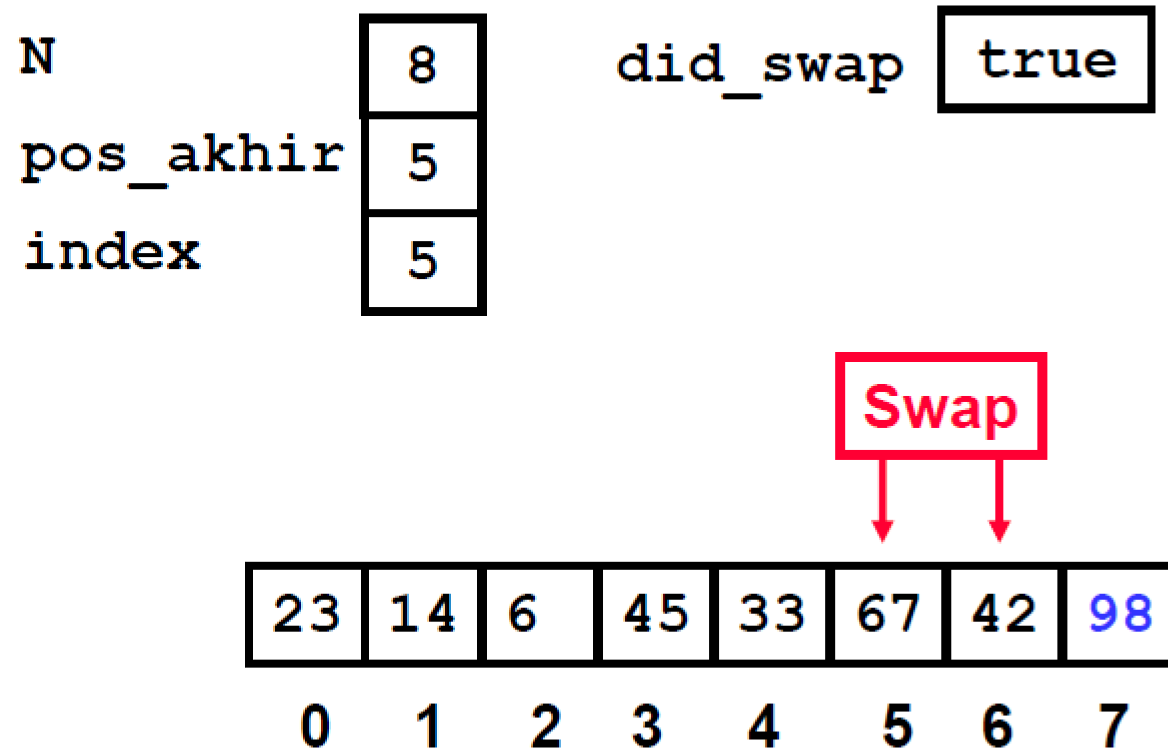
The Second “Bubble Up”

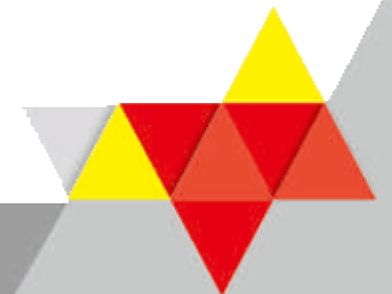
```
N            8
pos_akhir    5
index        5

did_swap   true
```





The Second “Bubble Up”



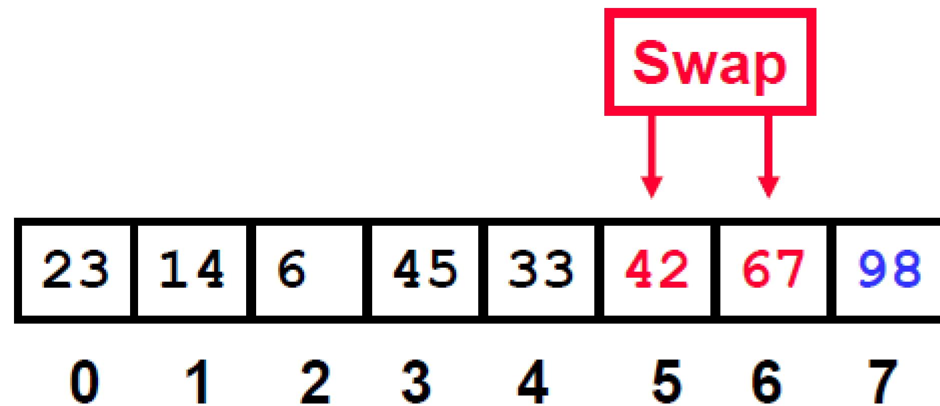


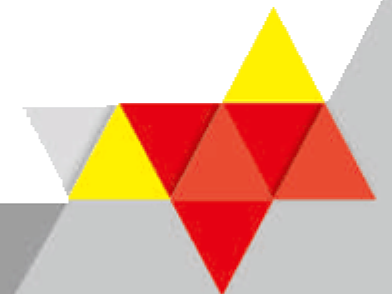
BUBBLE SORT

The Second “Bubble Up”

N	8
pos_akhir	5
index	5

did_swap: true





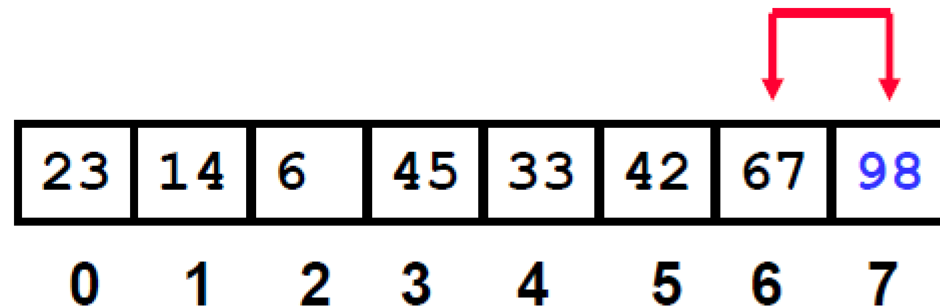
After Second Pass of Outer Loop

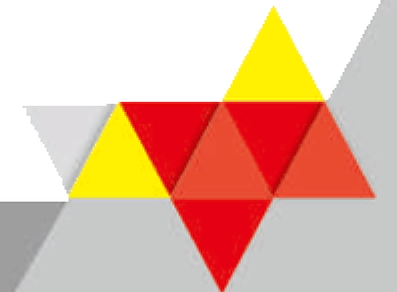
BUBBLE SORT

N	8
pos_akhir	5
index	6

did_swap: true

Finished second "Bubble Up"





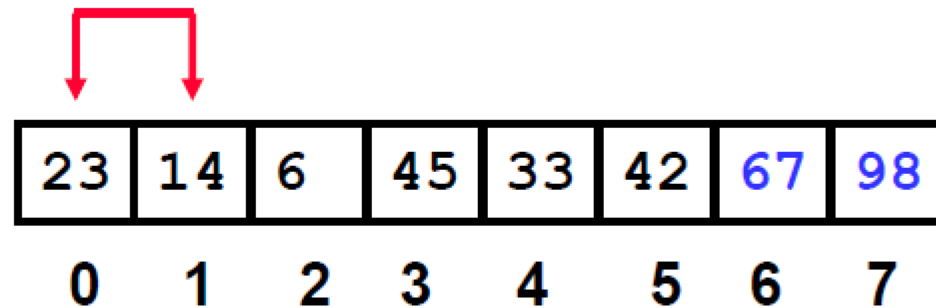
The Third "Bubble Up"

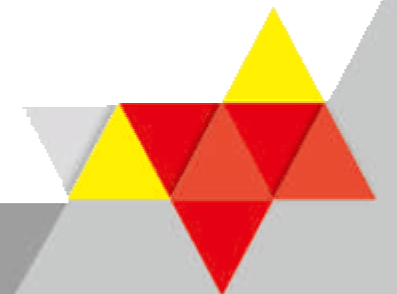
N 8

pos_akhir 4

index 0

did_swap false

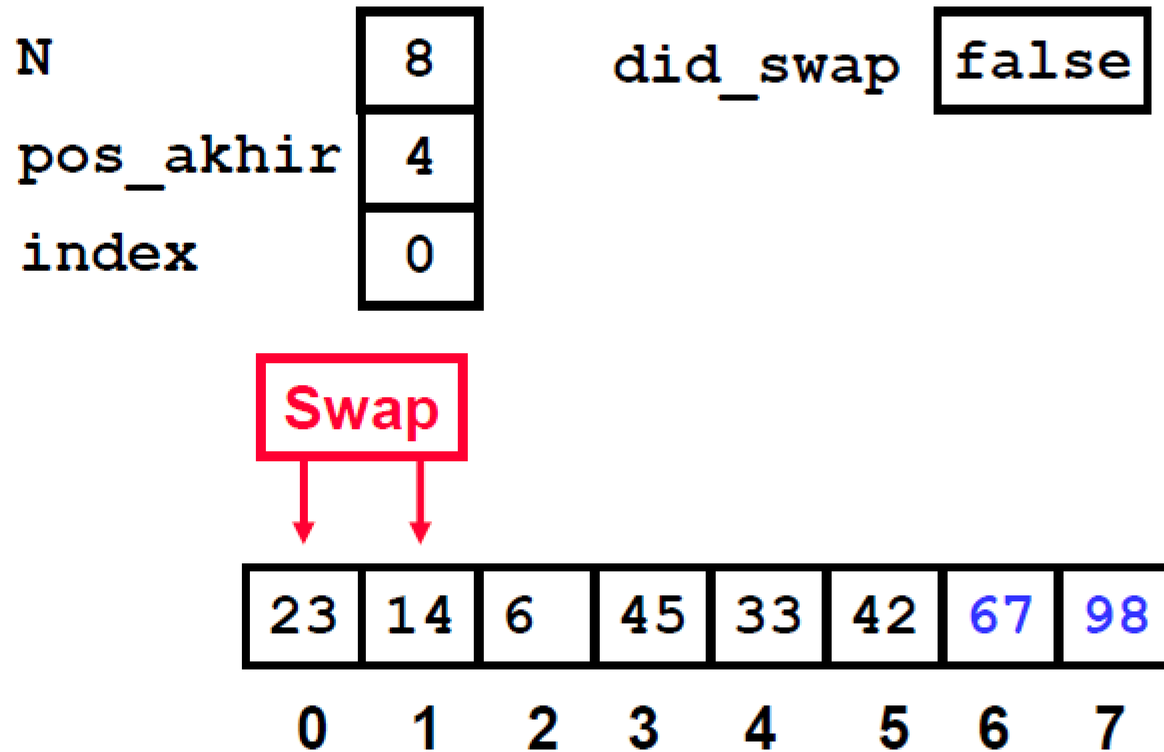


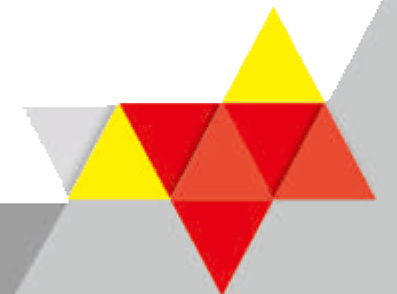


Universitas Dinamika

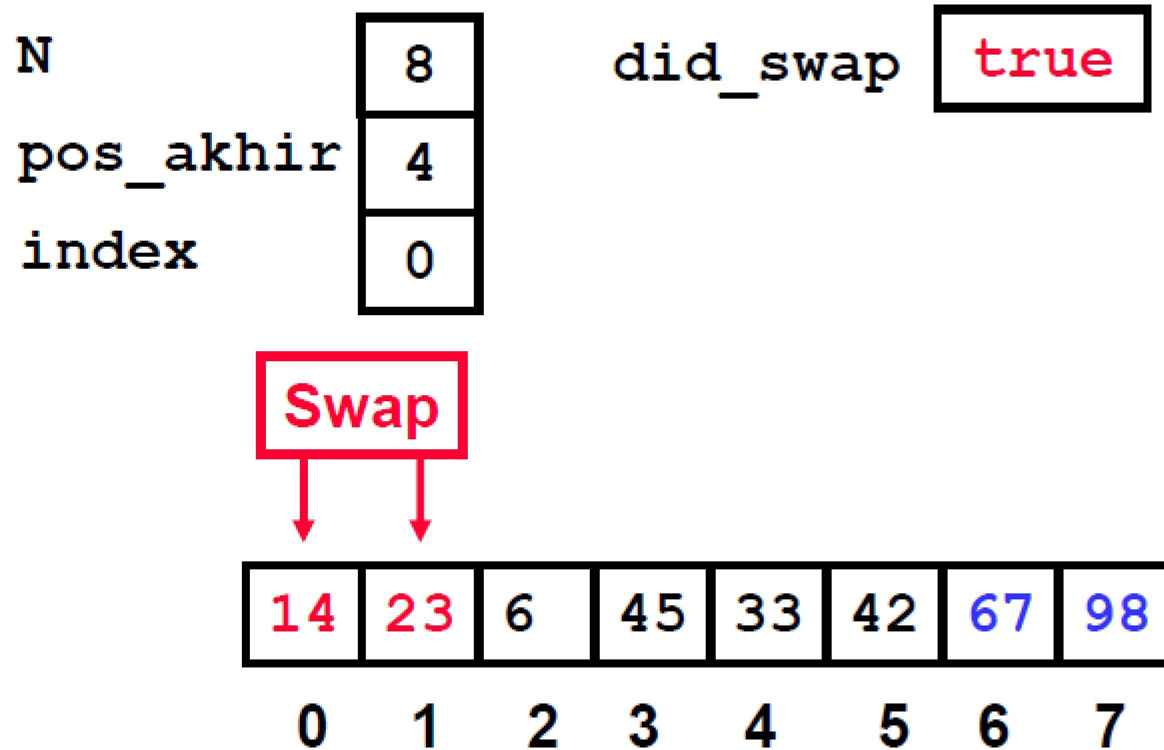
BUBBLE SORT

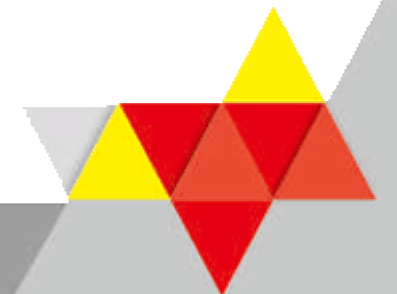
The Third "Bubble Up"





The Third "Bubble Up"

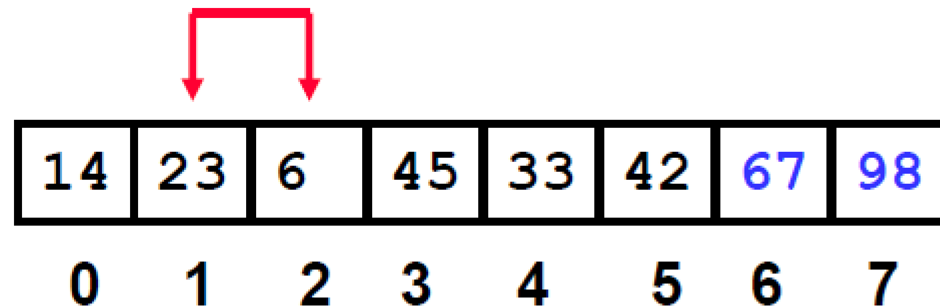


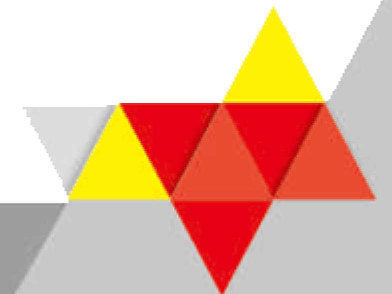


The Third "Bubble Up"

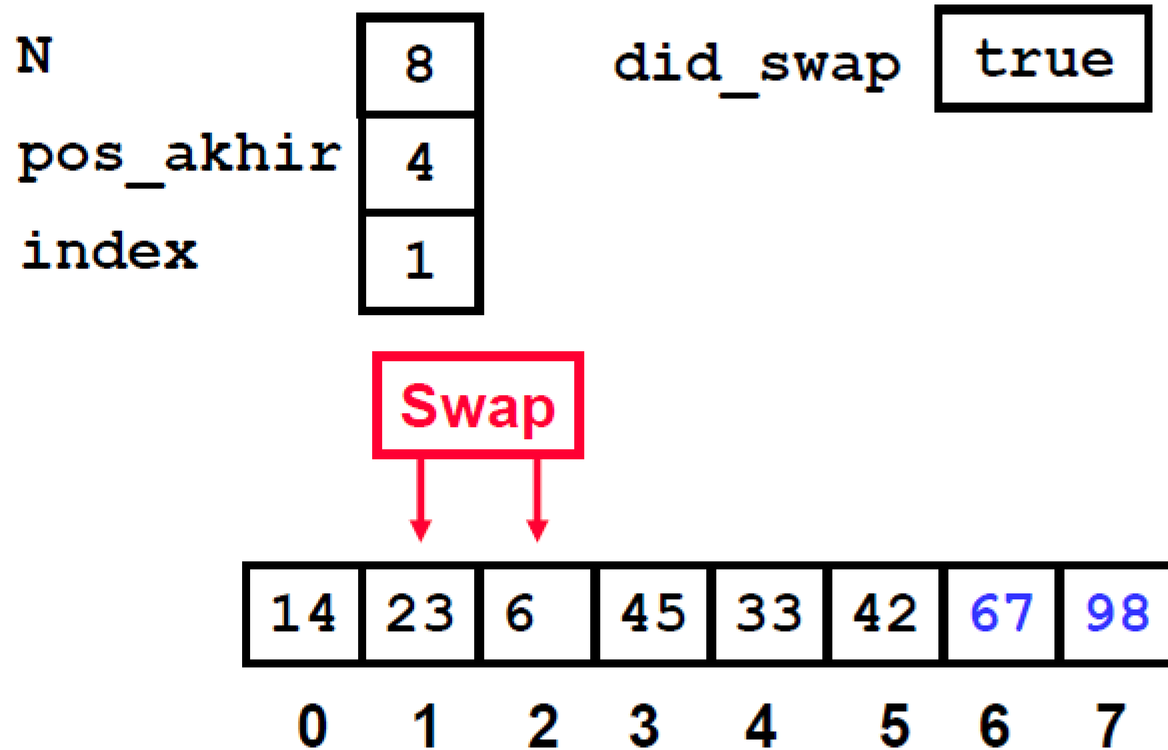
N	8
pos_akhir	4
index	1

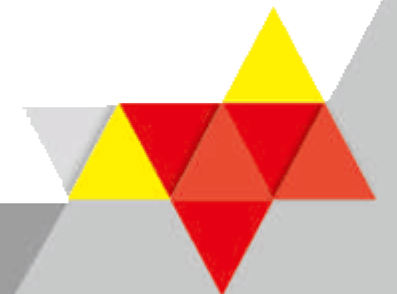
did_swap: true





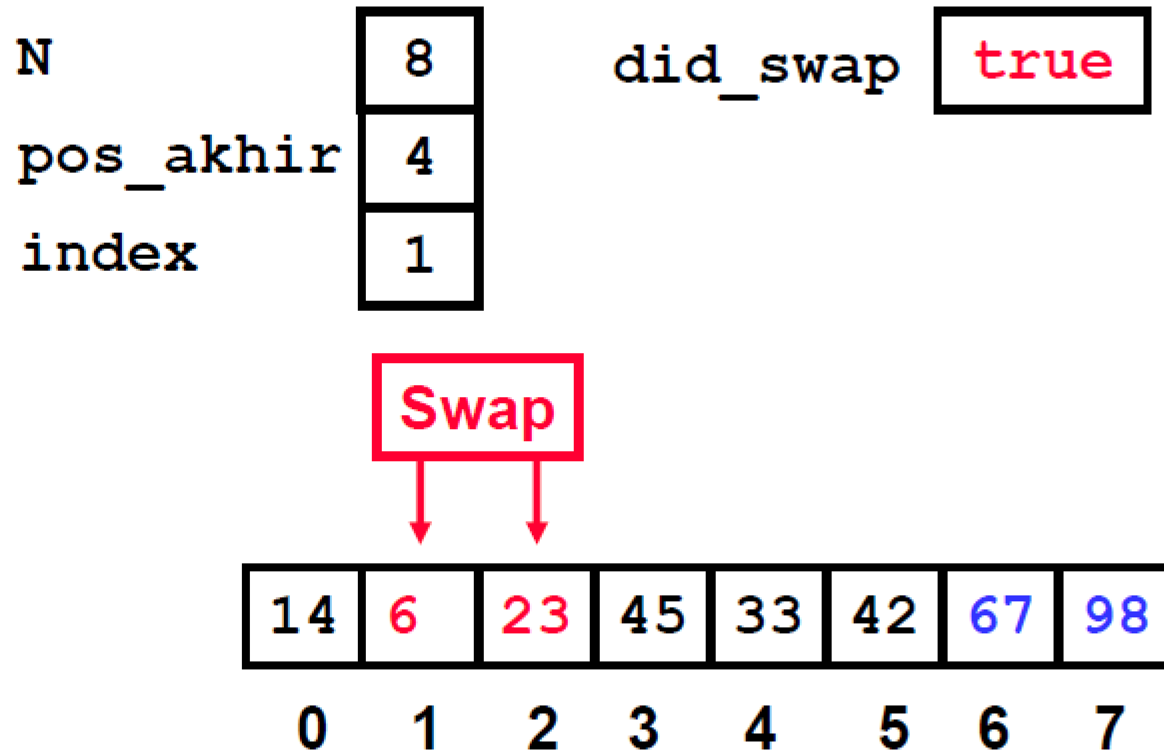
The Third "Bubble Up"

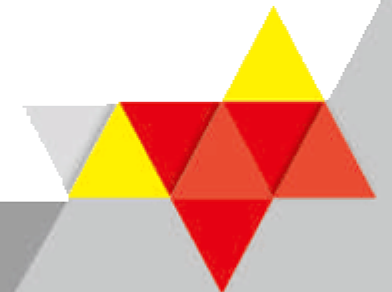




BUBBLE SORT

The Third "Bubble Up"

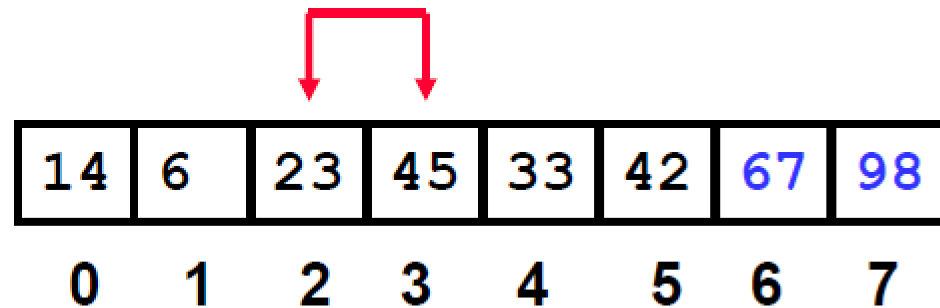


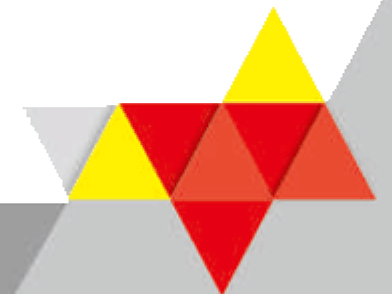


The Third “Bubble Up”

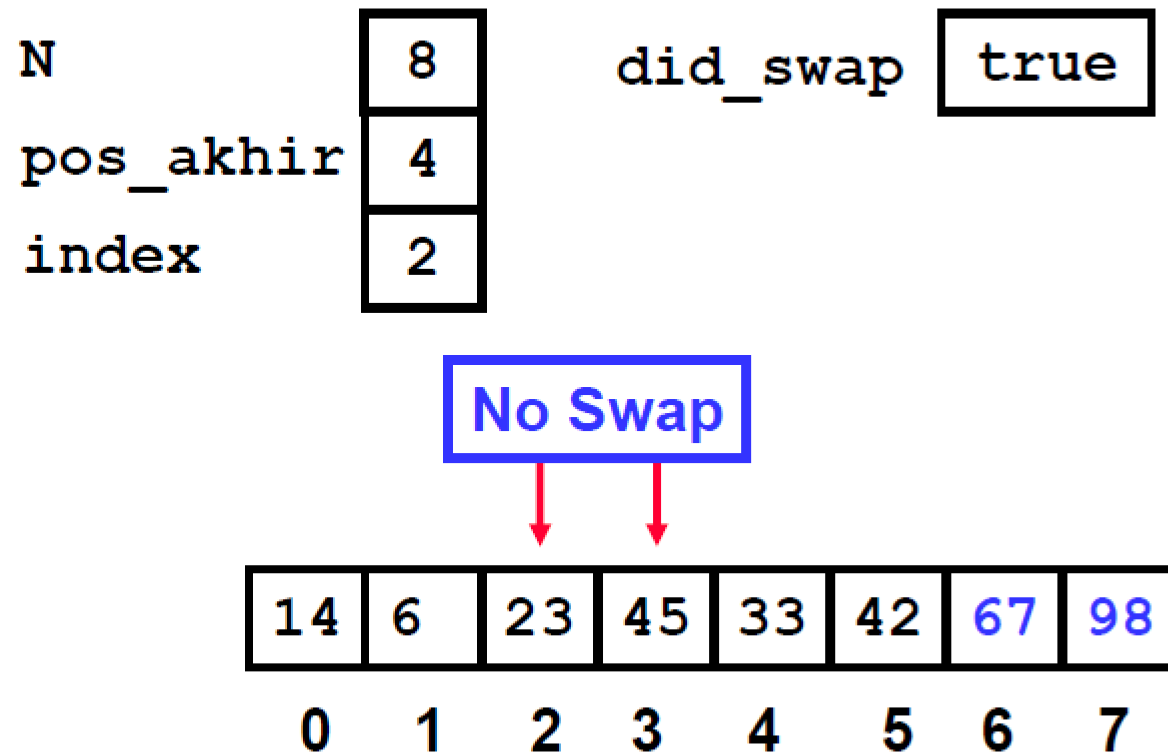
N	8
pos_akhir	4
index	2

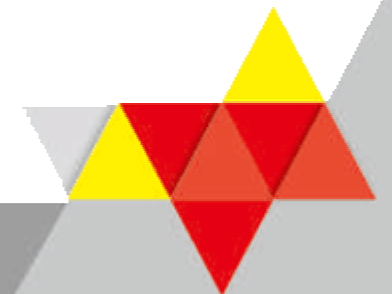
did_swap: true





The Third "Bubble Up"

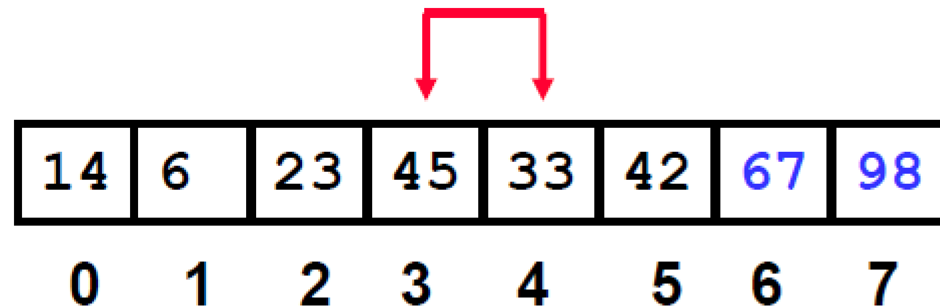


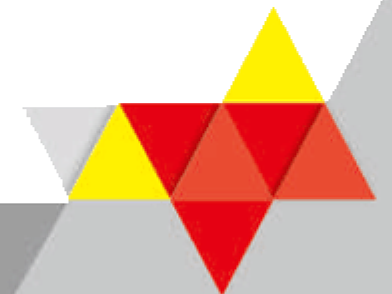


The Third “Bubble Up”

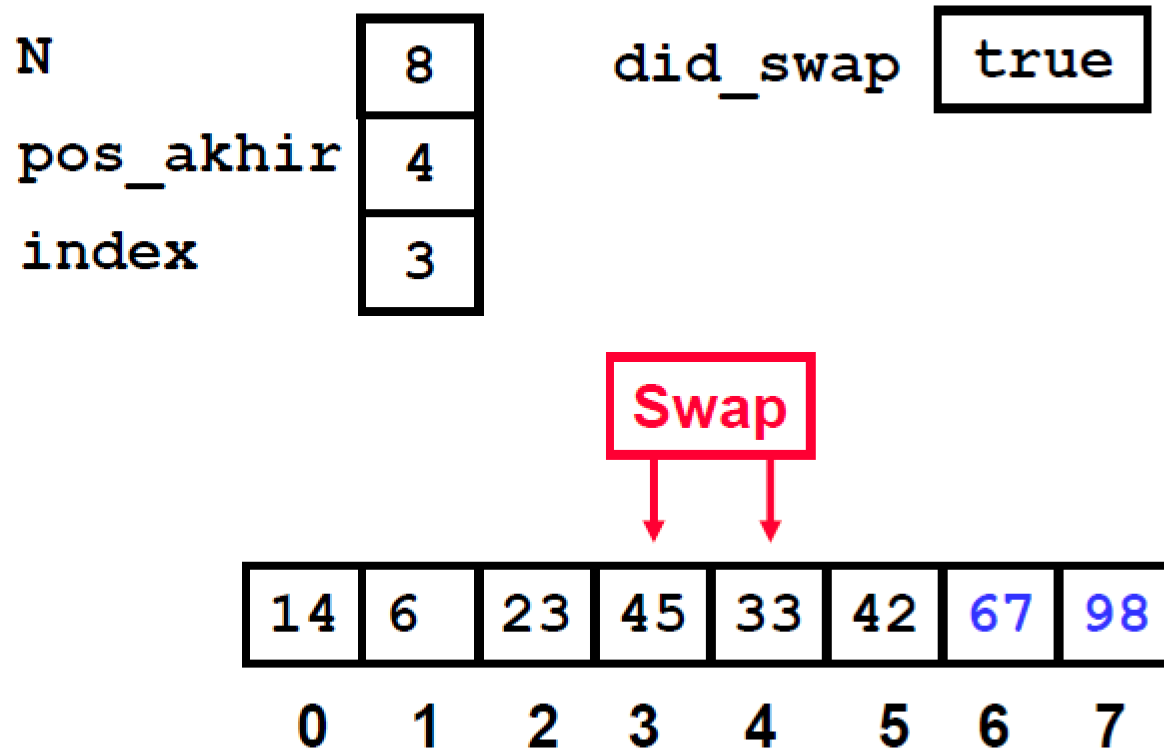
N	8
pos_akhir	4
index	3

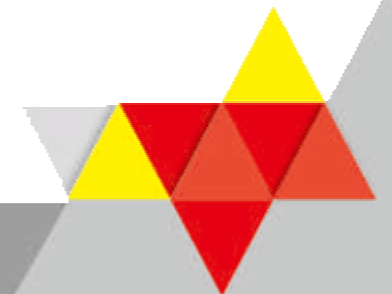
did_swap	true



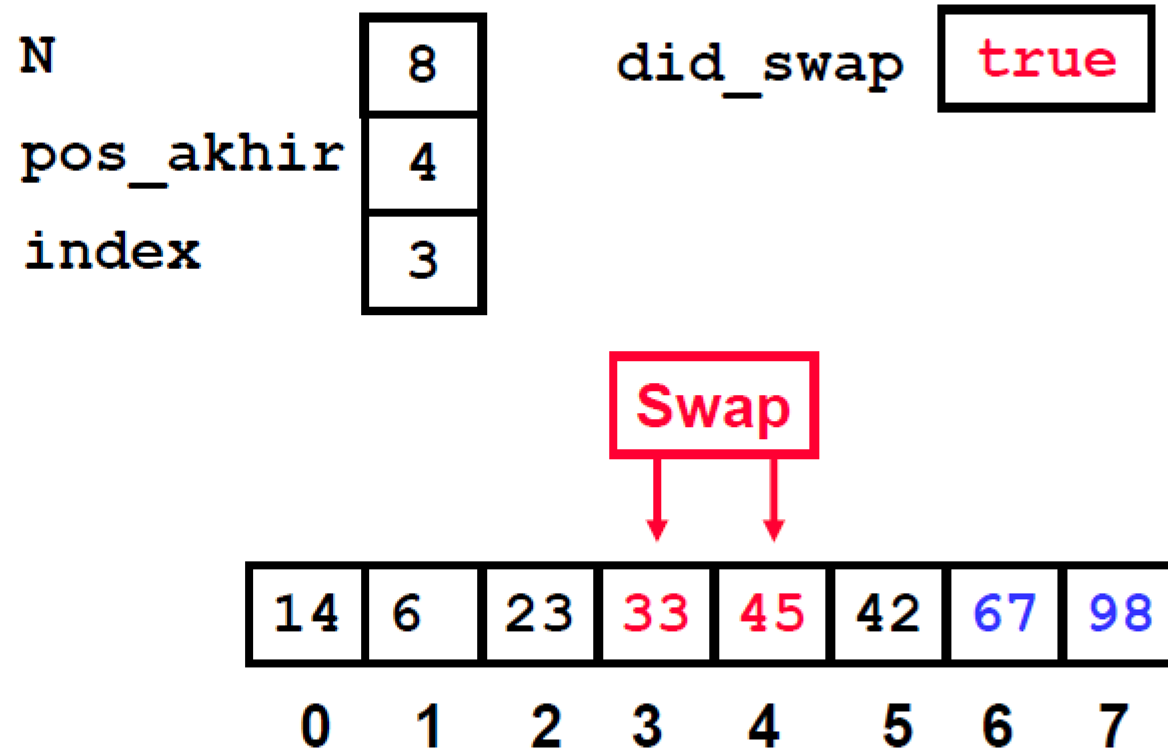


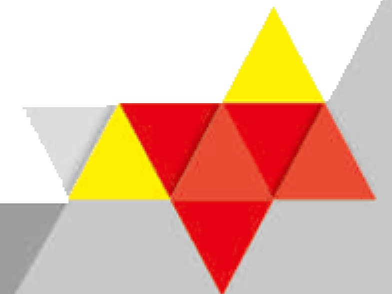
The Third "Bubble Up"





The Third "Bubble Up"

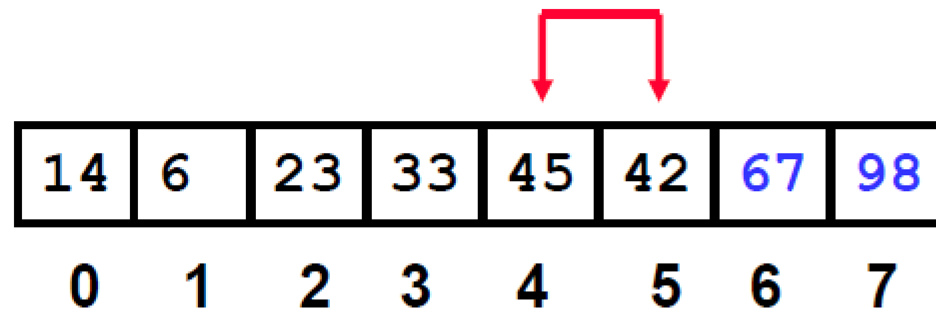


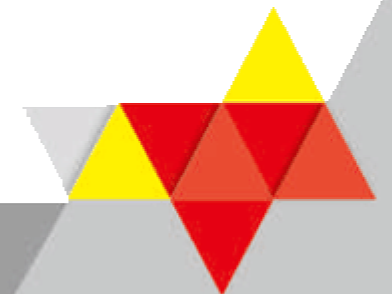


The Third “Bubble Up”

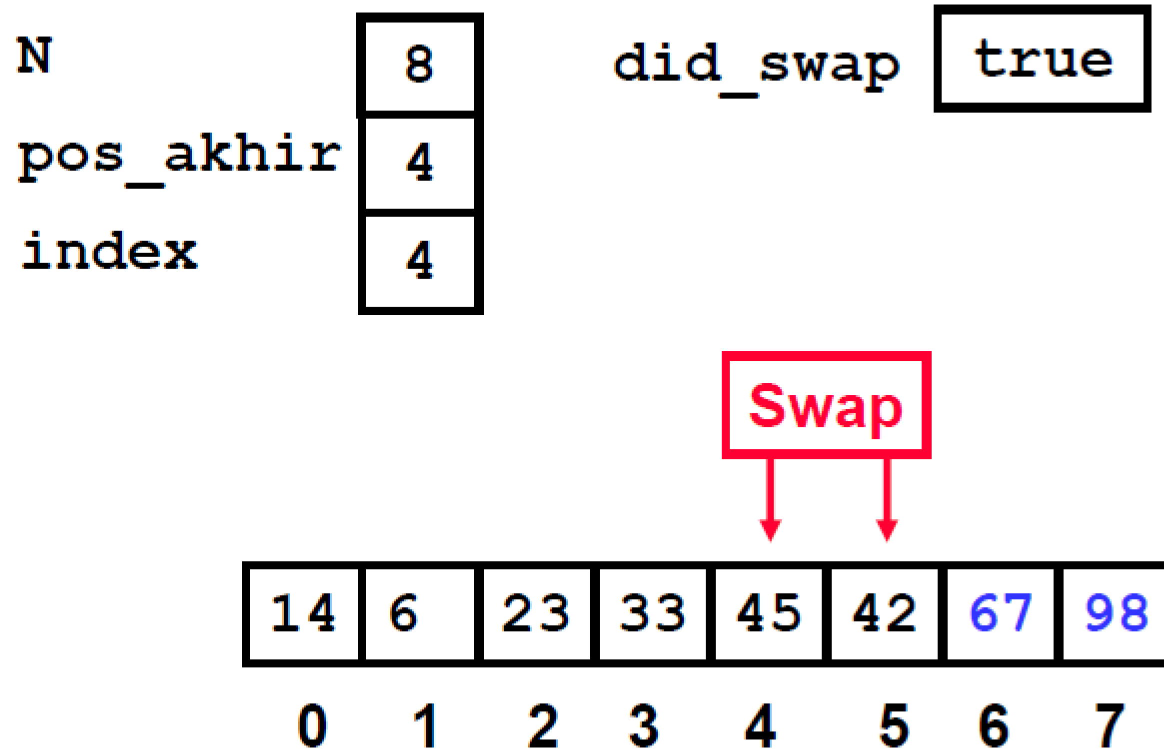
N	8
pos_akhir	4
index	4

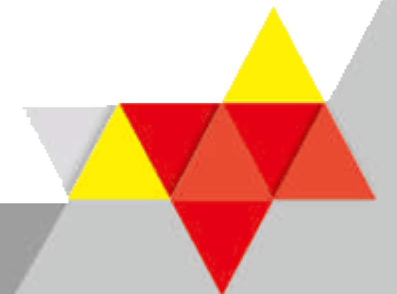
did_swap: true





The Third "Bubble Up"





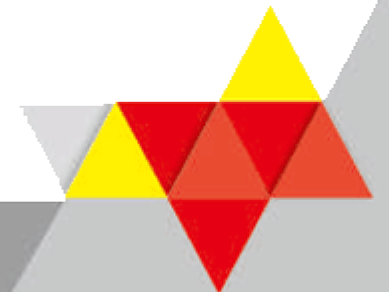
The Third “Bubble Up”

N	8
pos_akhir	4
index	4

did_swap: true

Swap (42 ↔ 45)

14	6	23	33	42	45	67	98
0	1	2	3	4	5	6	7



UNIVERSITAS Dinamika

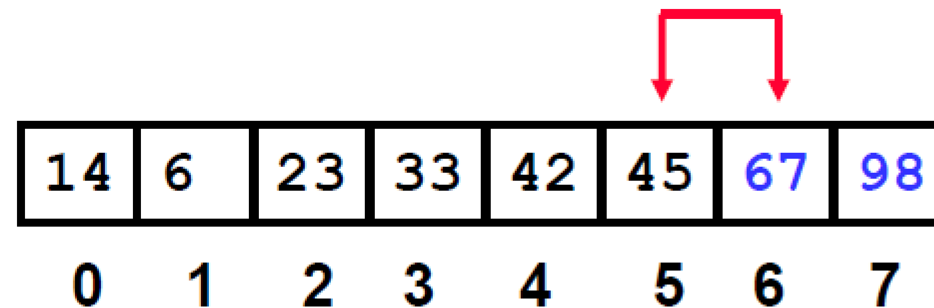
BUBBLE SORT

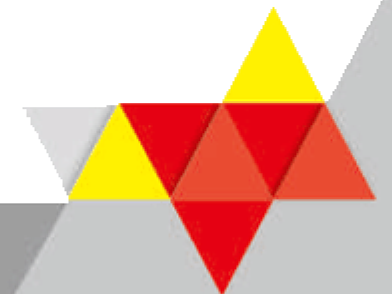
After Third Pass of Outer Loop

N	8
pos_akhir	4
index	5

did_swap: true

Finished third "Bubble Up"

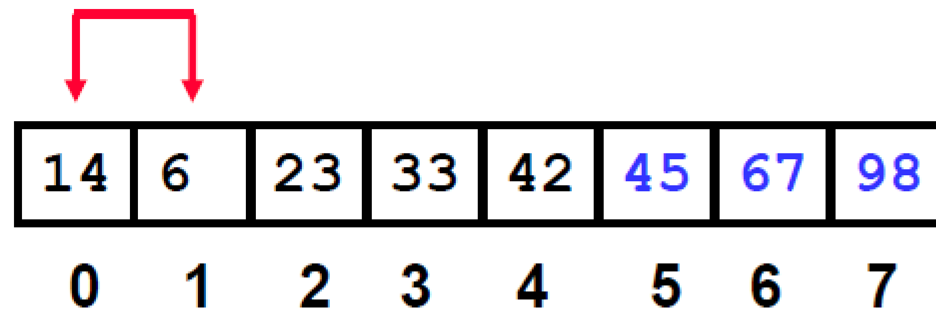


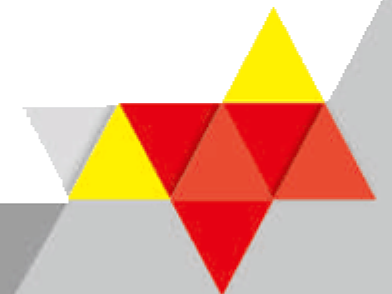


The Fourth “Bubble Up”

N	8
pos_akhir	3
index	0

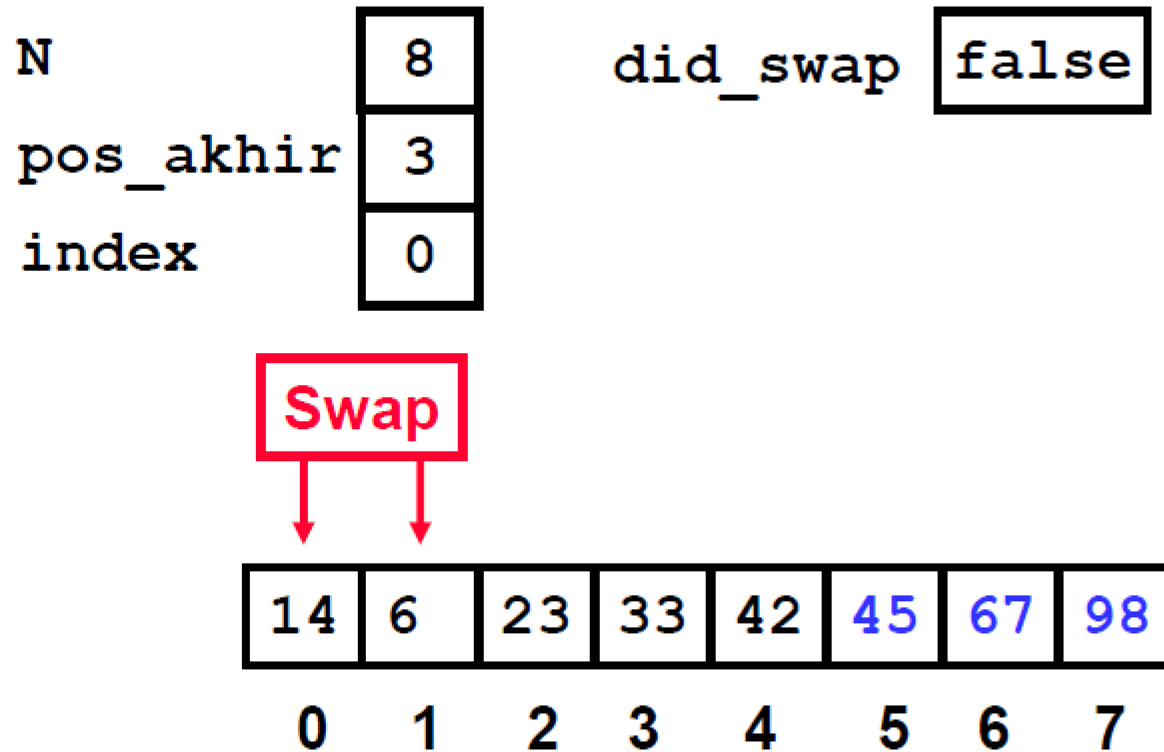
did_swap false

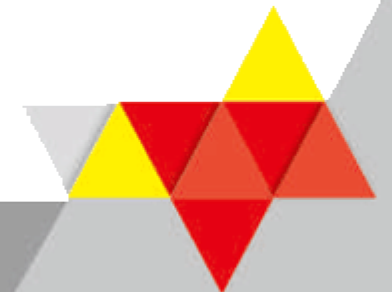




BUBBLE SORT

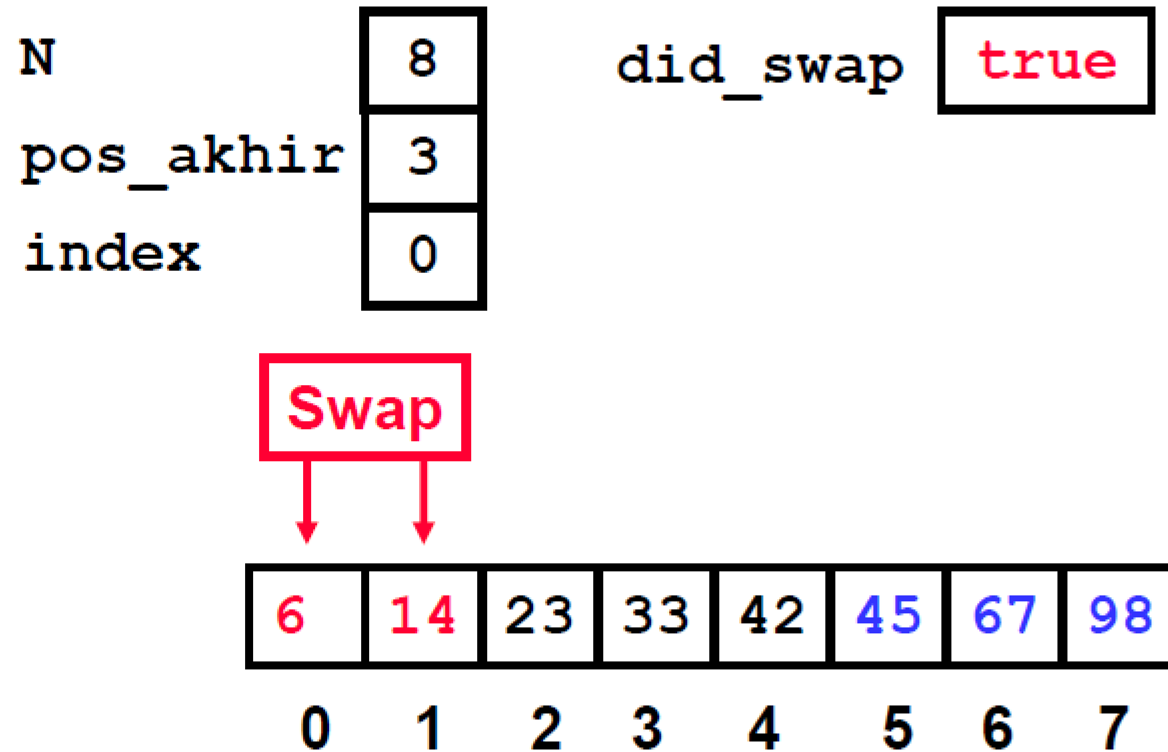
The Fourth “Bubble Up”

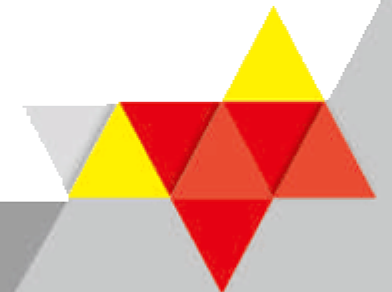


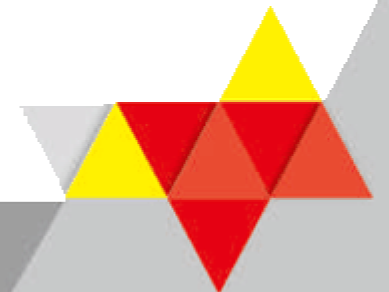


BUBBLE SORT

The Fourth “Bubble Up”





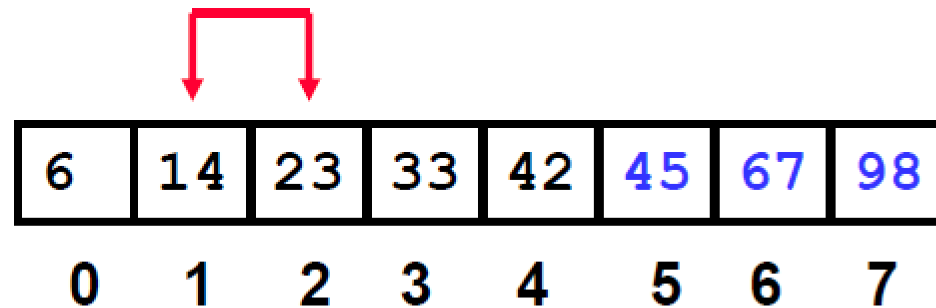


BUBBLE SORT

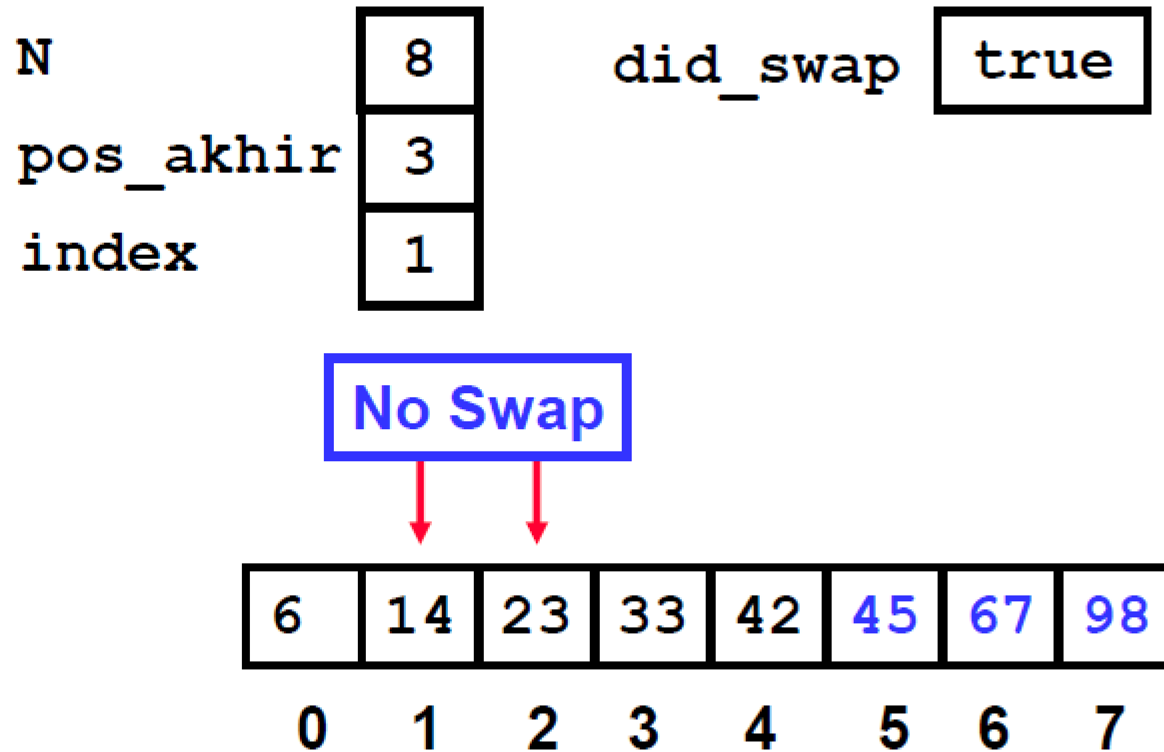
The Fourth "Bubble Up"

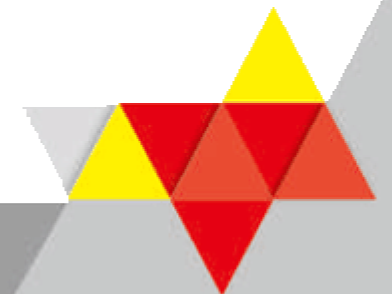
N	8
pos_akhir	3
index	1

did_swap: true



The Fourth “Bubble Up”

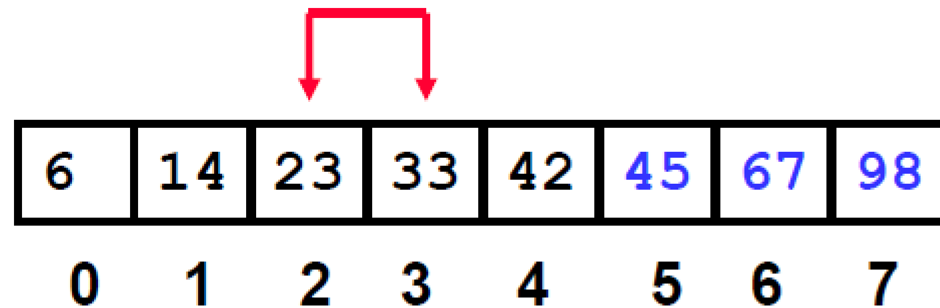


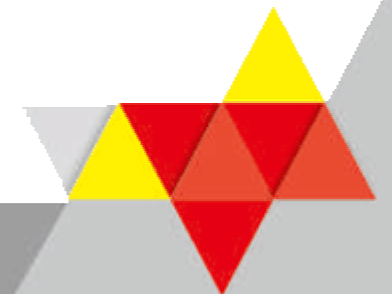


The Fourth "Bubble Up"

N	8
pos_akhir	3
index	2

did_swap: true

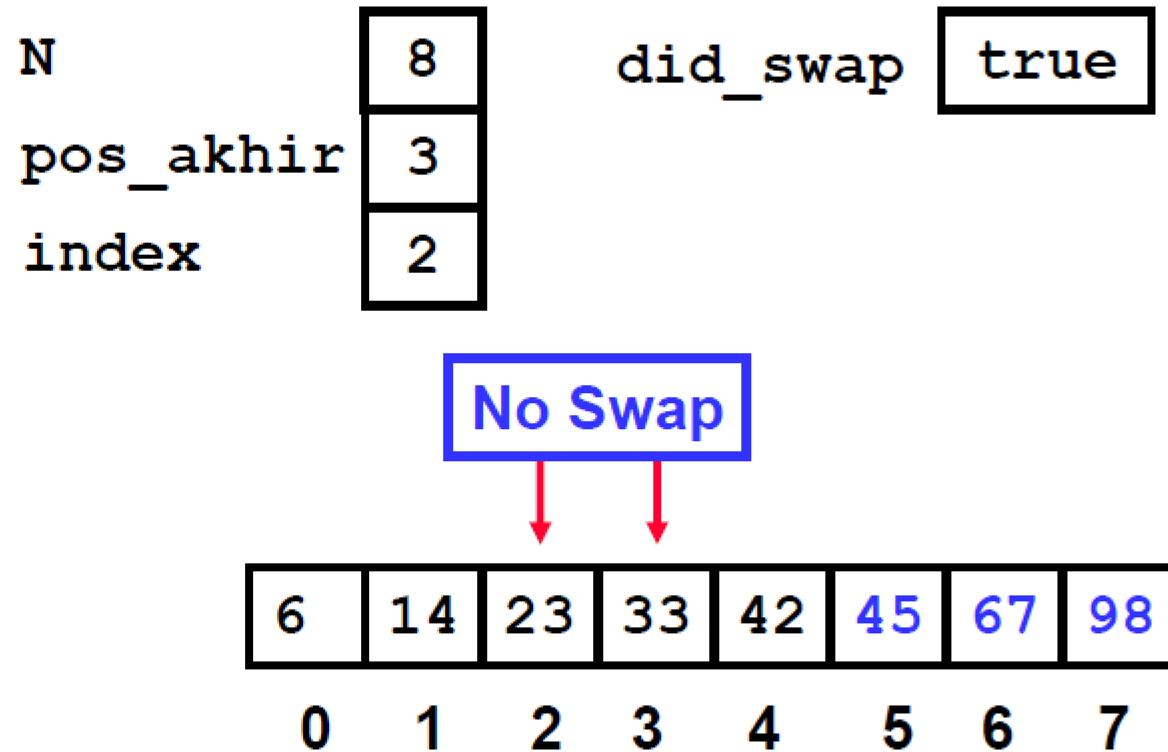


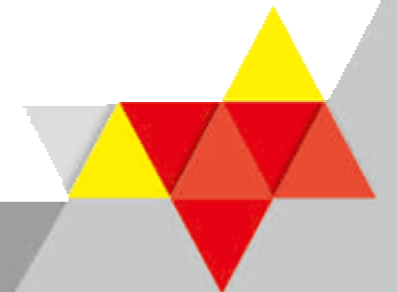


Universitas Dinamika

BUBBLE SORT

The Fourth "Bubble Up"

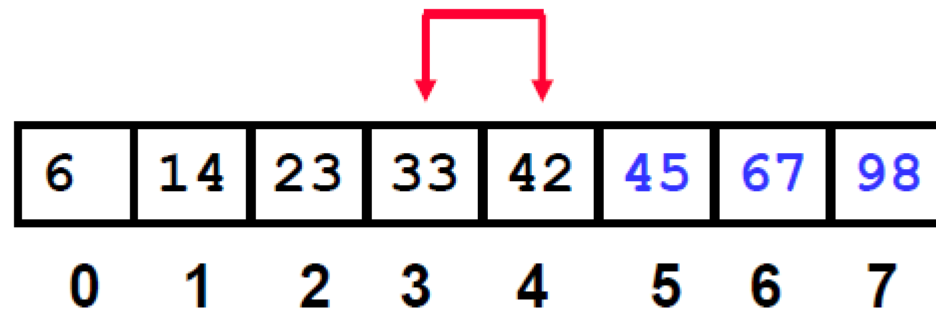


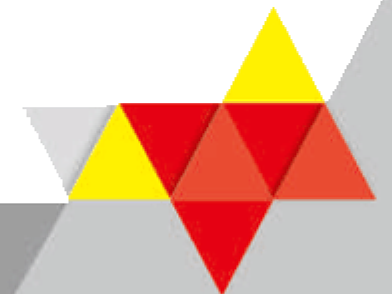


The Fourth "Bubble Up"

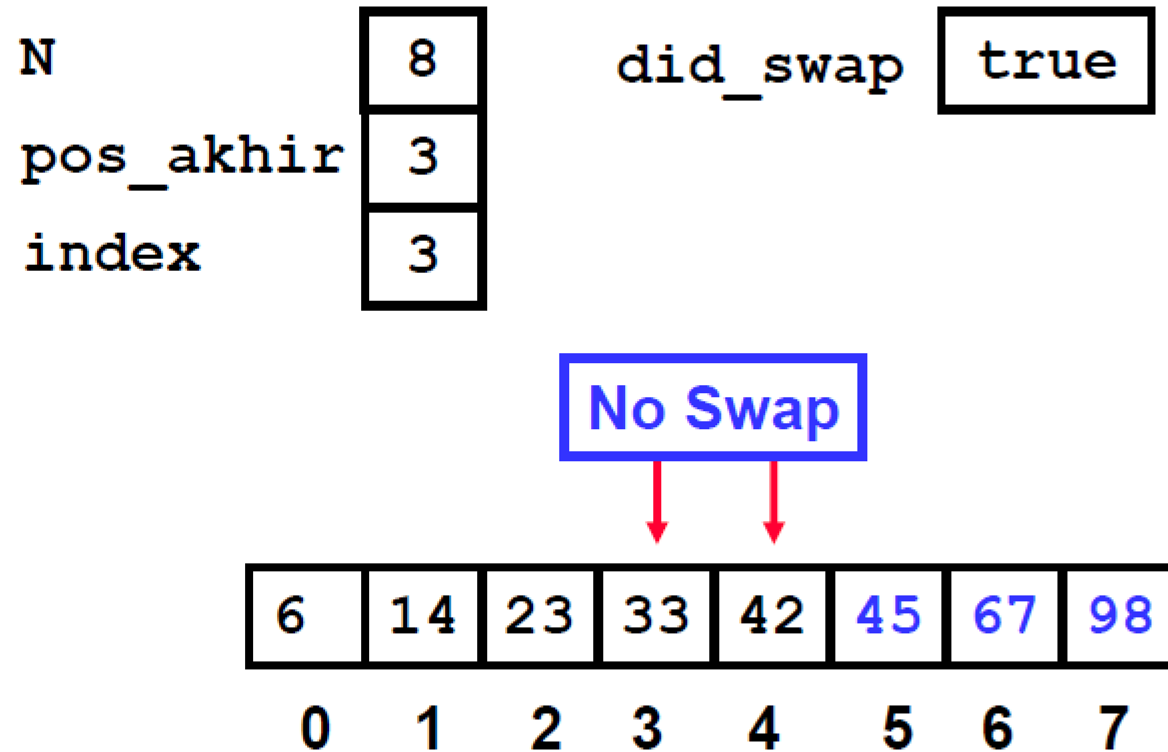
N	8
pos_akhir	3
index	3

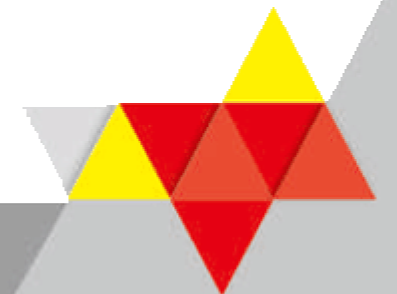
did_swap	true





The Fourth "Bubble Up"





After Fourth Pass of Outer Loop

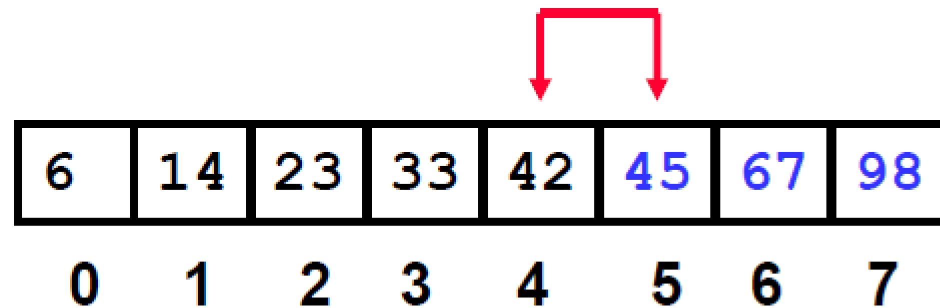
N 8

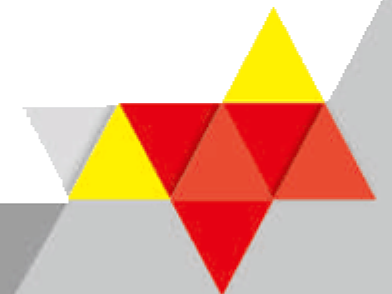
pos_akhir 3

index 4

did_swap true

Finished fourth "Bubble Up"

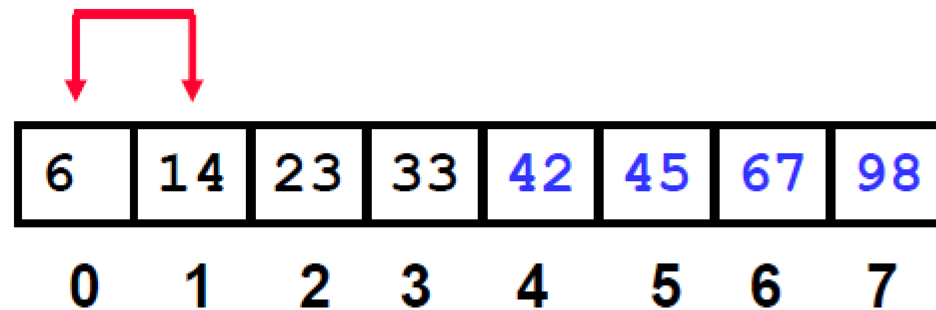


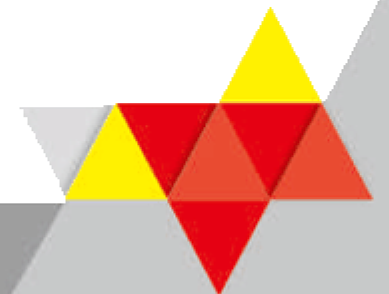


The Fifth “Bubble Up”

N	8
pos_akhir	2
index	0

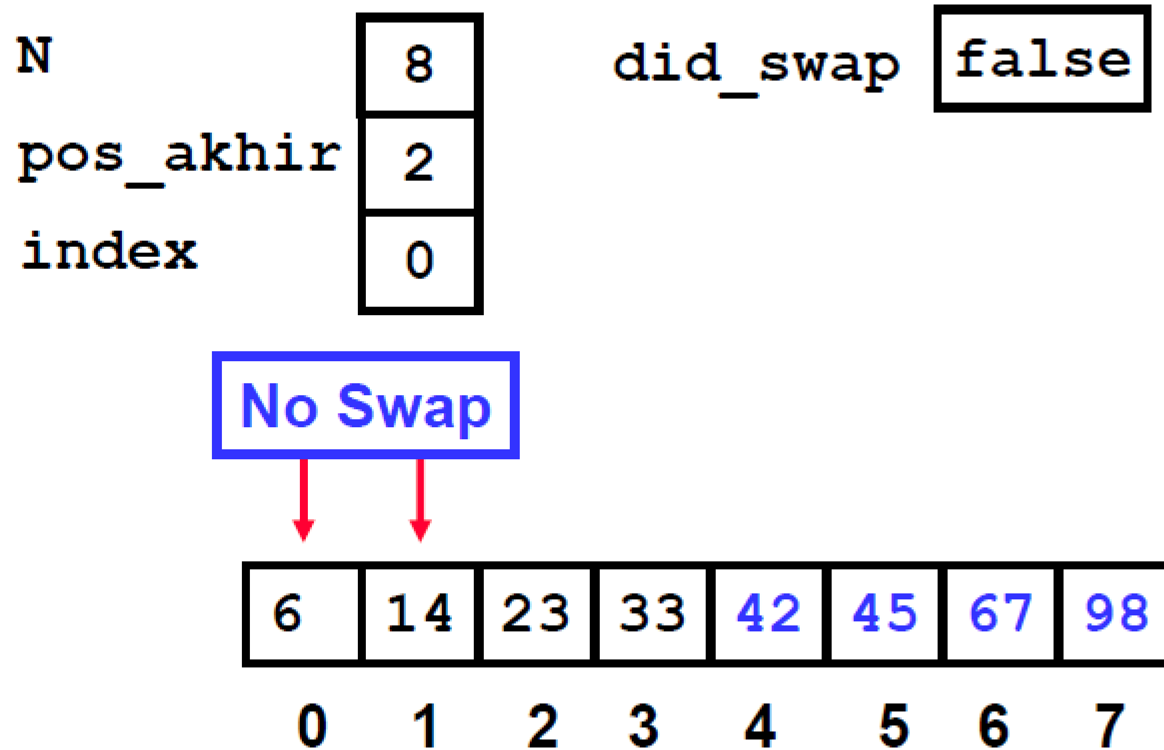
did_swap	false

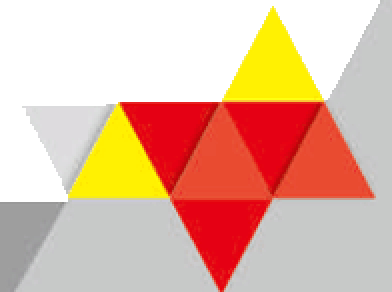




BUBBLE SORT

The Fifth “Bubble Up”

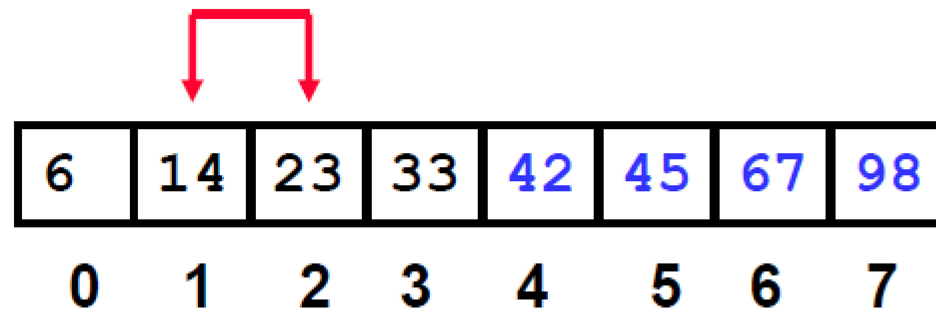


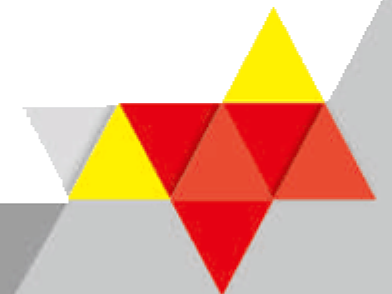


The Fifth “Bubble Up”

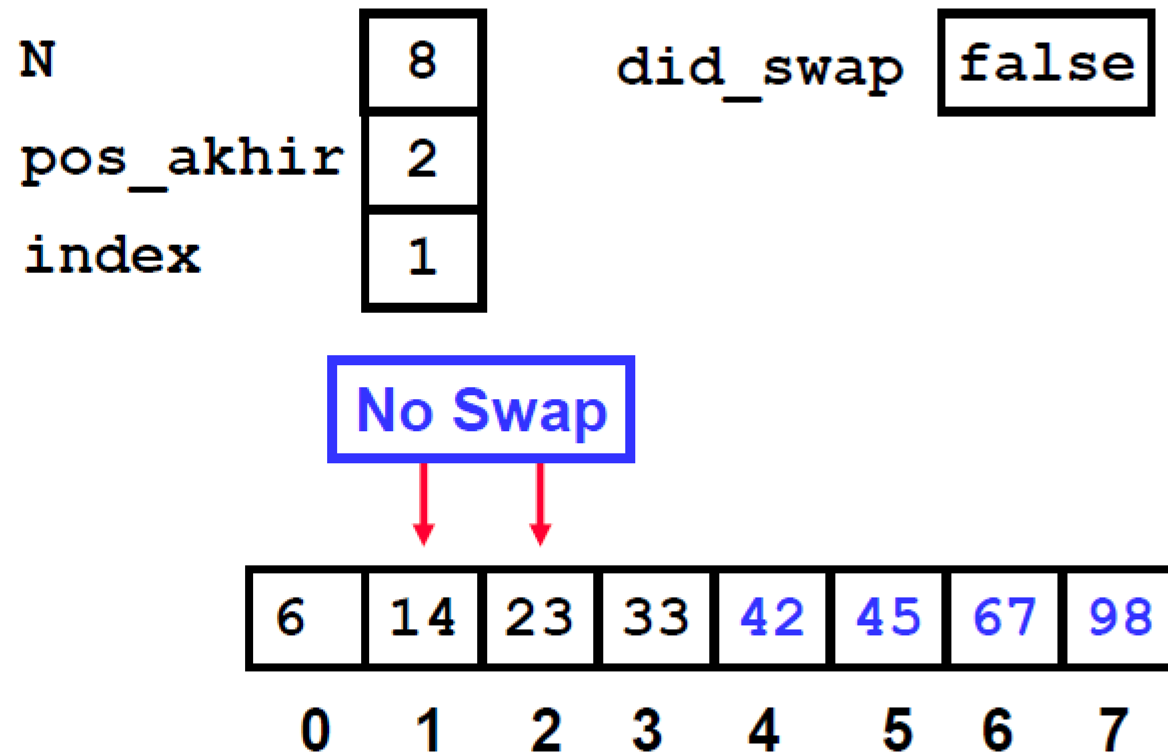
N	8
pos_akhir	2
index	1

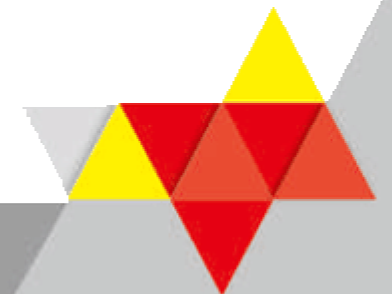
did_swap: false





The Fifth "Bubble Up"

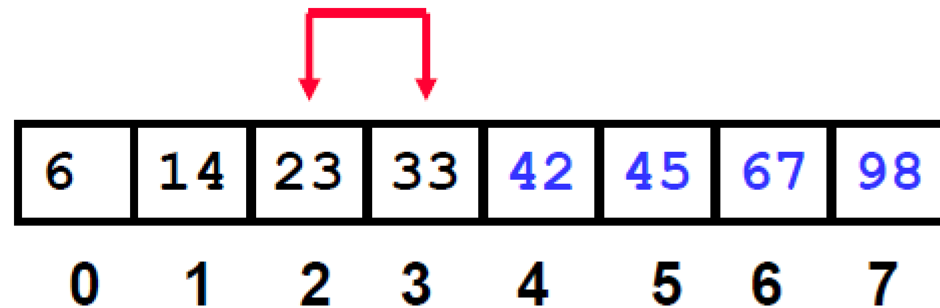


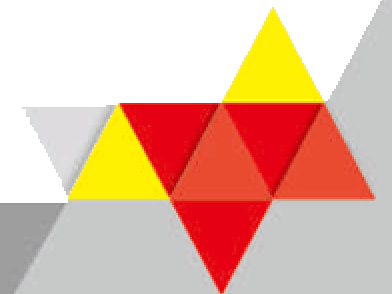


The Fifth “Bubble Up”

N	8
pos_akhir	2
index	2

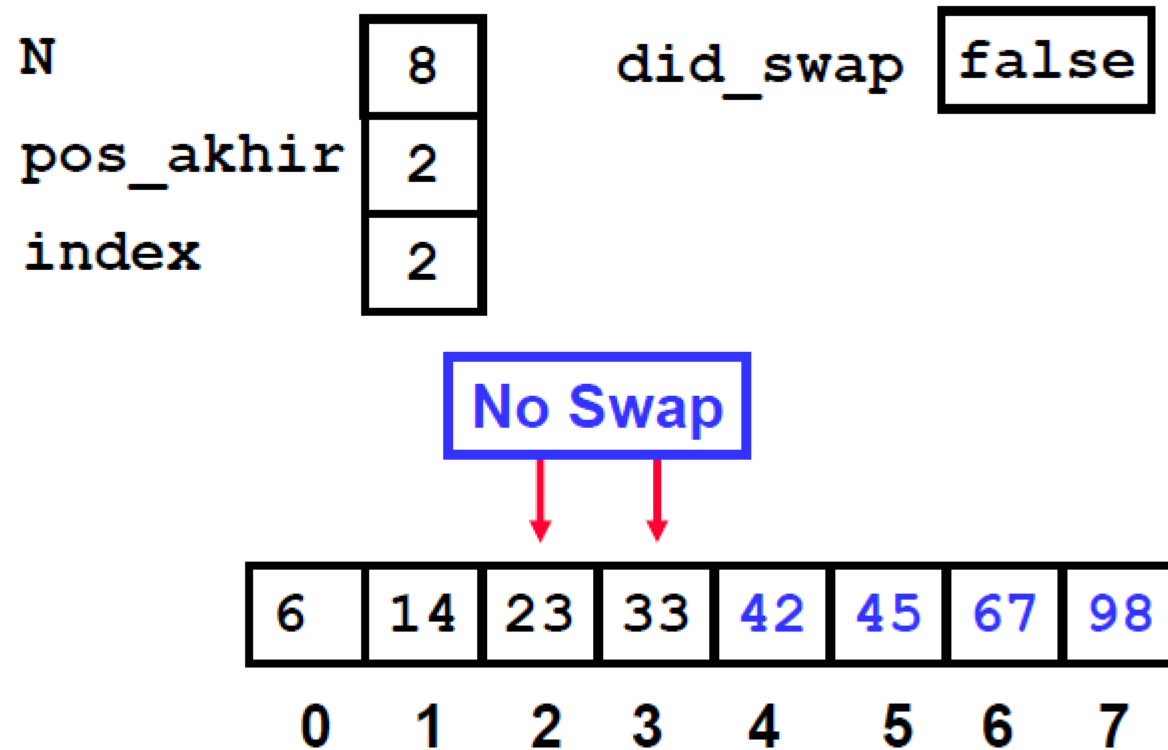
did_swap: false

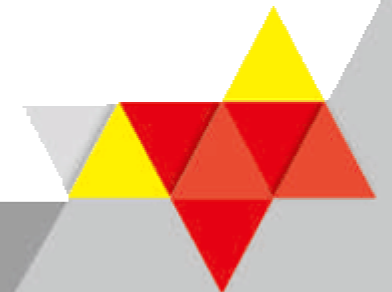




BUBBLE SORT

The Fifth "Bubble Up"



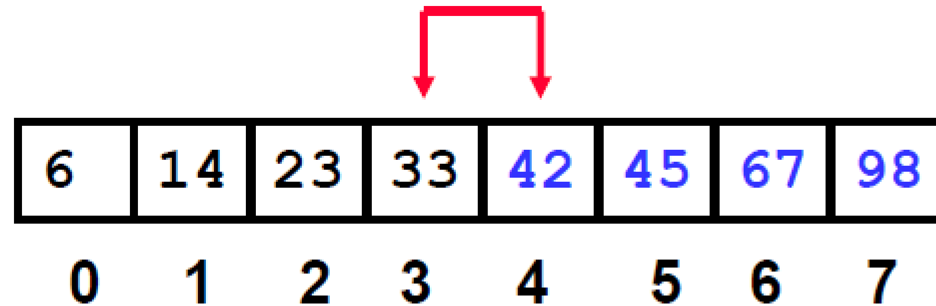


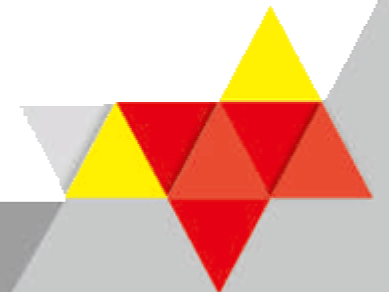
After Fifth Pass of Outer Loop

N	8
pos_akhir	2
index	3

did_swap: false

Finished fifth "Bubble Up"





Finished "Early"

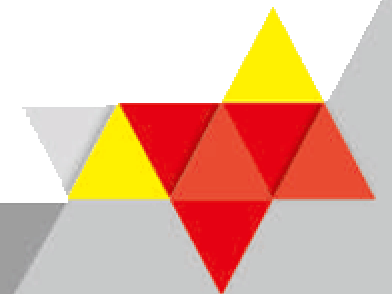
N	8
pos_akhir	2
index	3

did_swap: false

We didn't do any swapping, so all of the other elements must be correctly placed.

We can "skip" the last two passes of the outer loop.

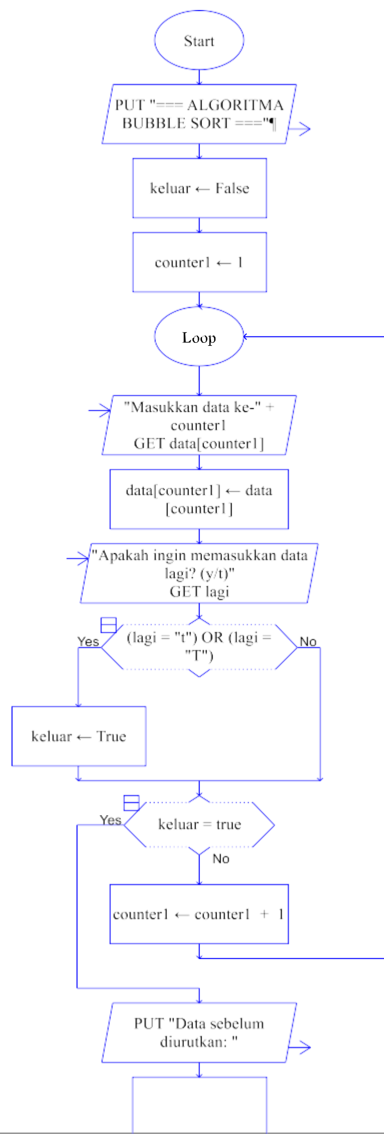
6	14	23	33	42	45	67	98
0	1	2	3	4	5	6	7

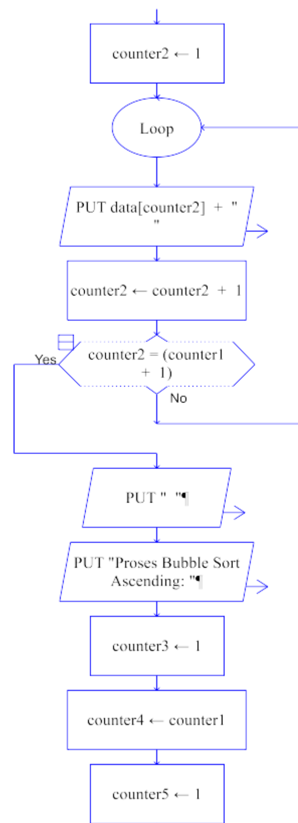


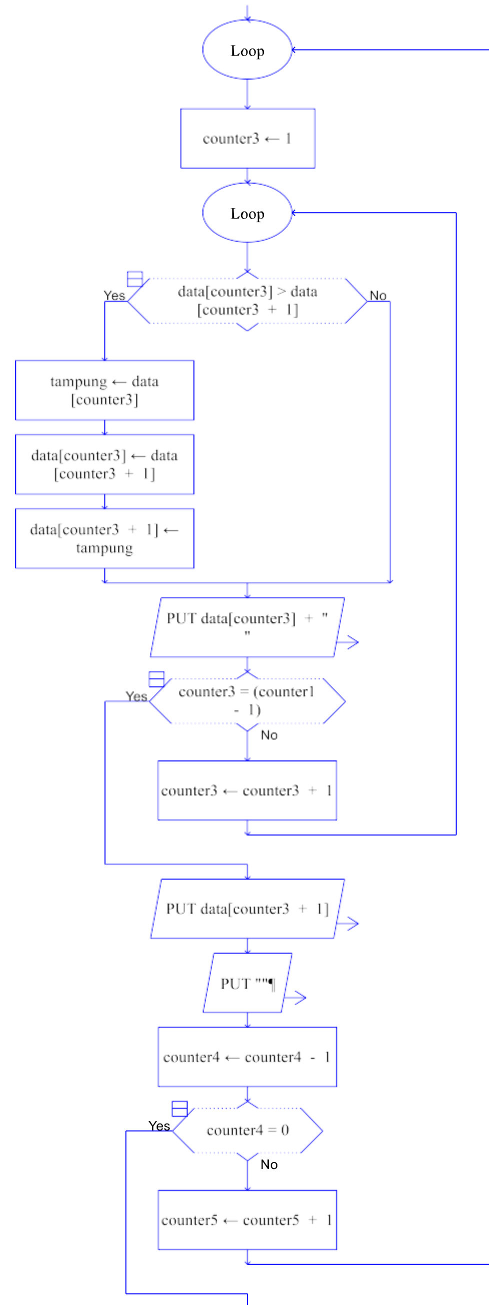
Algoritma

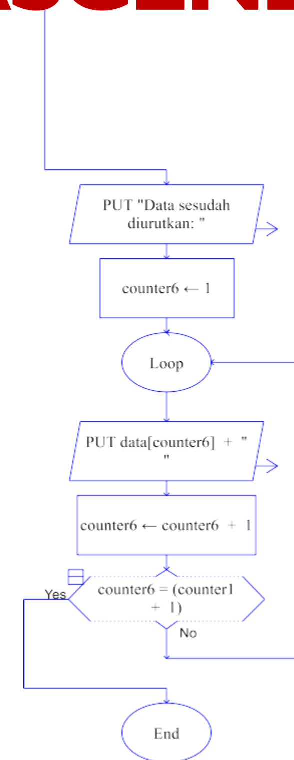
1. i ← N
2. selama (i >0) kerjakan baris 3 sampai dengan 11
3. did_swap = false
4. j ← 1
5. Selama (j < i) kerjakan baris 5 sampai dengan 8
6. Jika (Data[j-1] > Data[j]) maka kerjakan baris 7 & 8
7. tukar Data[j-1] dengan Data[j]
8. did_swap = true
9. j ← j + 1
10. Jika did_swap = false keluar dari loop
11. i ← i - 1

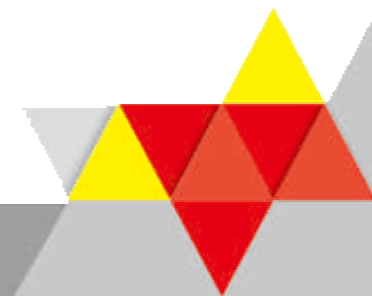
BUBBLE SORT ASCENDING



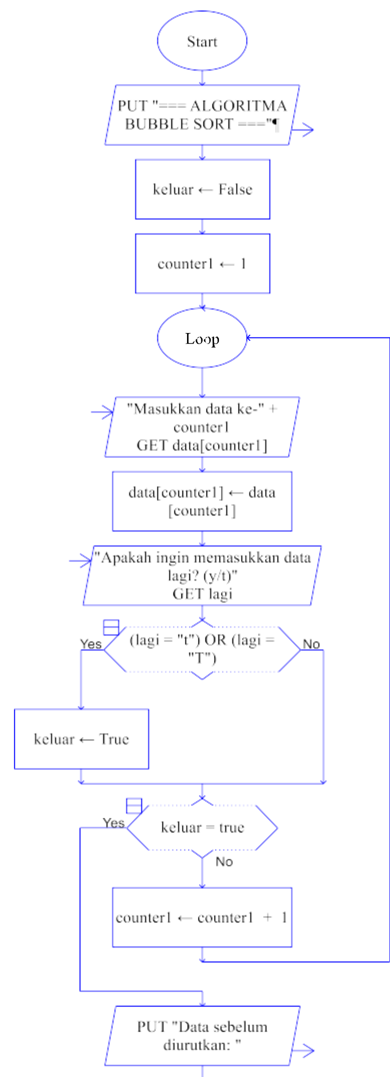


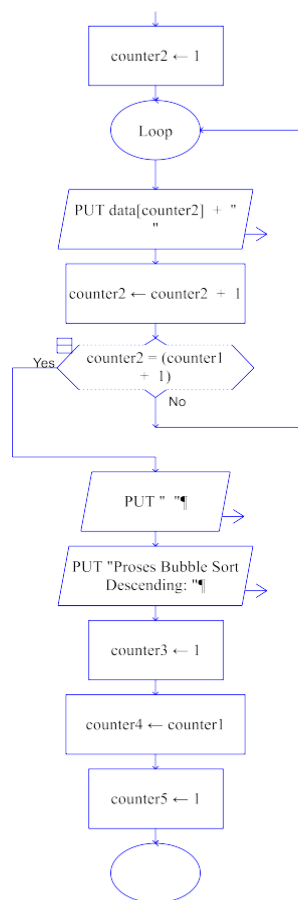


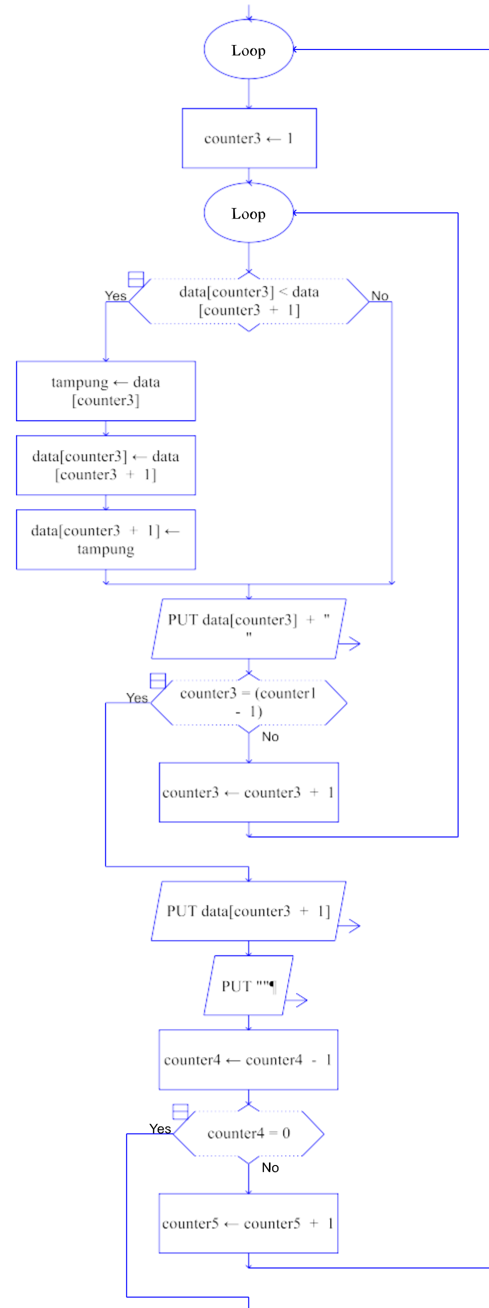


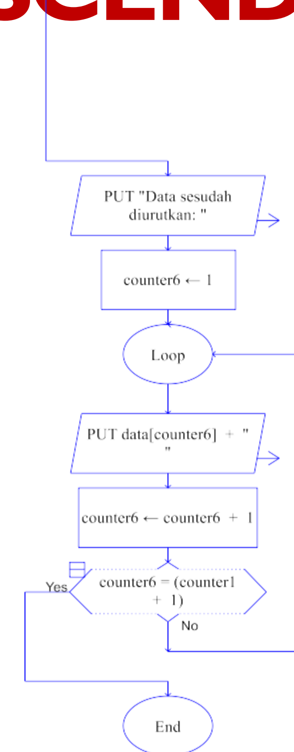


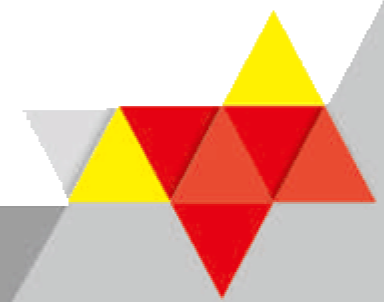
BUBBLE SORT DESCENDING



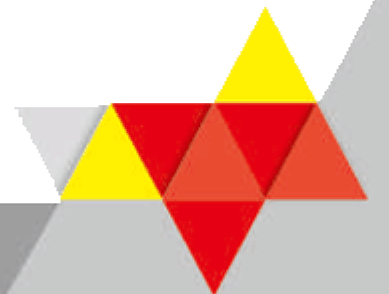




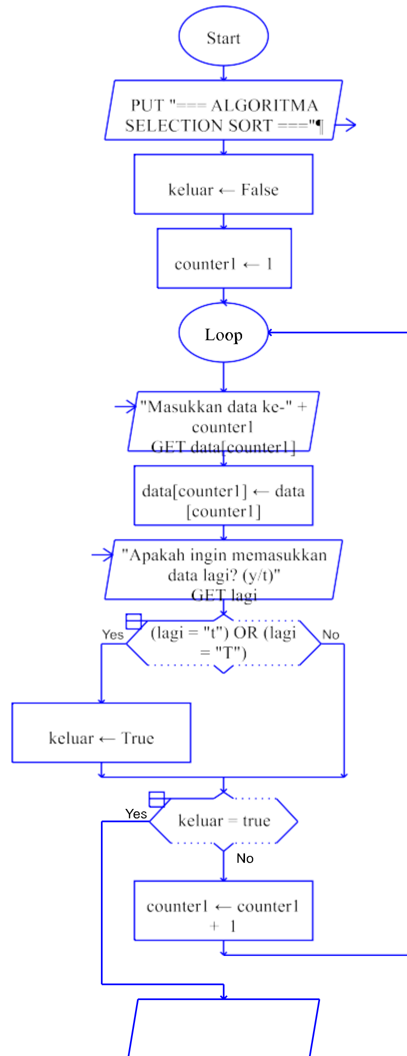


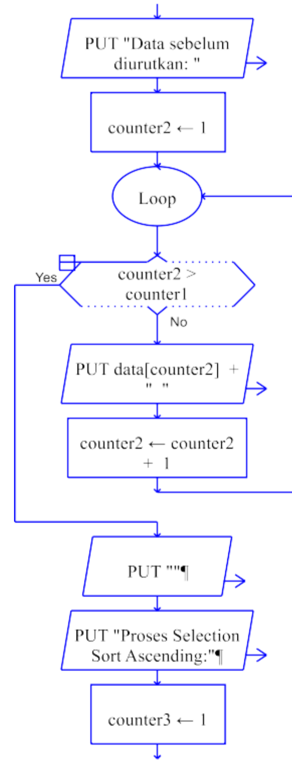


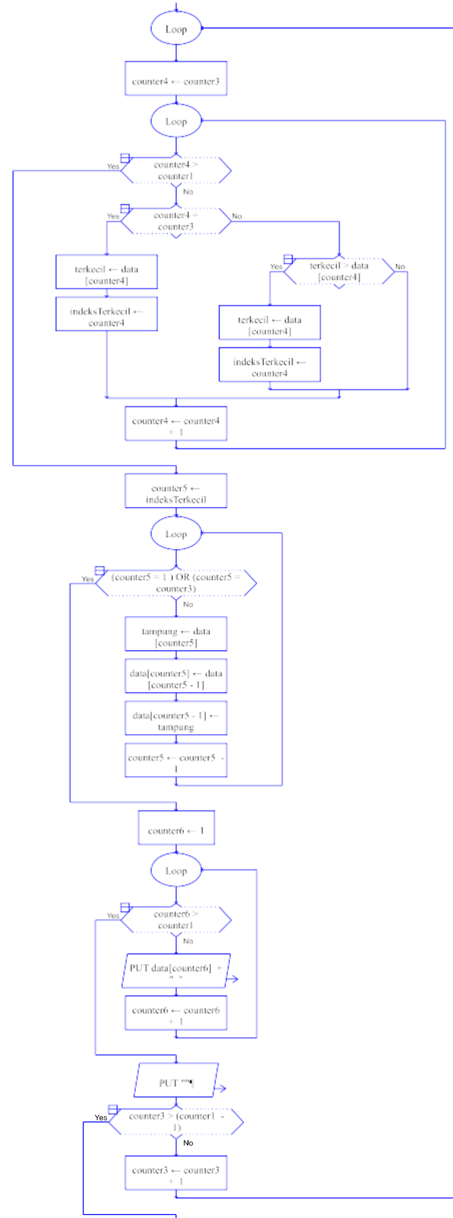
2. SELECTION SORT

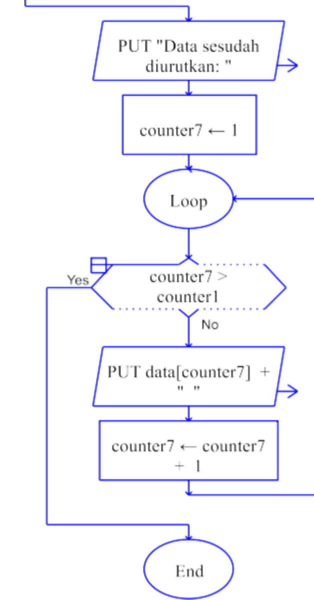


SELECTION SORT ASCENDING



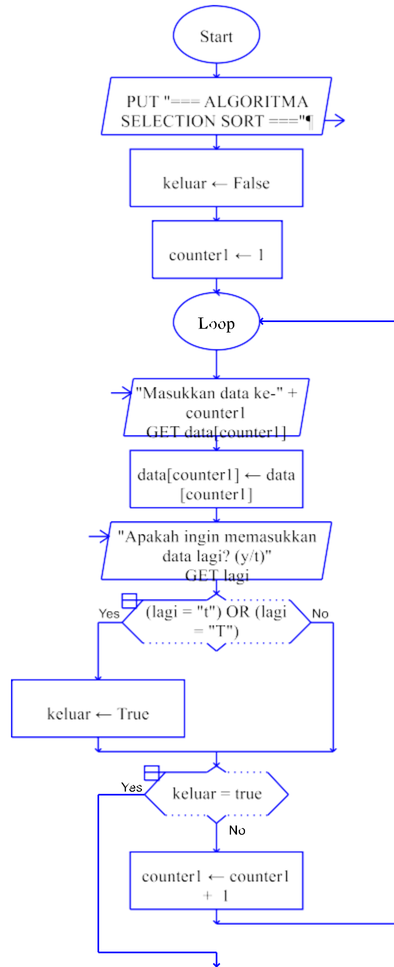


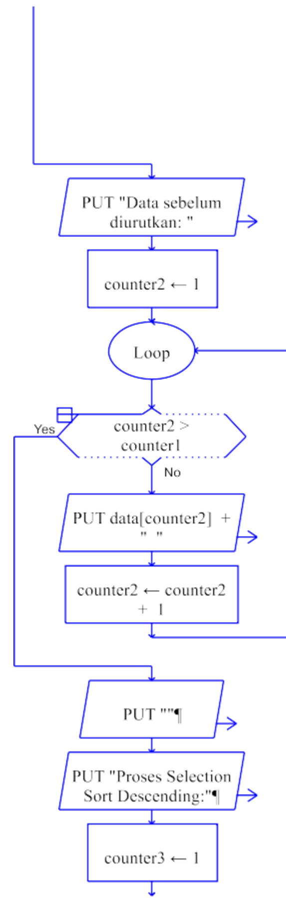


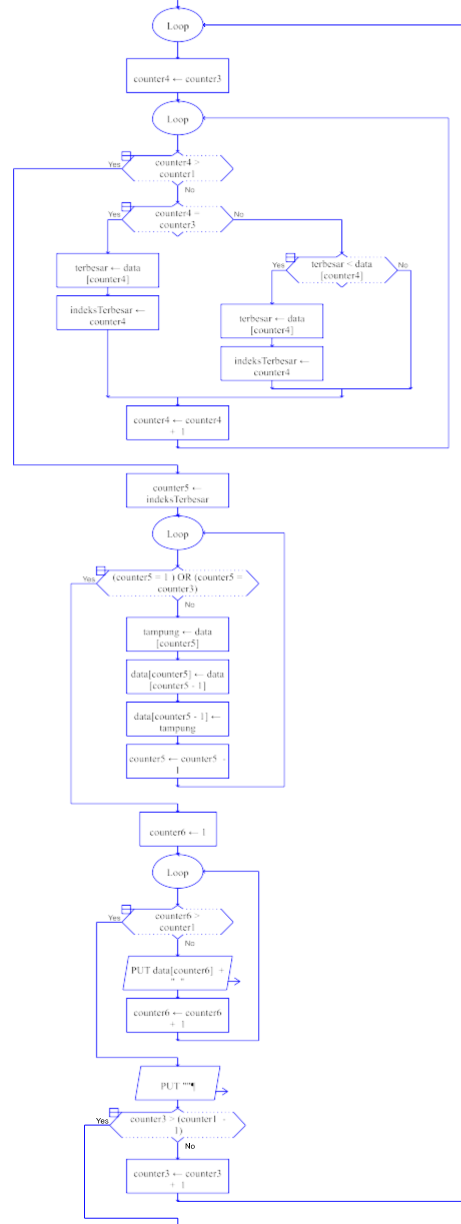


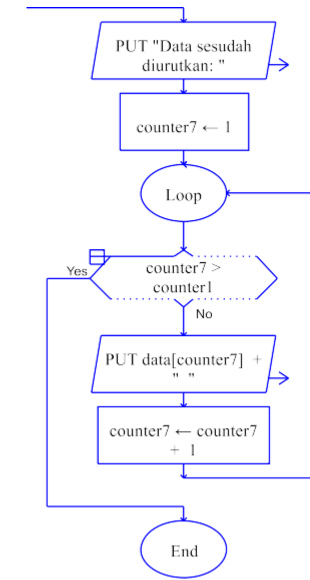
UNIVERSITAS Dinamika

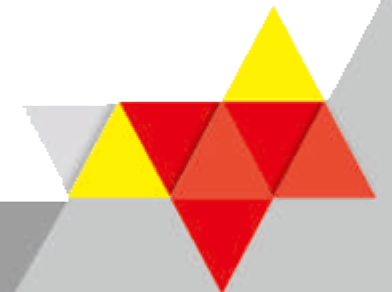
SELECTION SORT DESCENDING



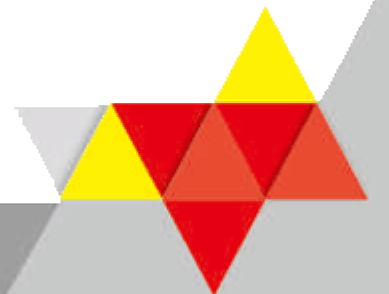








3. MERGE SORT



Divide and Conquer

- **Metode Divide and Conquer, setiap kali memecah persoalan menjadi setengahnya, namun menggunakan hasil dari kedua bagian tersebut:**
  - **memotong permasalahan menjadi dua bagian hingga permasalahan trivial → tidak ber-problem lagi**
  - **menyelesaikan untuk dua bagian**
  - **mengkombinasikan penyelesaian**

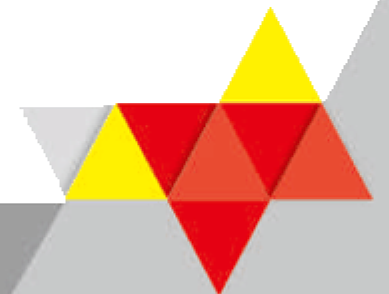
MERGE SORT

- *A divide-and-conquer algorithm:*
  Membagi unsorted array menjadi 2 bagian hingga menghasilkan sub-arrays yang hanya berisi satu elemen
- *Merge together* solusi dari sub-problem

  HOW?

  - Bandingkan elemen pertama dari 2 sub-array
  - Ambil elemen yang terkecil dan letakkan pada array hasil
  - Teruskan proses pembandingan dan pengambilan, sampai seluruh elemen sub-array dipindahkan ke array hasil

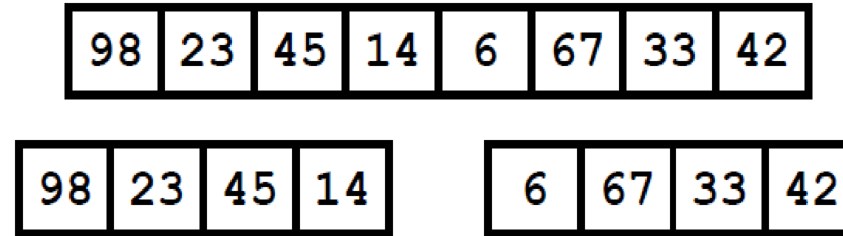
37	23	6	89	15	12	2	19

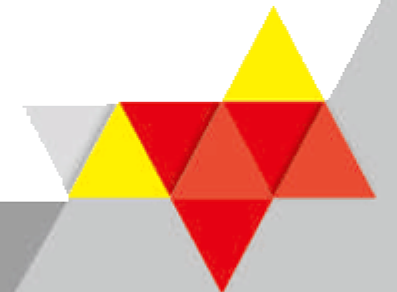


MERGE SORT

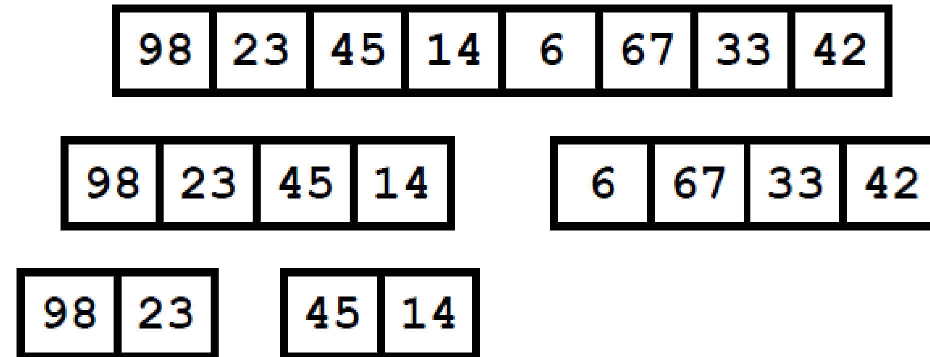
98	23	45	14	6	67	33	42

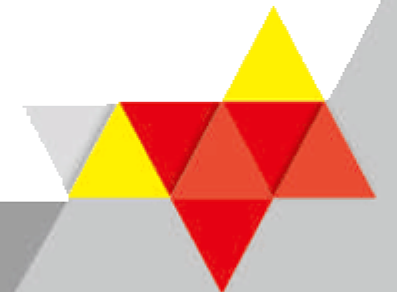
MERGE SORT





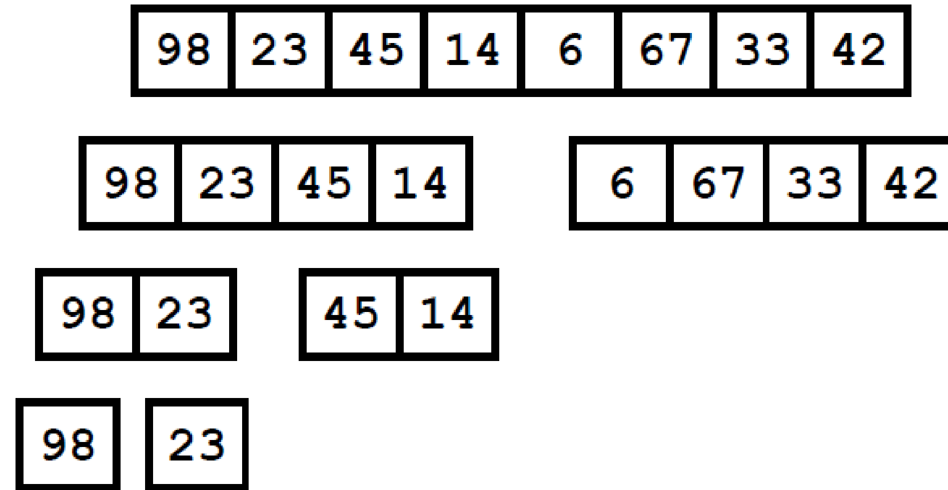
MERGE SORT

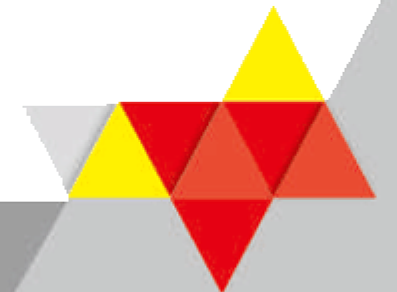




UNIVERSITAS Dinamika

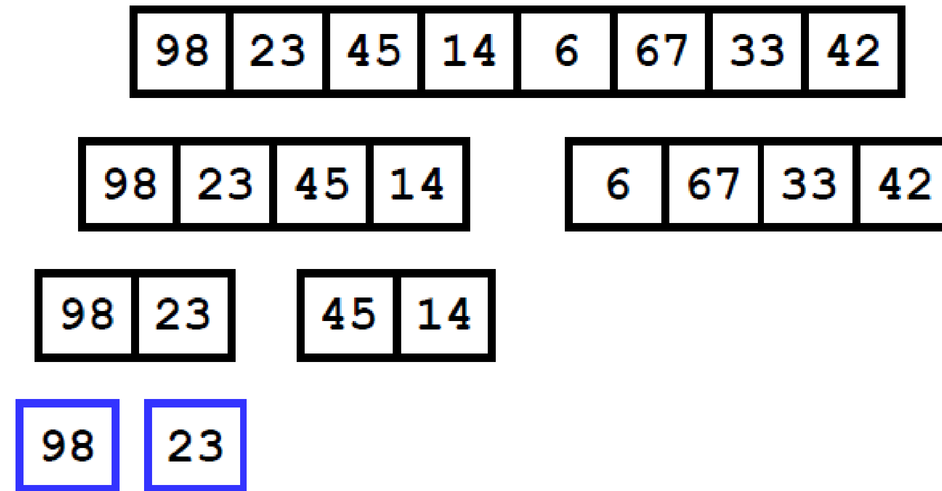
MERGE SORT



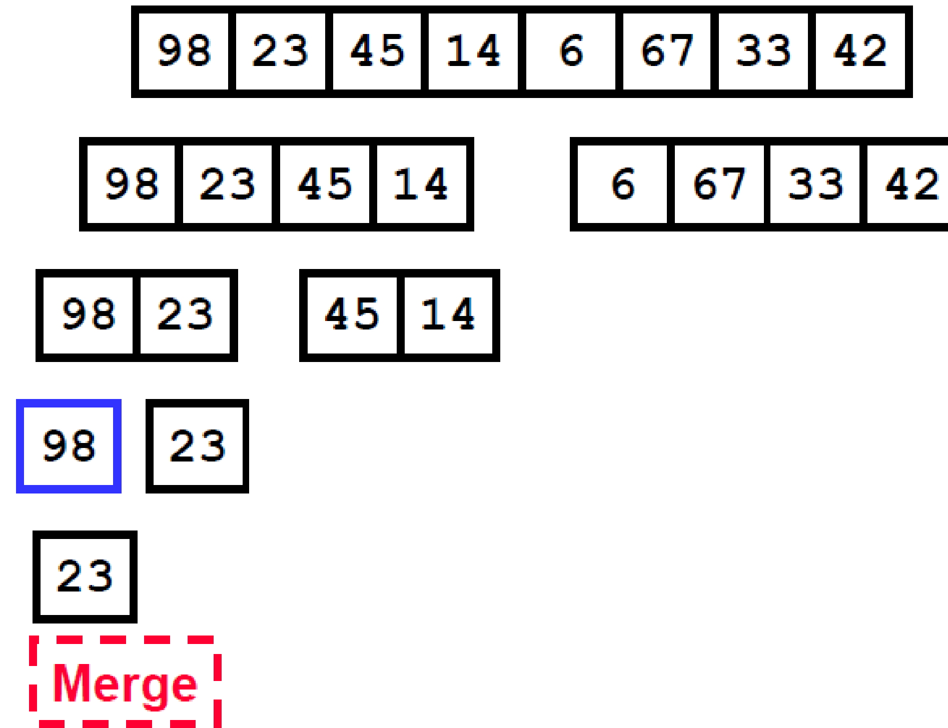


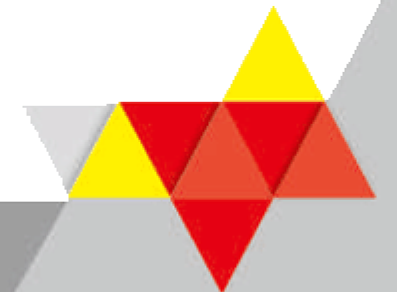
Universitas Dinamika

MERGE SORT



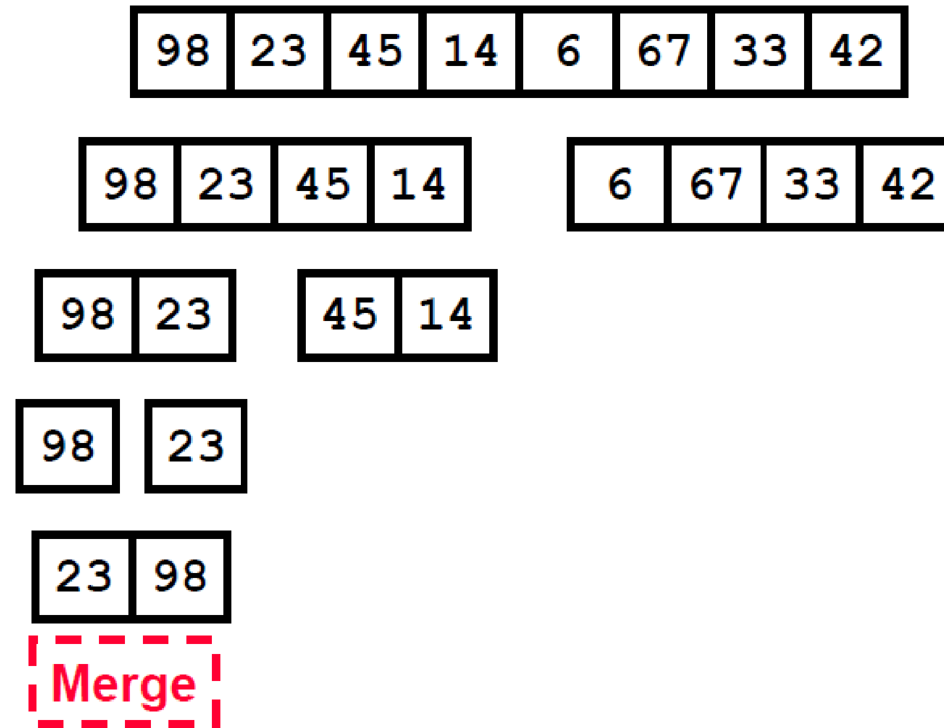
MERGE SORT

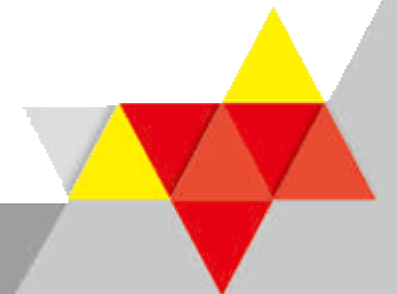




Universitas Dinamika

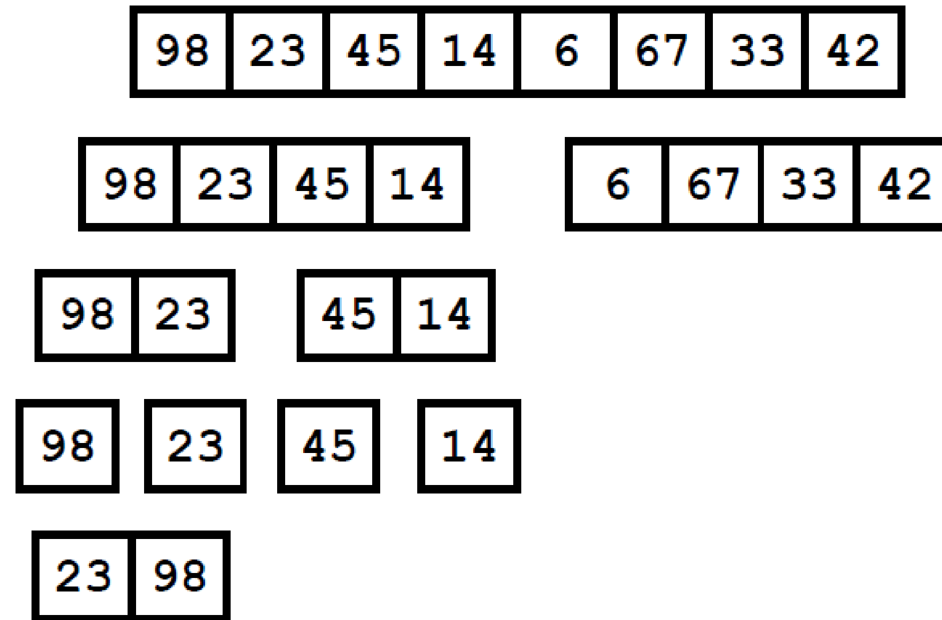
MERGE SORT

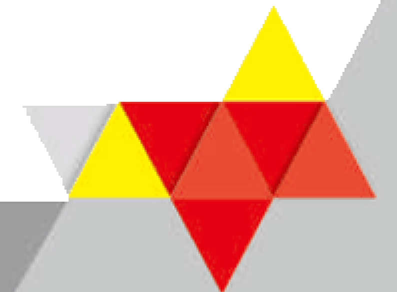




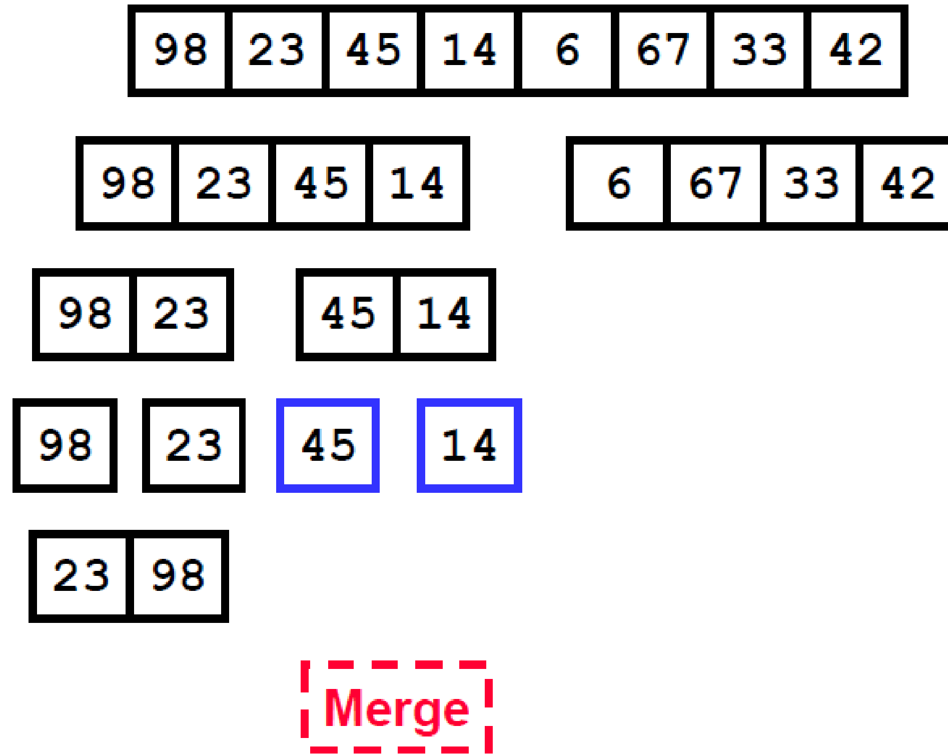
Universitas Dinamika

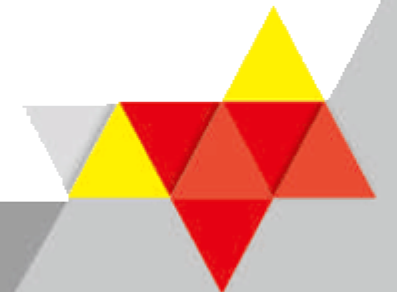
MERGE SORT



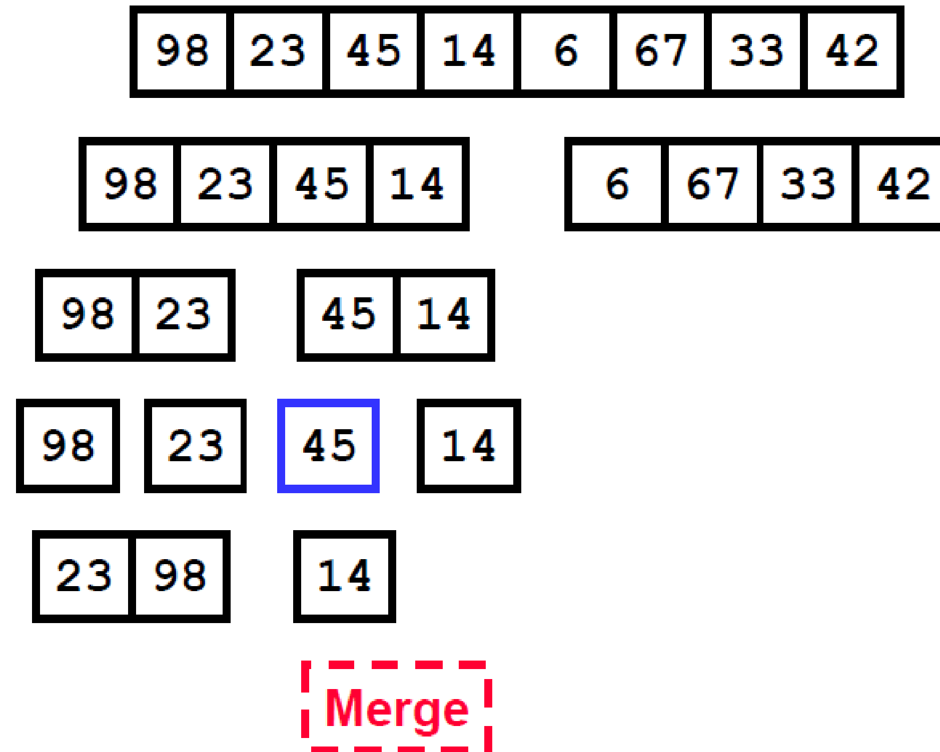


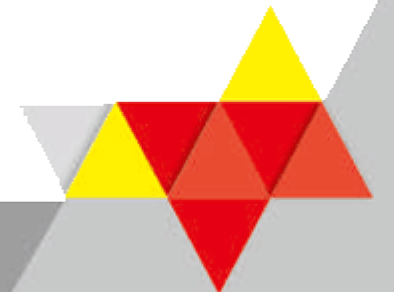
MERGE SORT



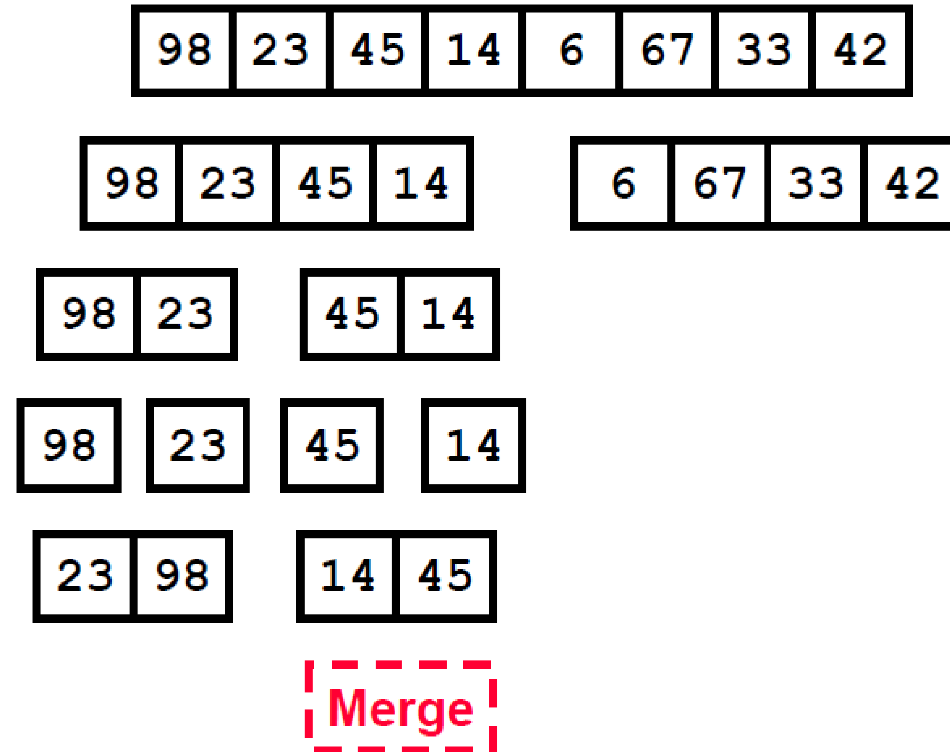


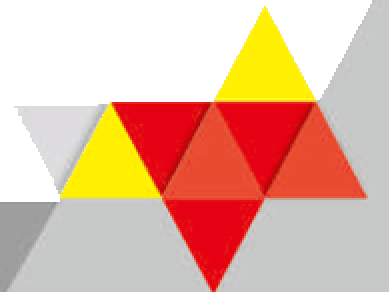
MERGE SORT





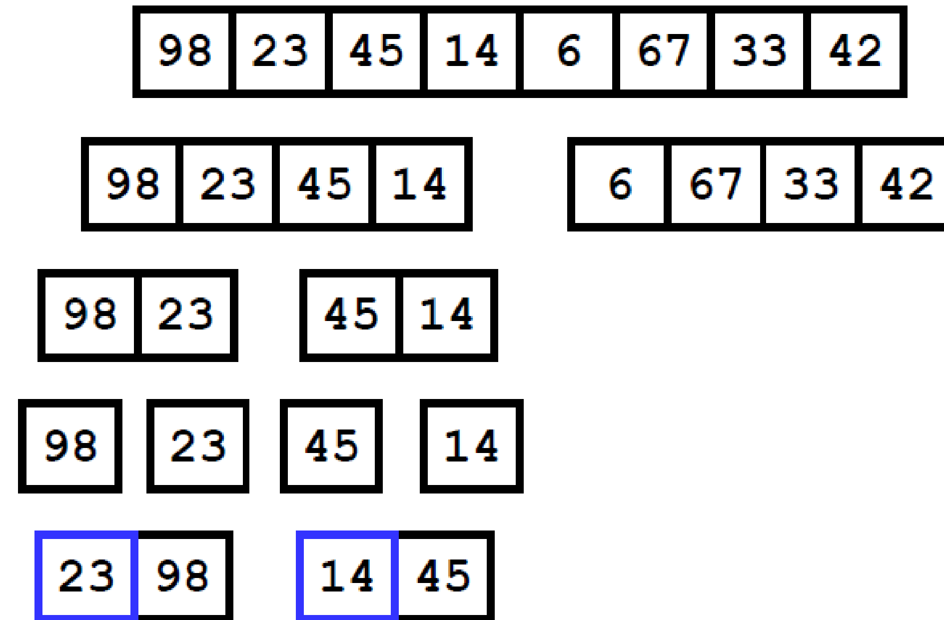
MERGE SORT

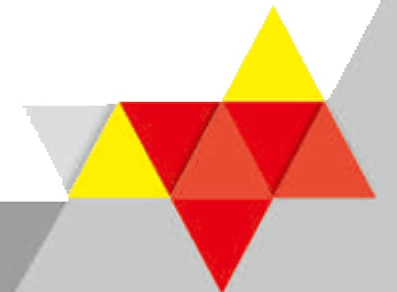




UNIVERSITAS Dinamika

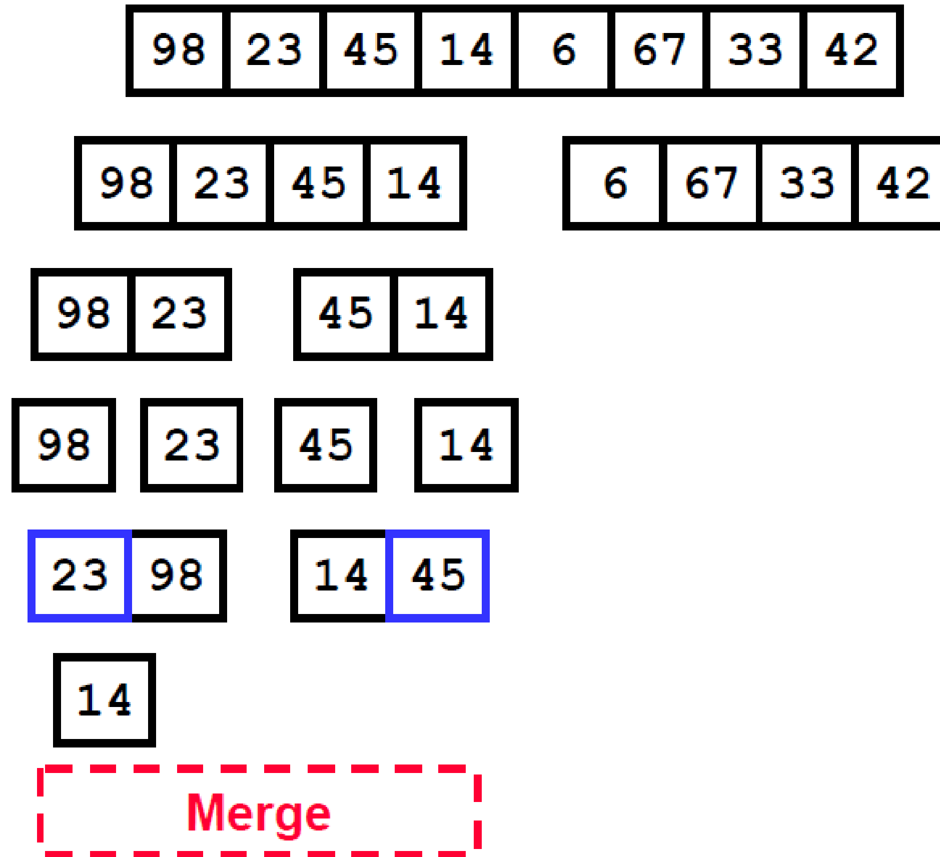
MERGE SORT

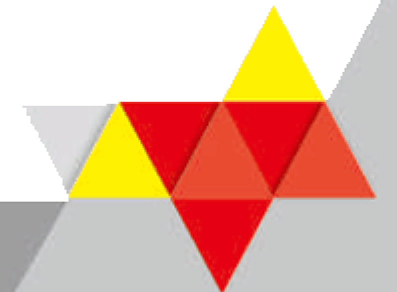




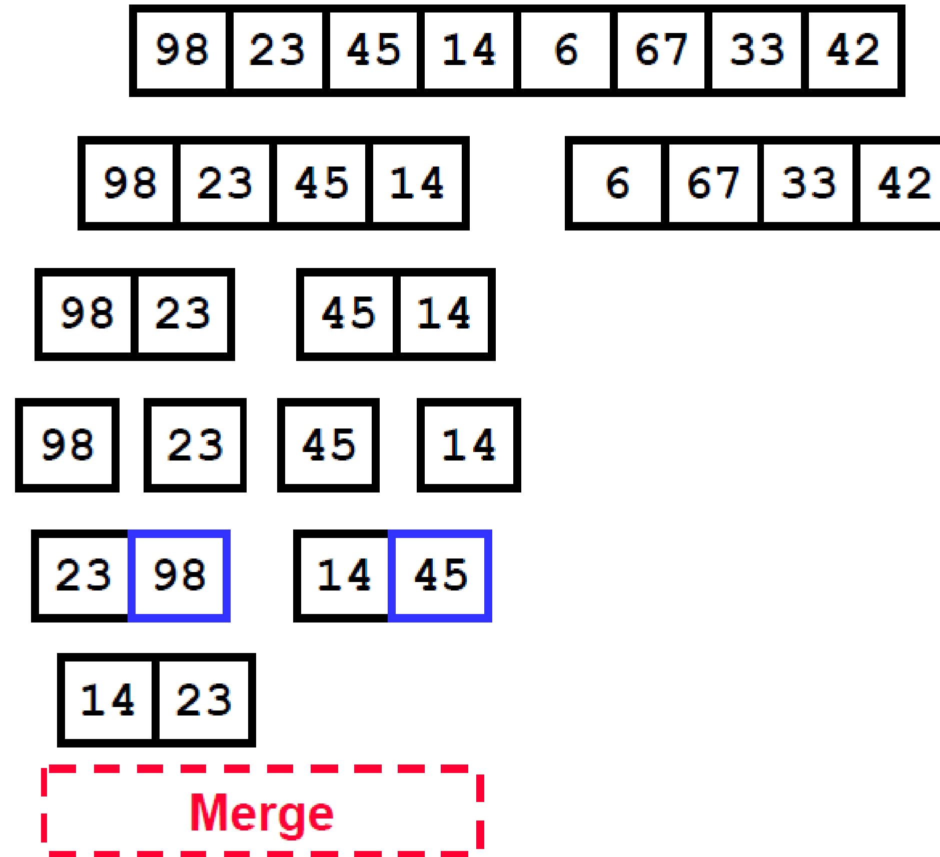
UNIVERSITAS Dinamika

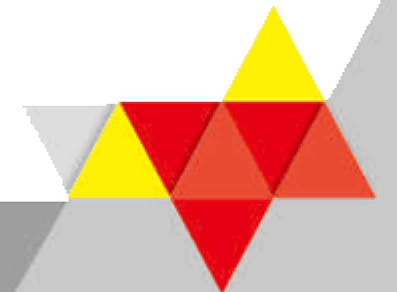
MERGE SORT





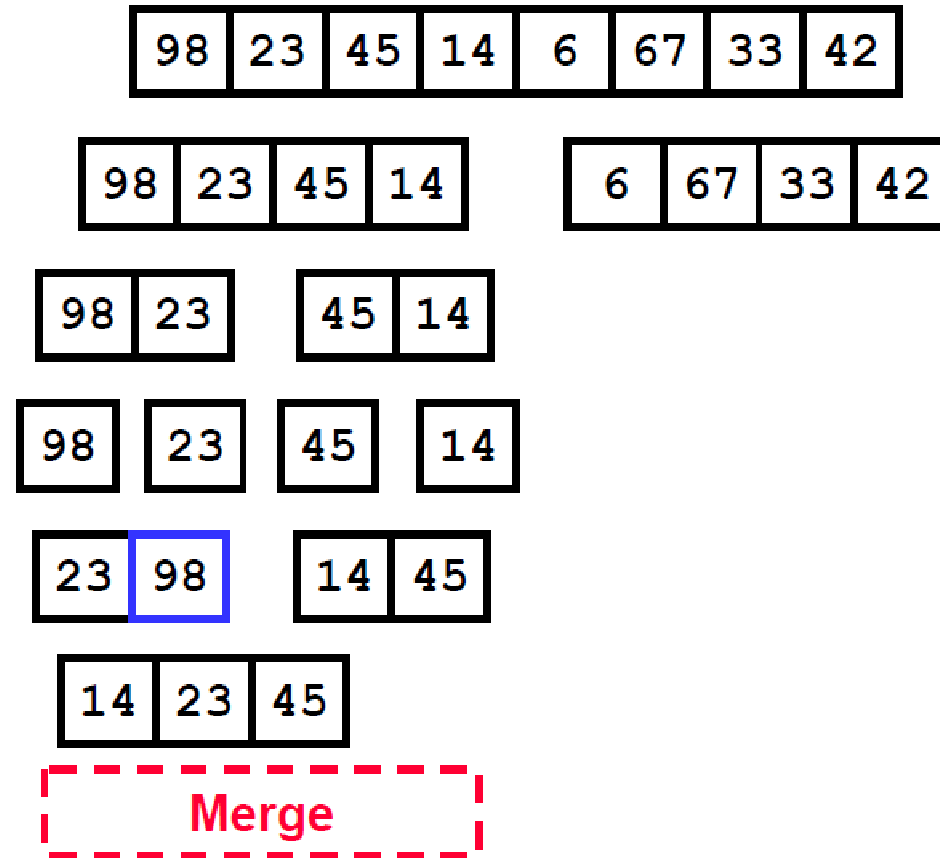
MERGE SORT

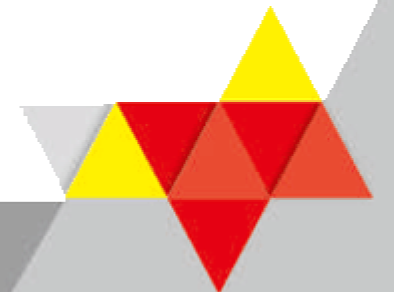




Universitas Dinamika

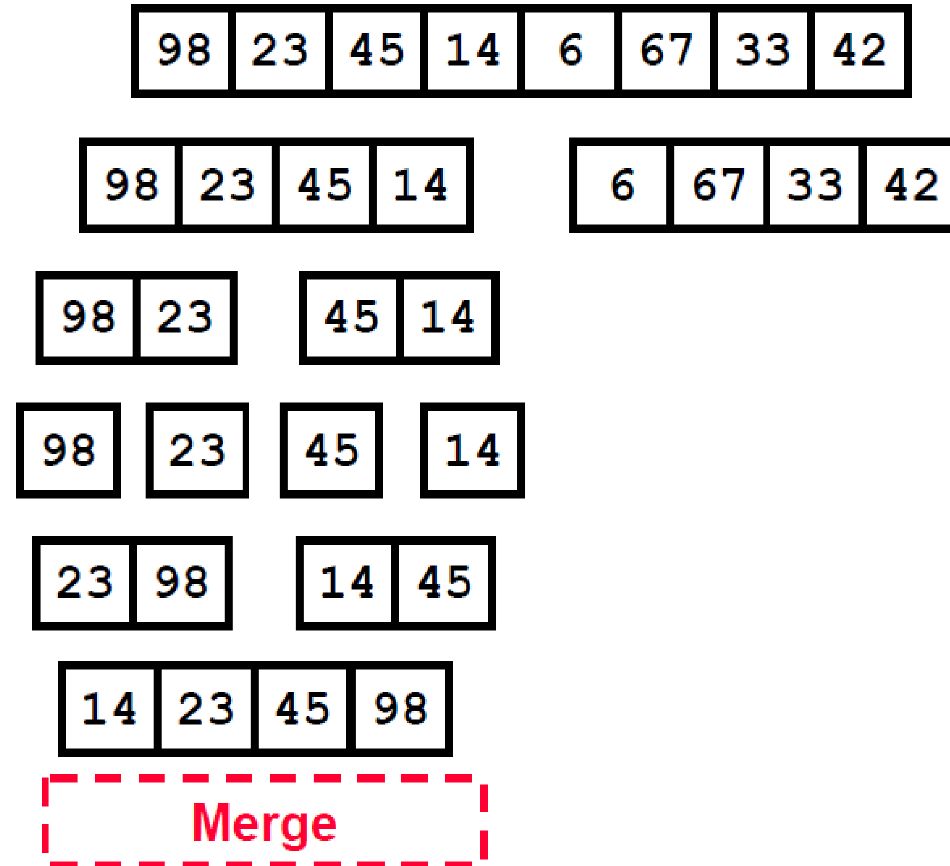
MERGE SORT

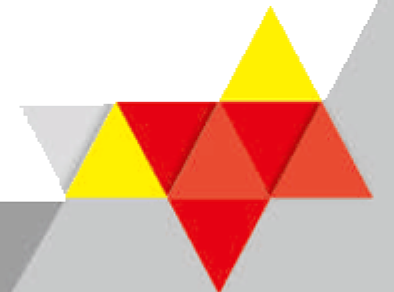




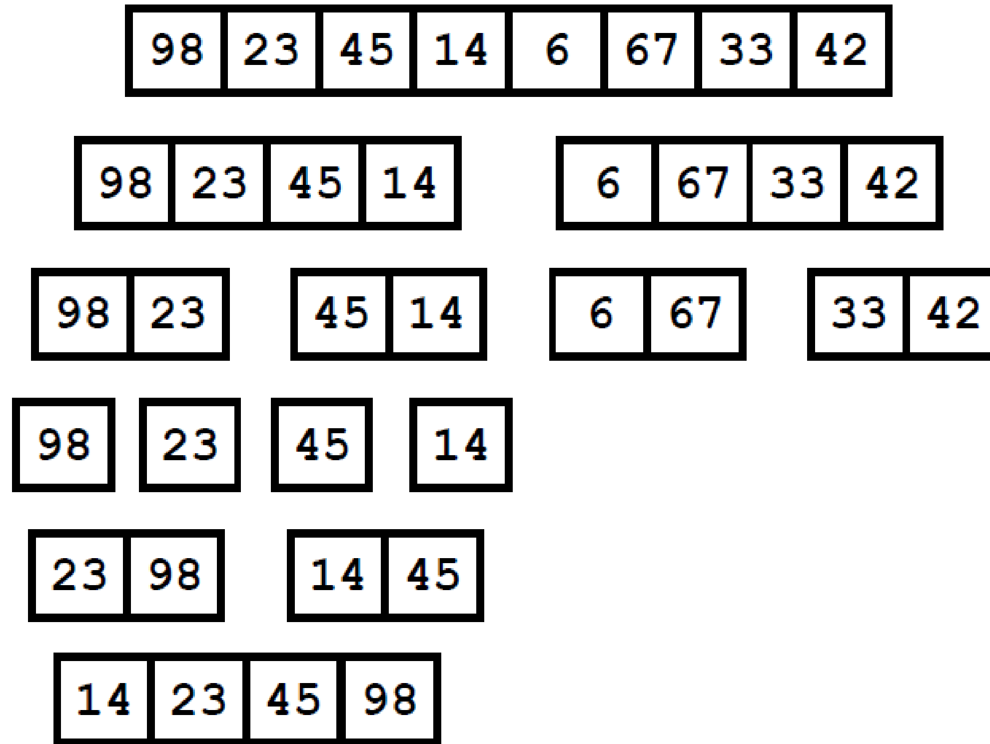
Universitas Dinamika

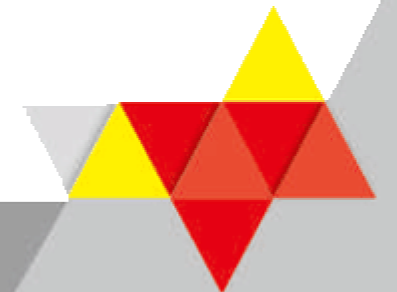
MERGE SORT



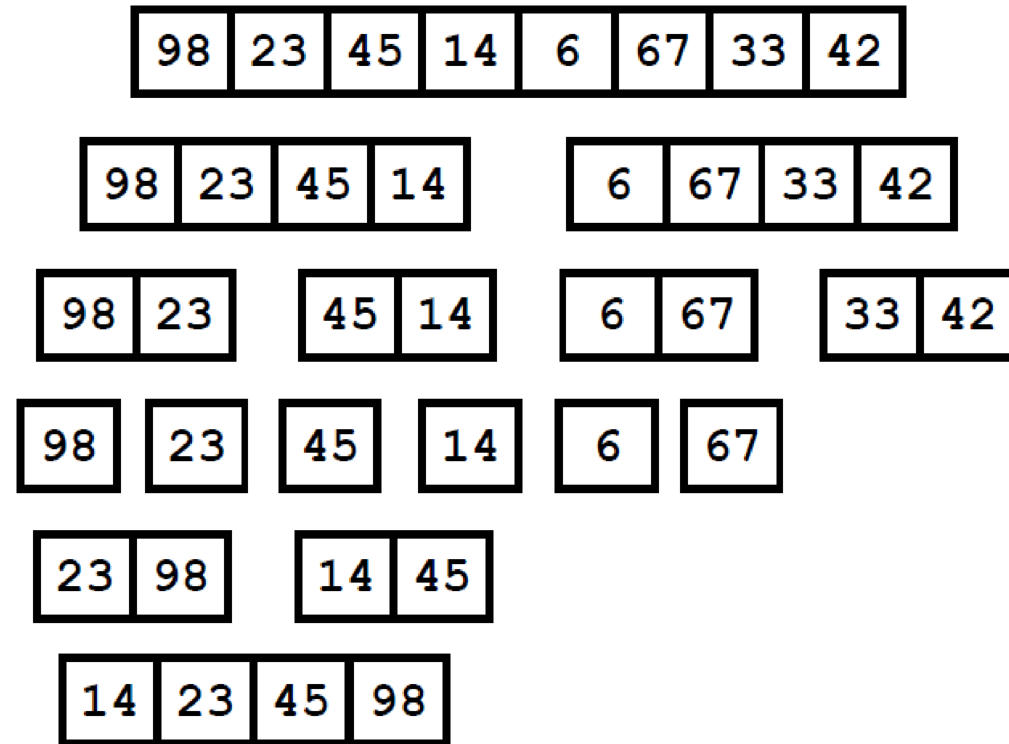


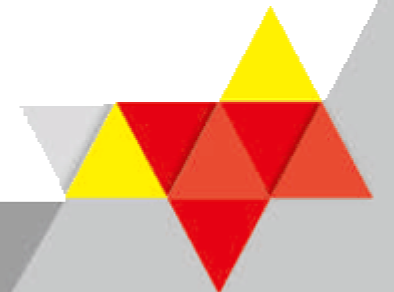
MERGE SORT



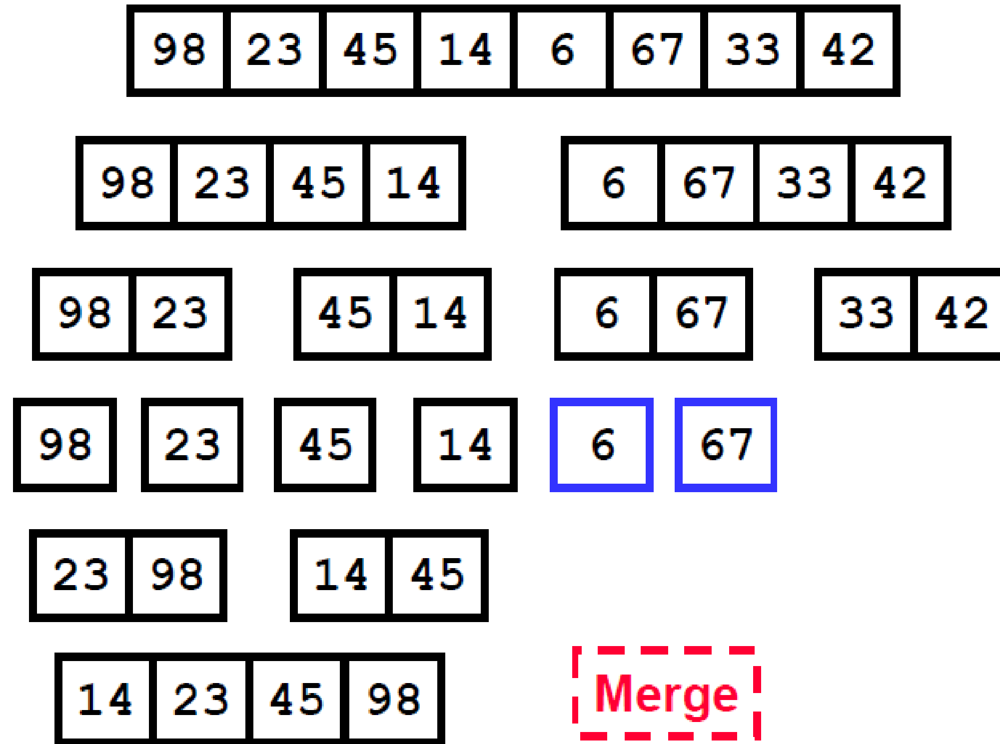


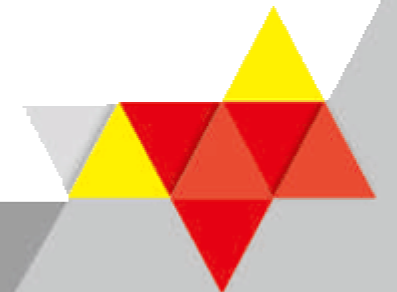
MERGE SORT



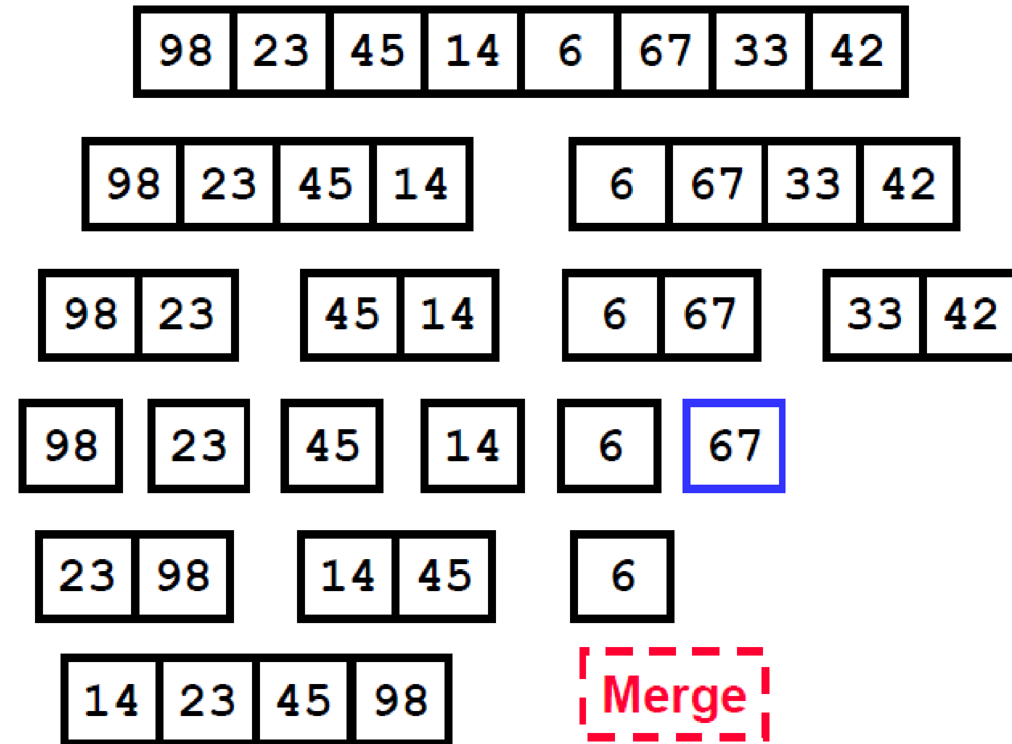


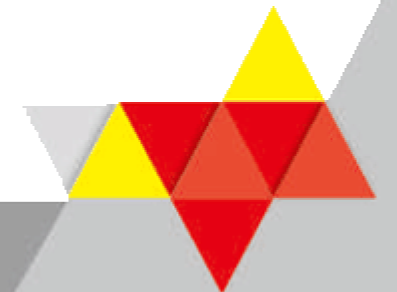
MERGE SORT



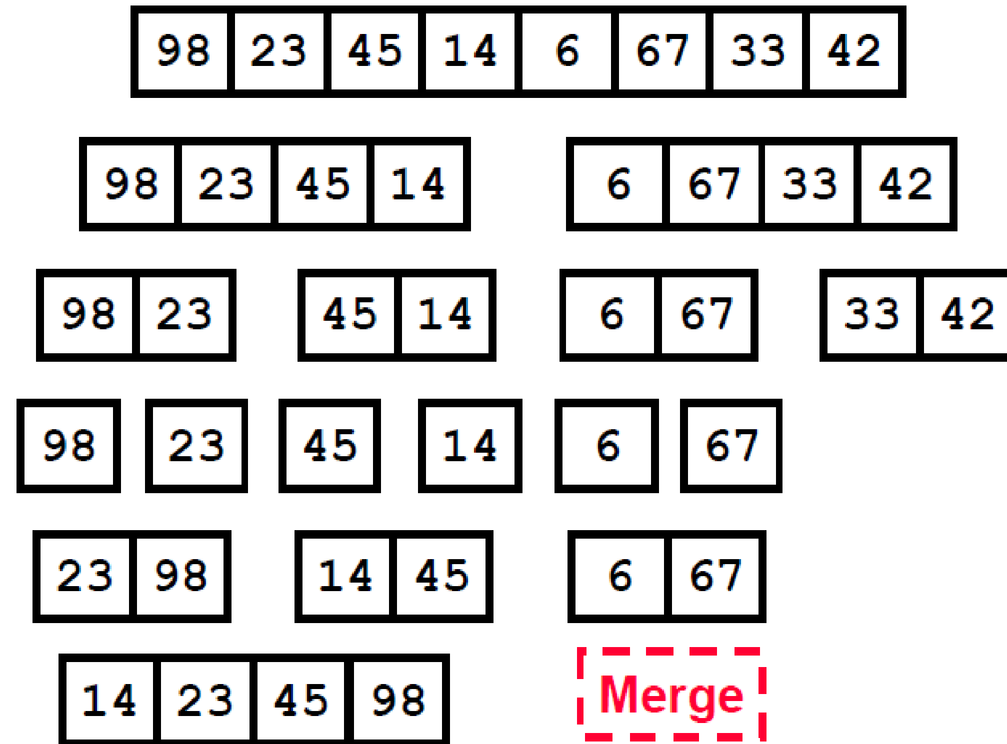


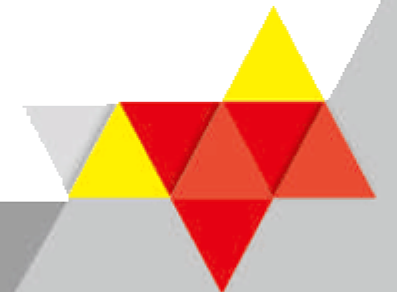
MERGE SORT





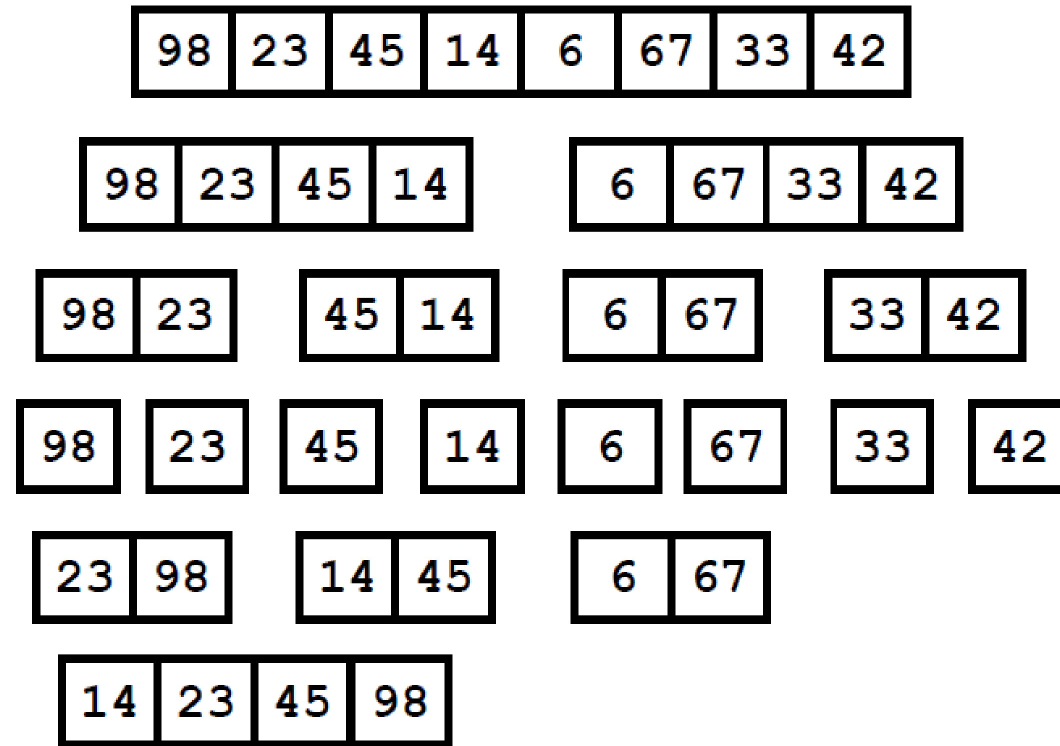
MERGE SORT

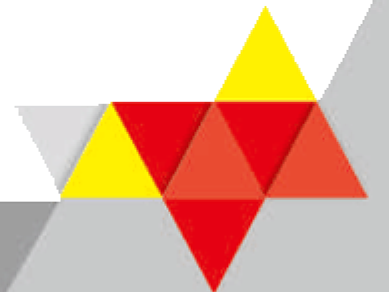




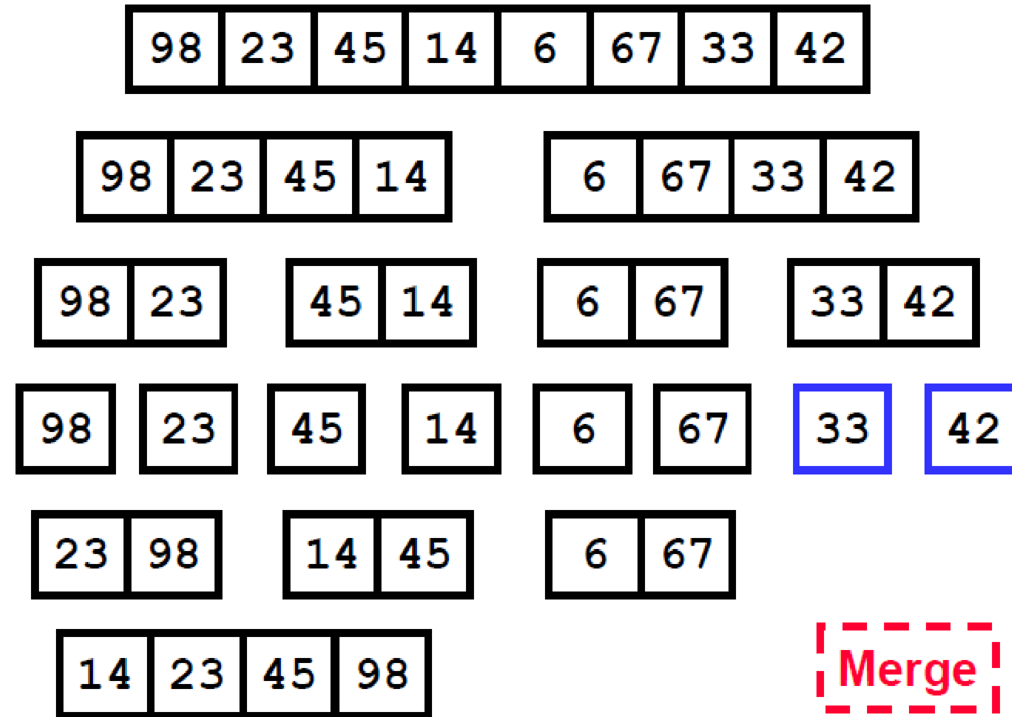
UNIVERSITAS Dinamika

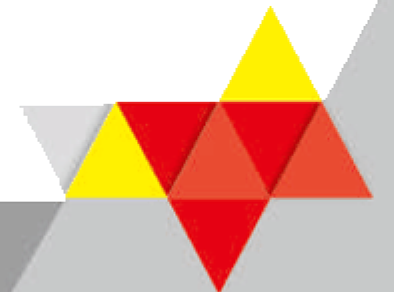
MERGE SORT



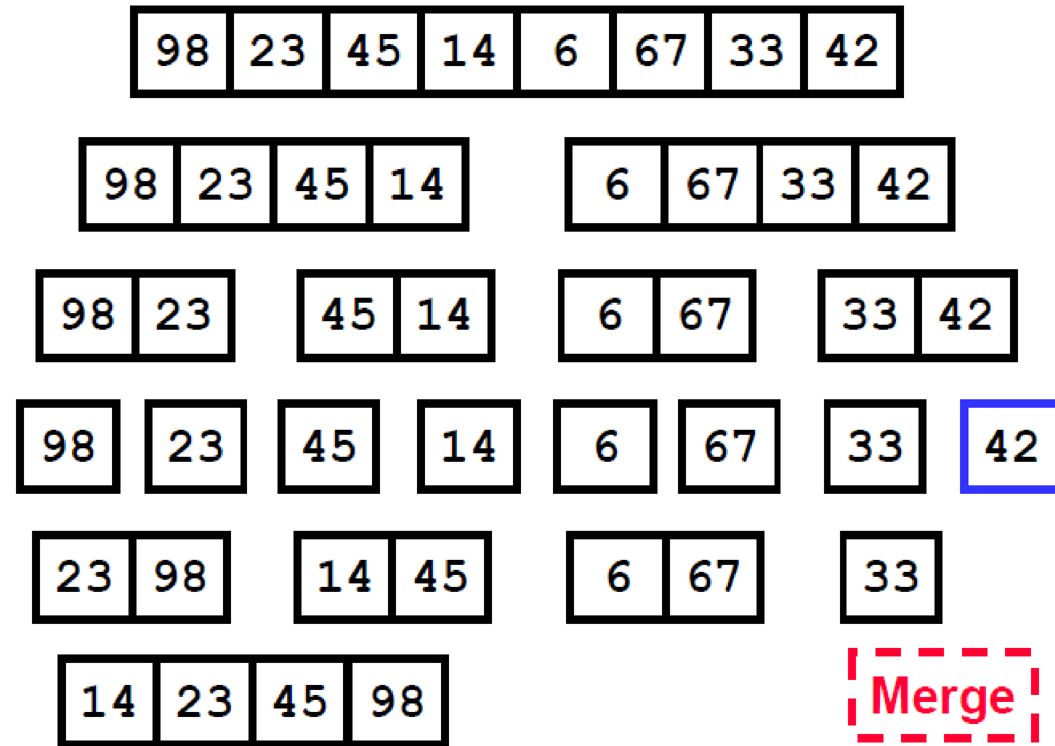


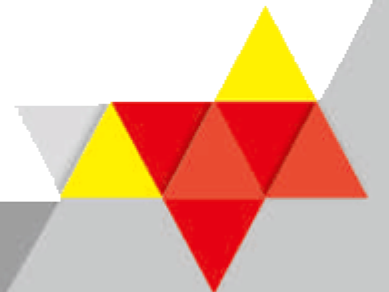
MERGE SORT



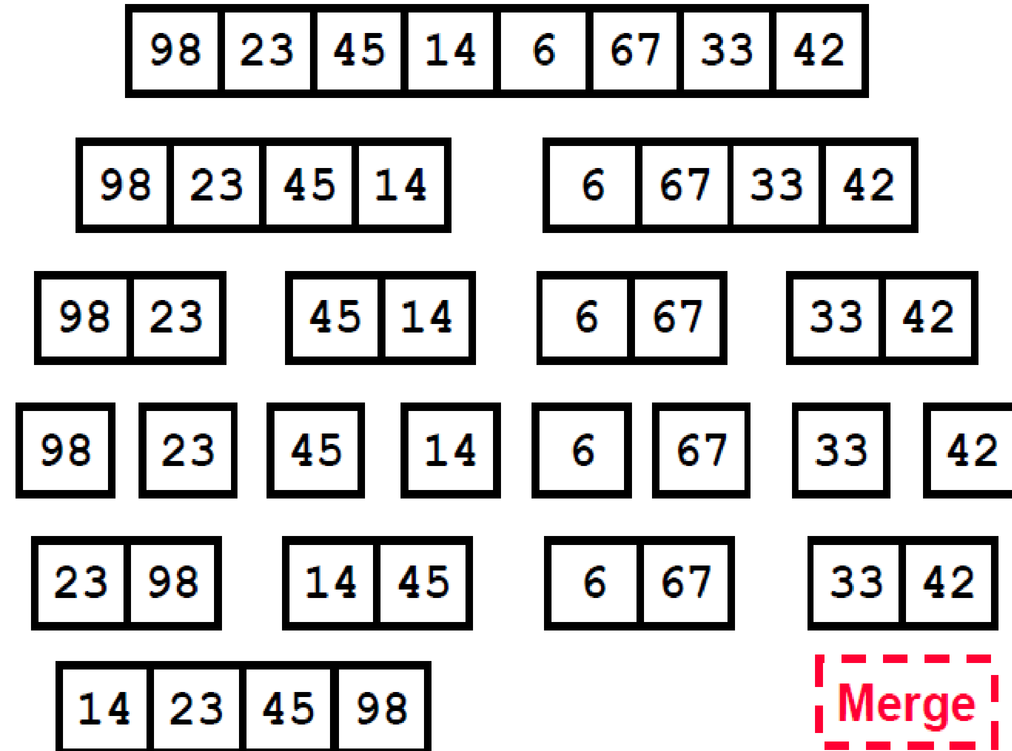


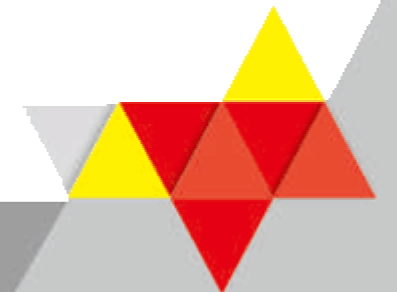
MERGE SORT



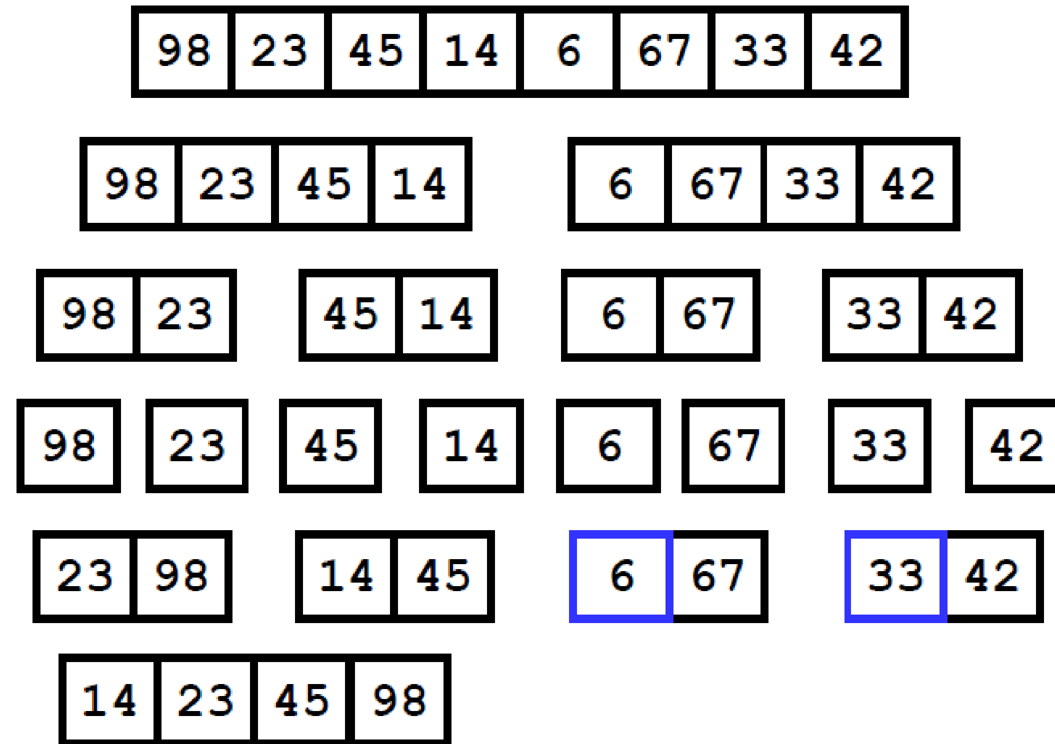


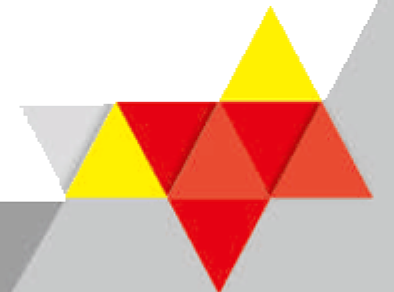
MERGE SORT



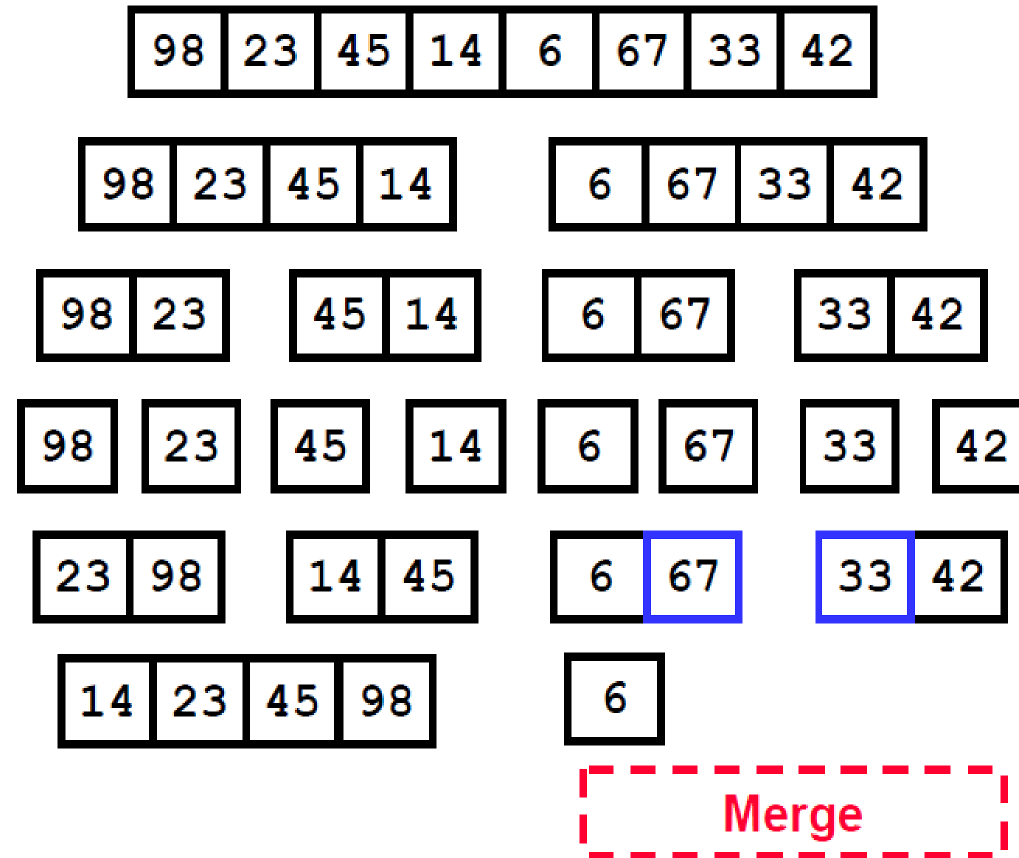


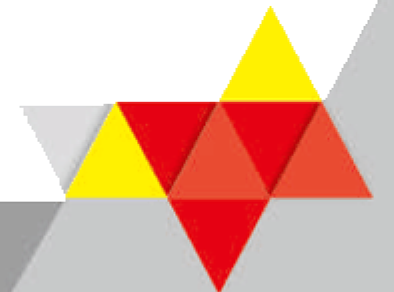
MERGE SORT





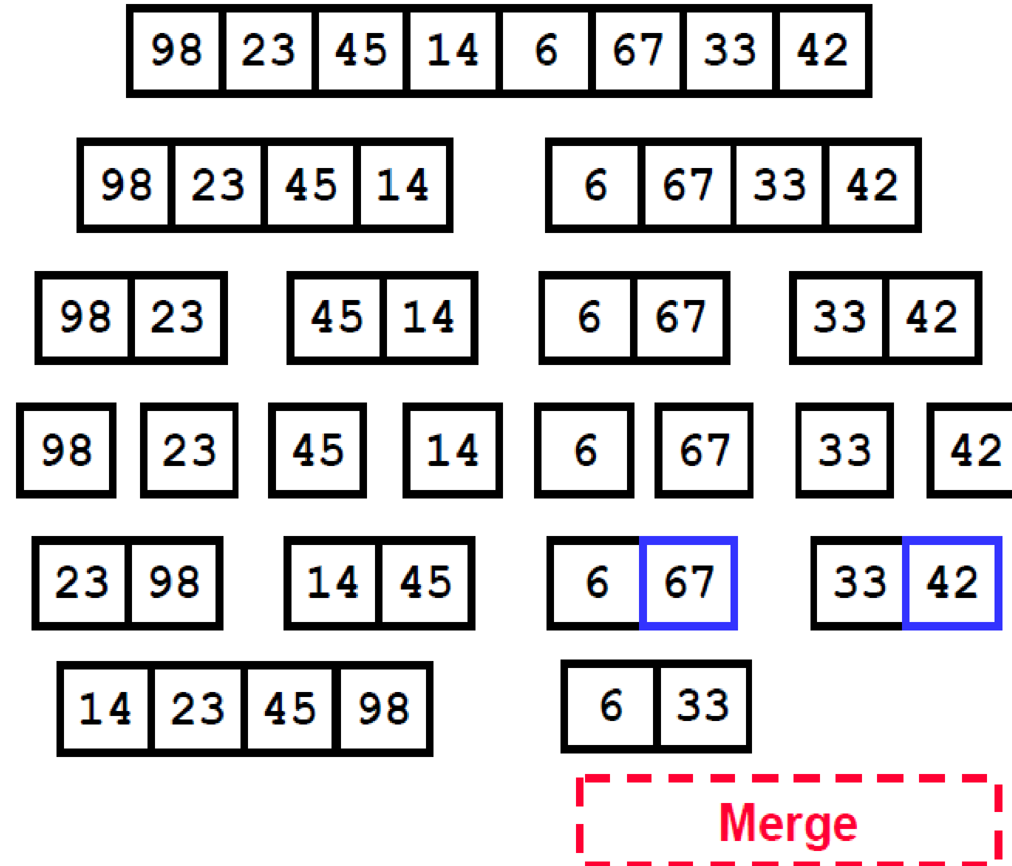
MERGE SORT

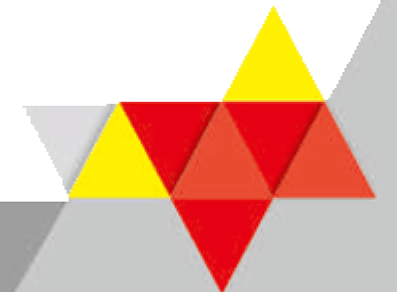




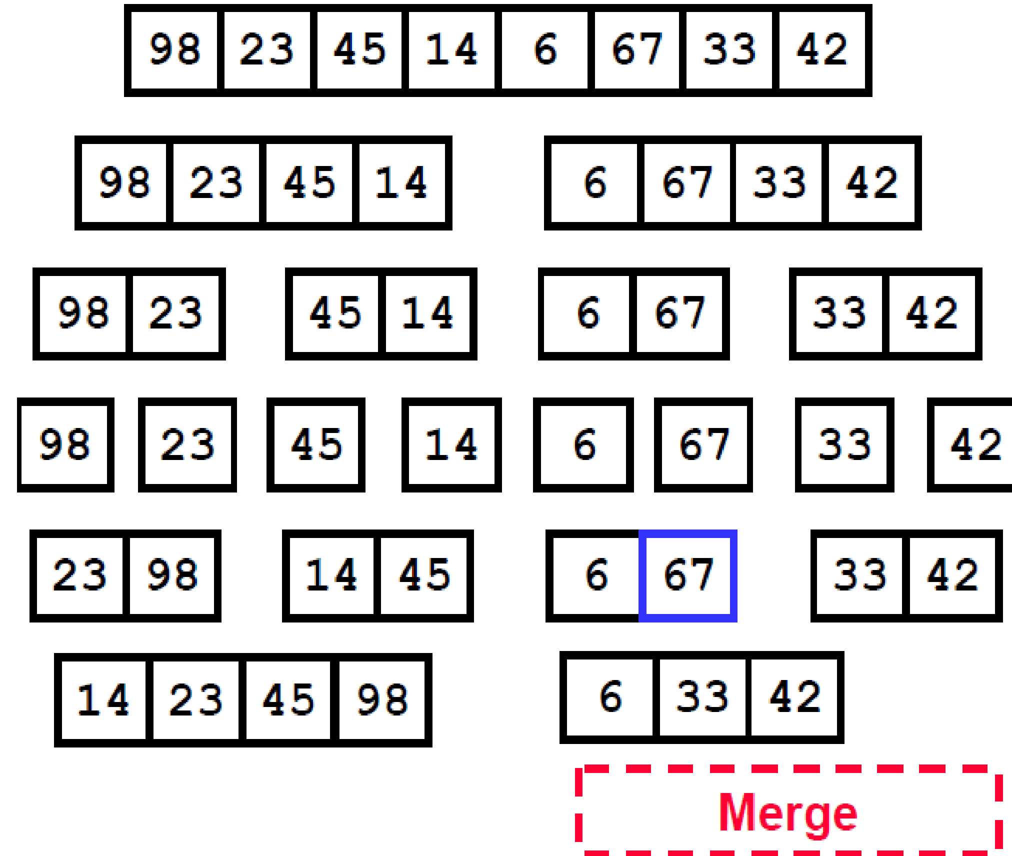
Universitas Dinamika

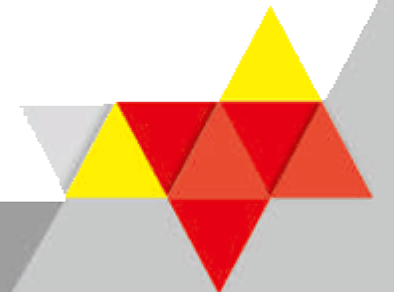
MERGE SORT



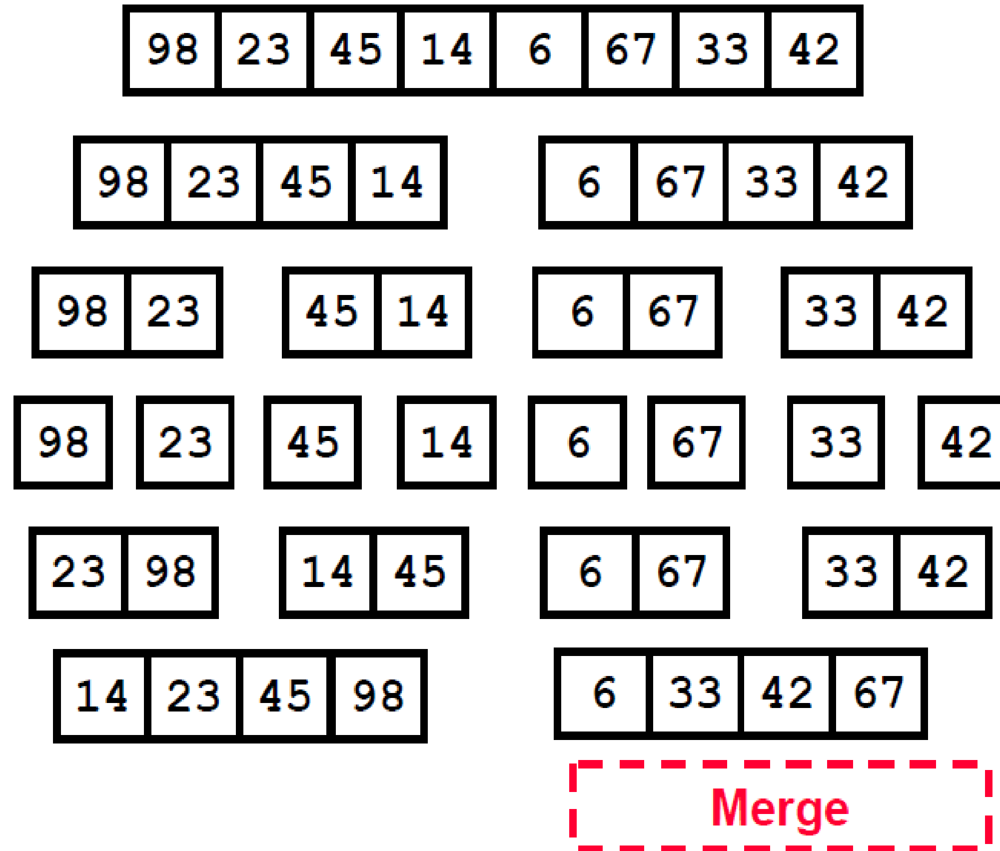


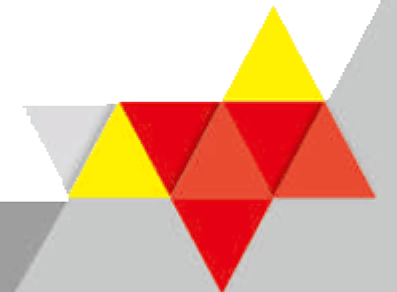
MERGE SORT



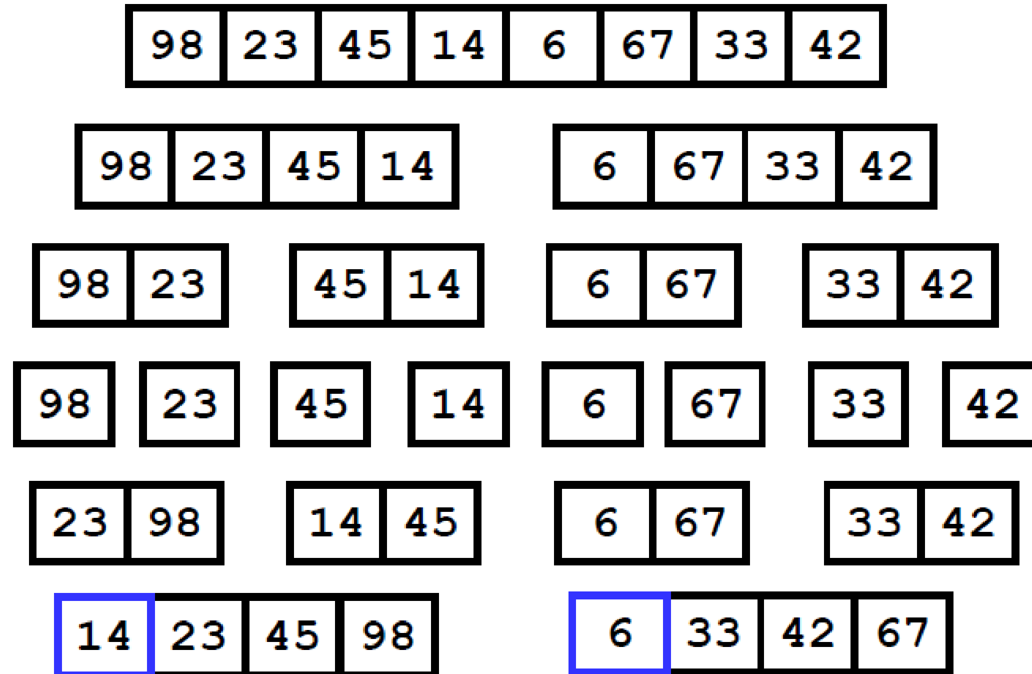


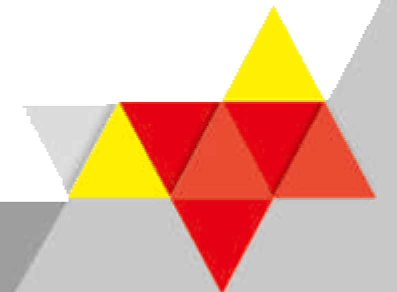
MERGE SORT





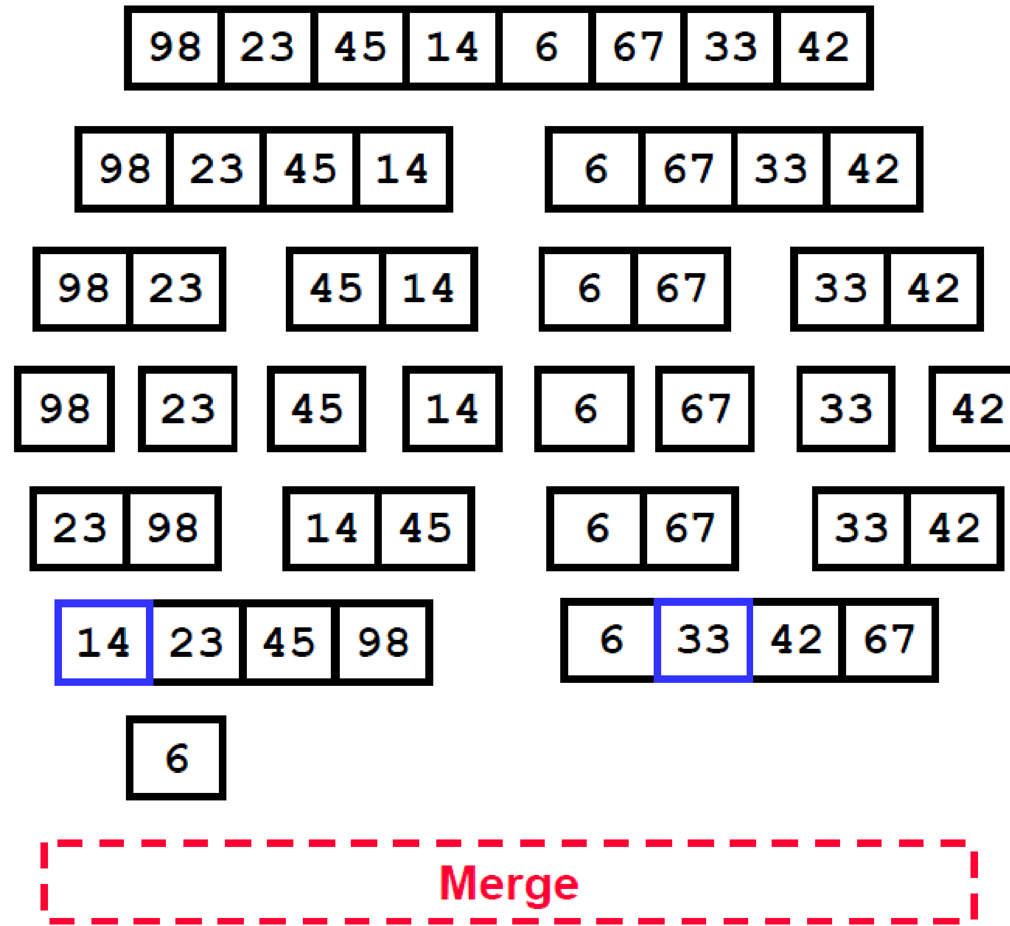
MERGE SORT

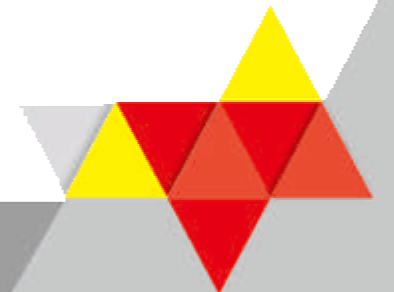




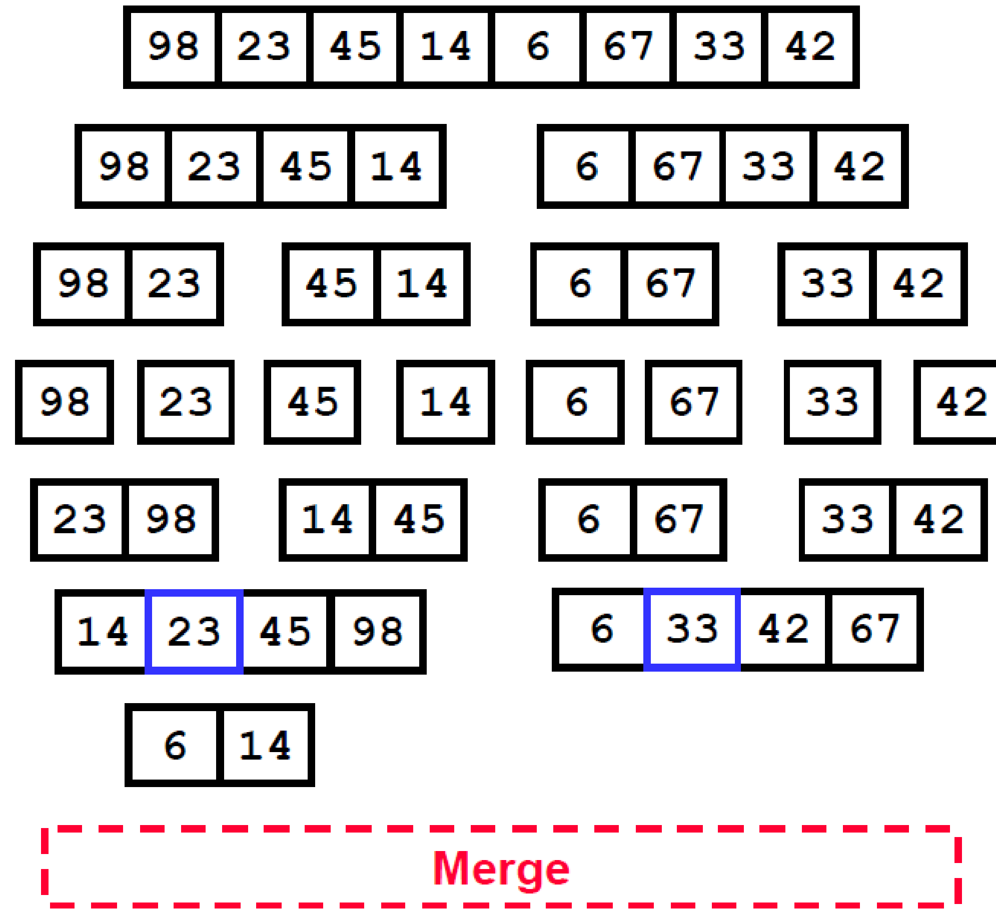
Universitas Dinamika

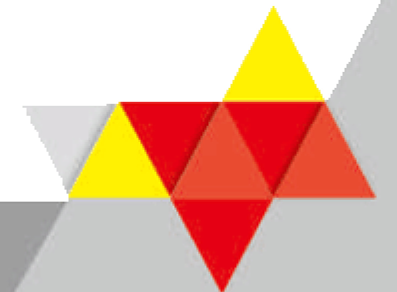
MERGE SORT



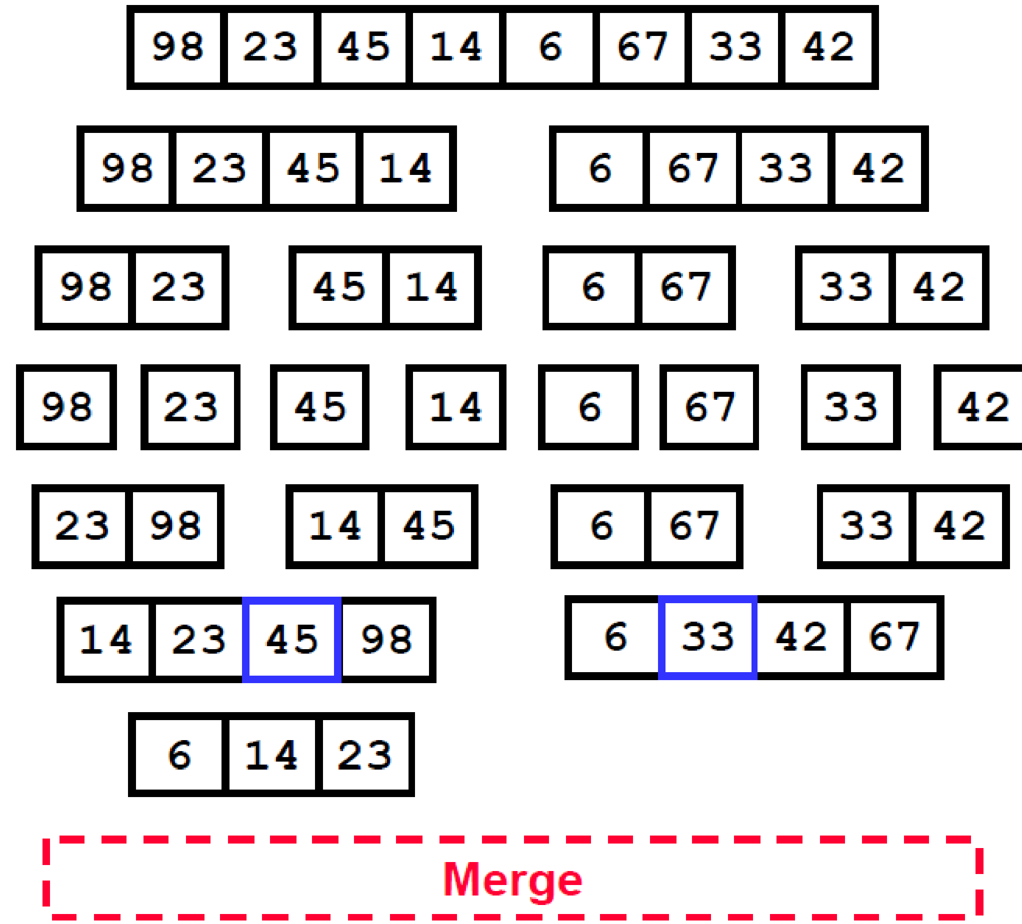


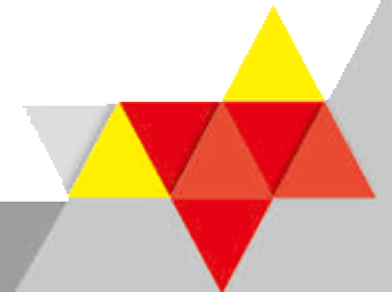
MERGE SORT



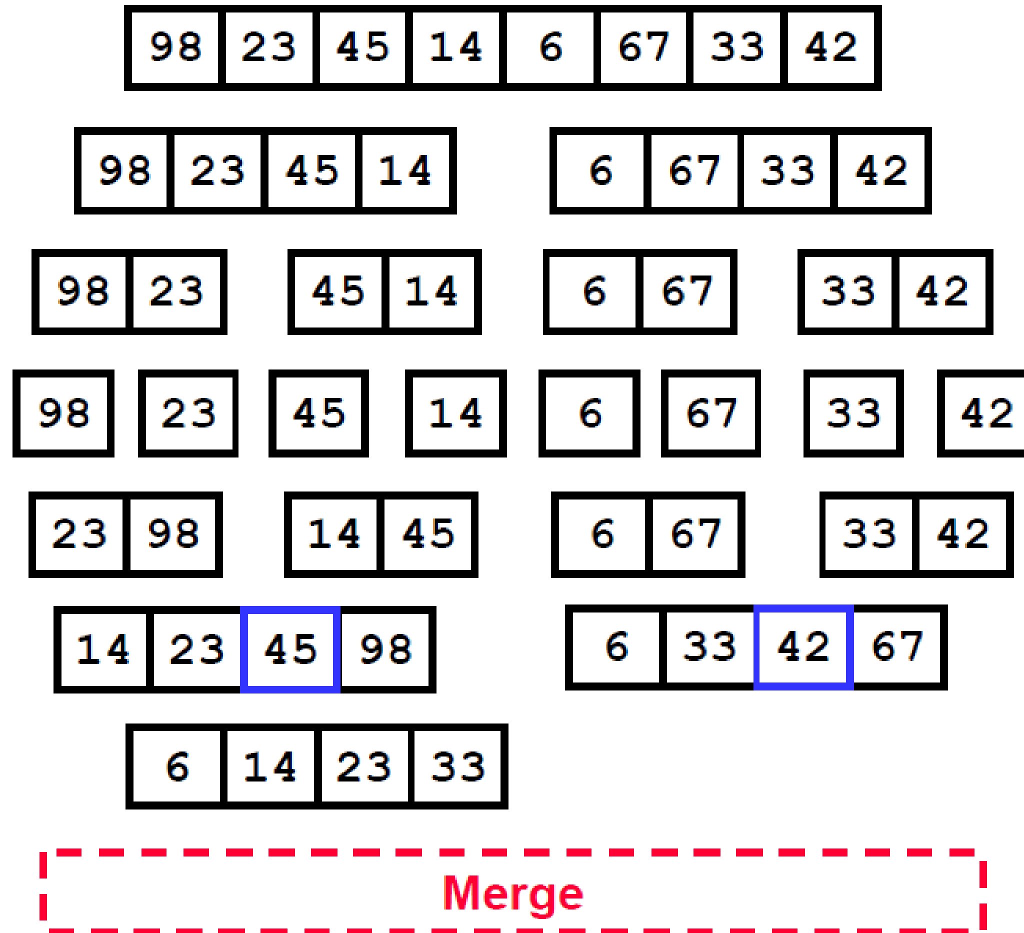


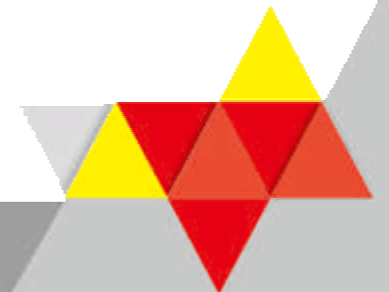
MERGE SORT



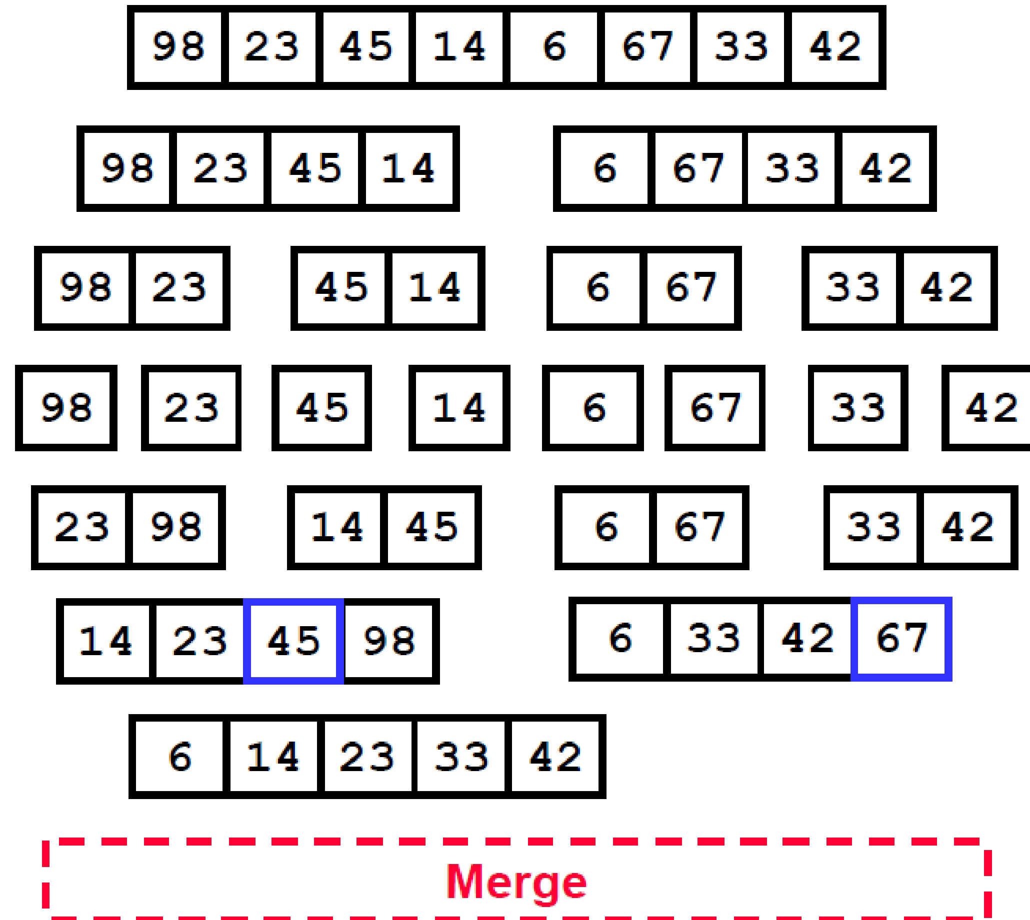


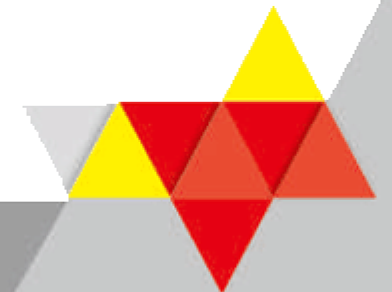
MERGE SORT





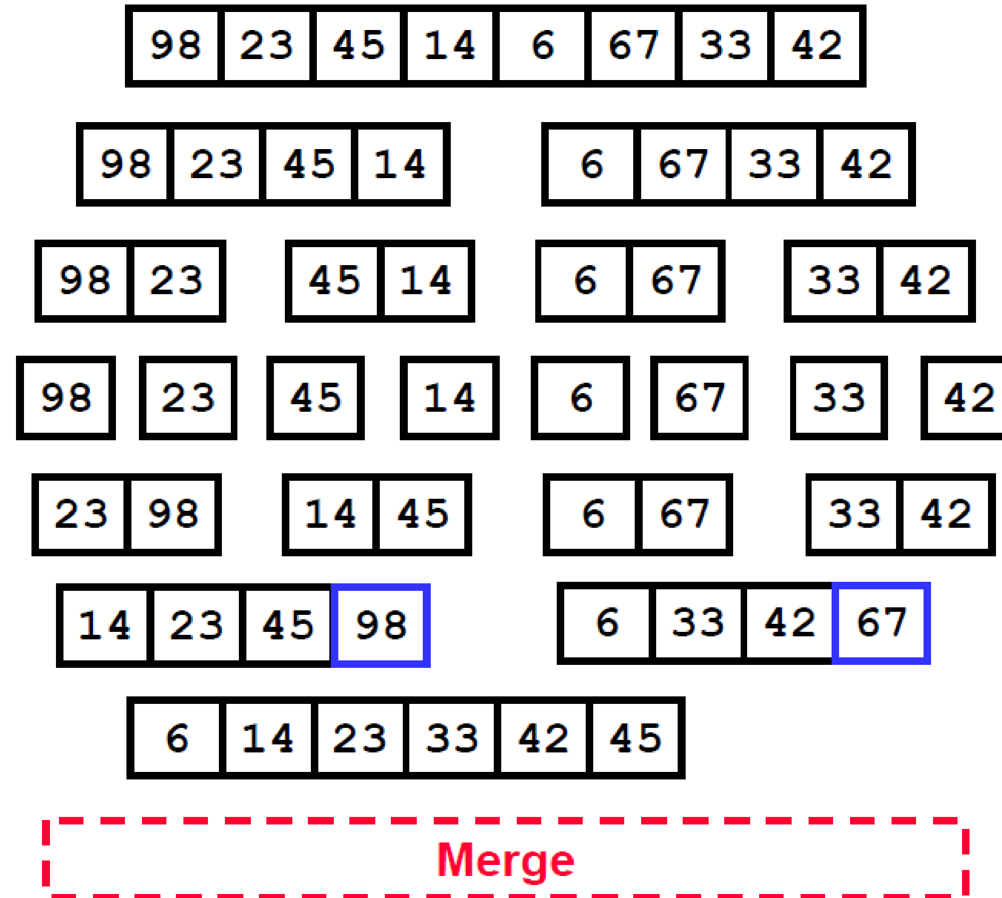
MERGE SORT



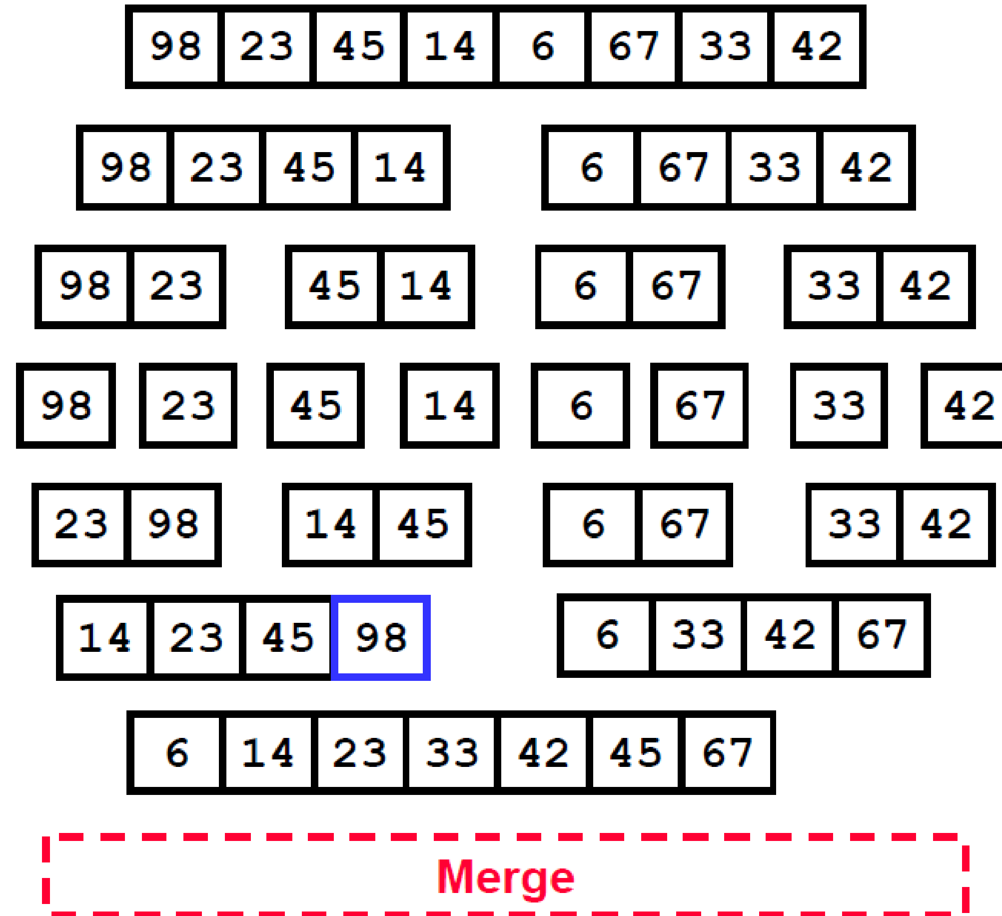


UNIVERSITAS Dinamika

MERGE SORT

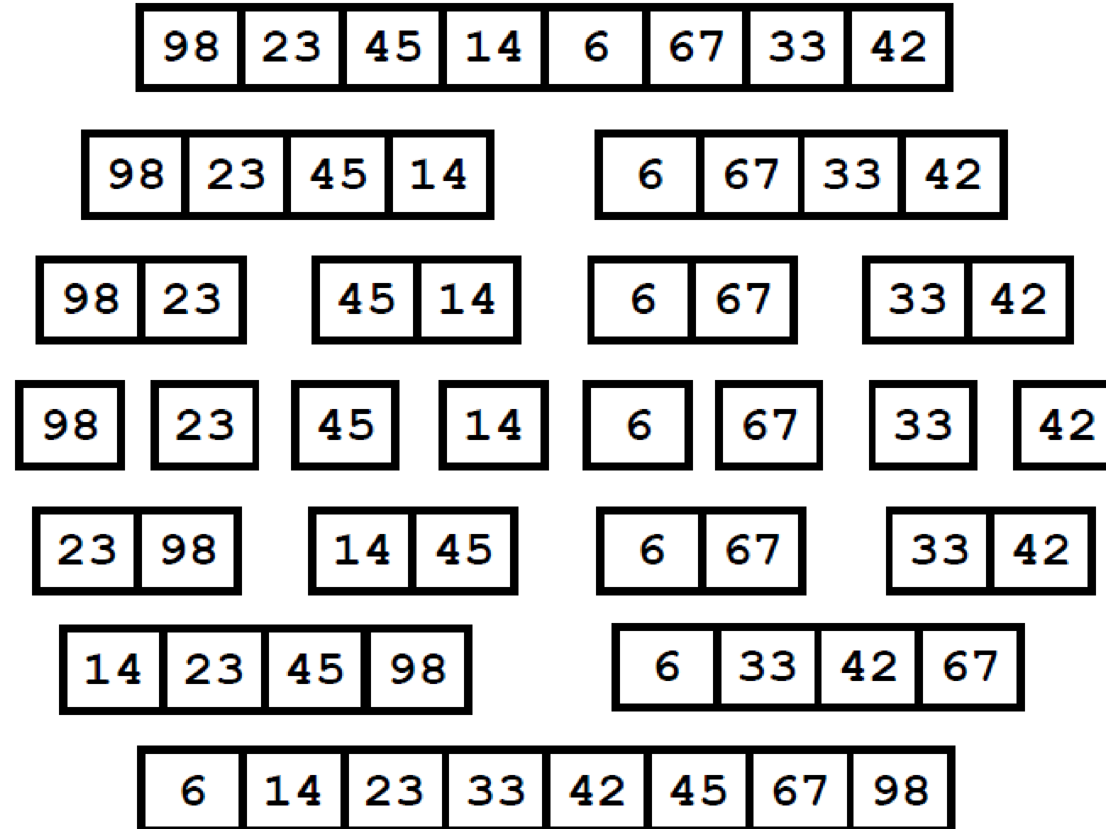


MERGE SORT



UNIVERSITAS Dinamika

MERGE SORT

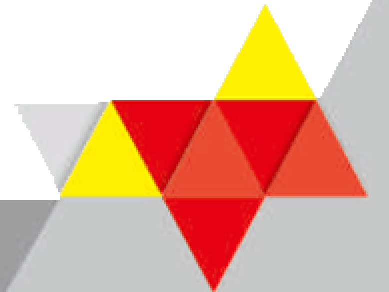


Universitas Dinamika

MERGE SORT

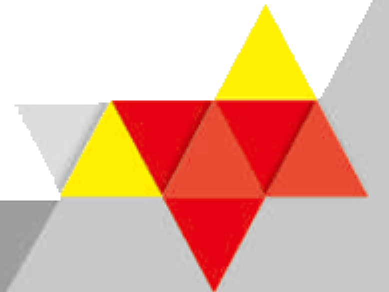
98	23	45	14	6	67	33	42

6	14	23	33	42	45	67	98



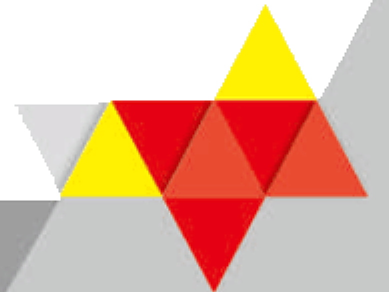
Algoritma Merge Sort

```
void MergeSortRekursif(l, r)
1. jika (l < r) maka kerjakan baris 2-5
2.          med = (l+r) / 2 ;
3.          MergeSortRekursif(l,med);
4.          MergeSortRekursif(med+1,r);
5.          Merge(l,med,r);
```



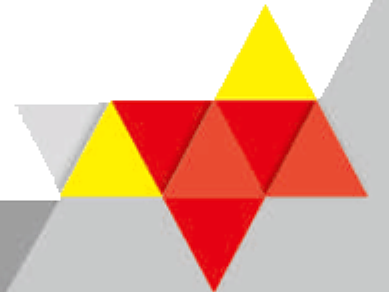
Fungsi Merge

```
void Merge(left, median, right)
1.  kiri1 ← left
2. kanan1 ← median
3. kiri2 ← median+1
4. kanan2 ← right
5. i ← left;
6. selama (kiri1<=kanan1) dan (kiri2<=kanan2) kerjakan 7-1
7.    jika (Data[kiri1] <= Data[kiri2]) kerjakan 8-9
8.           hasil[i] = Data[kiri1];
9.           kiri1++
10.   jika tidak kerjakan baris 11-12
11.          hasil[i] = Data[kiri2];
12.          kiri2++
13.    i++
```



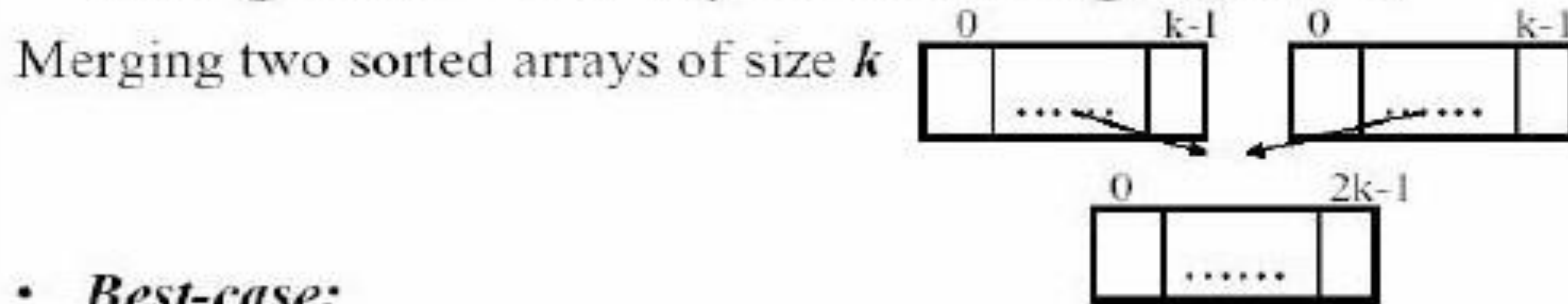
MERGE SORT

```
14. selama (kiri1<=kanan1) kerjakan baris 15-17
15.      hasil[i] = Data[kiri1]
16.      kiri1++
17.      i++

18. selama (kiri2<=kanan2) kerjakan baris 19-21
19.      hasil[i] = Data[kiri2]
20.      i++
21.      kiri2++

22.j ← left
23.selama (j <=right) kerjakan baris 24-25
24.      Data[j] = hasil[j]
25.      j++
```

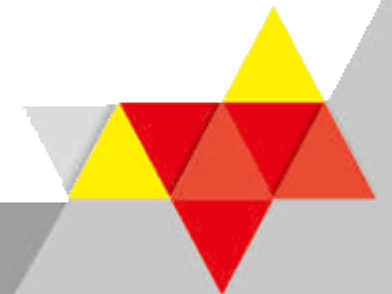


Mergesort – Analysis of Merge (cont.)

Merging two sorted arrays of size ***k***

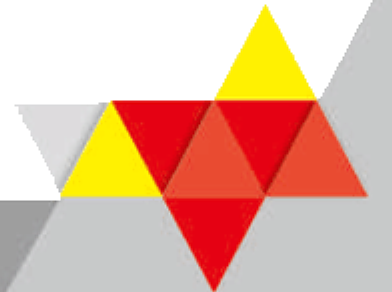


- ***Best-case:***
  - All the elements in the first array are smaller (or larger) than all the elements in the second array.
  - The number of moves: 2k + 2k
  - The number of key comparisons: k
- ***Worst-case:***
  - The number of moves: 2k + 2k
  - The number of key comparisons: 2k-1

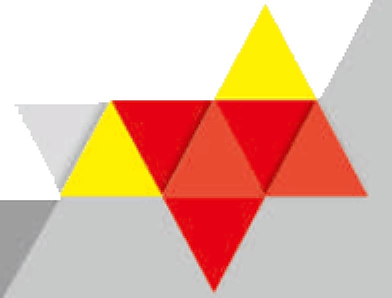


Summary

- **Divide the unsorted collection into two**
- **Until the sub-arrays only contain one element**
- **Then merge the sub-problem solutions together**

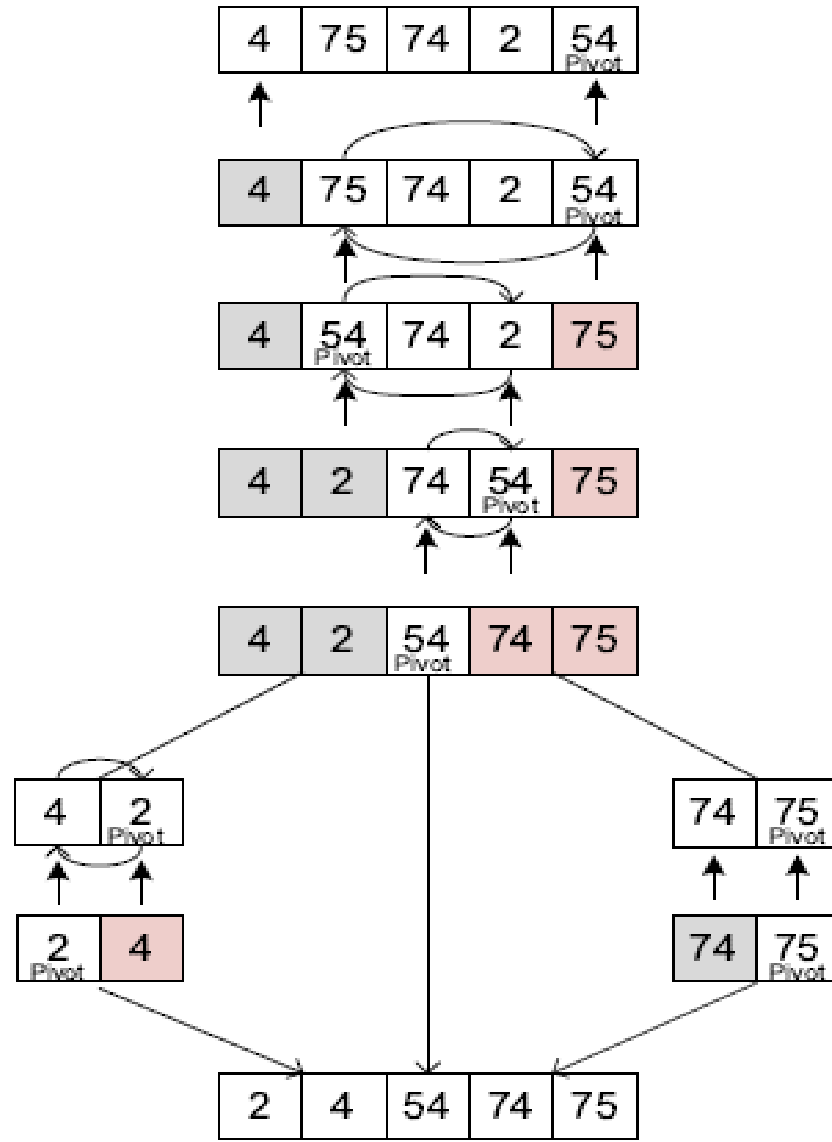


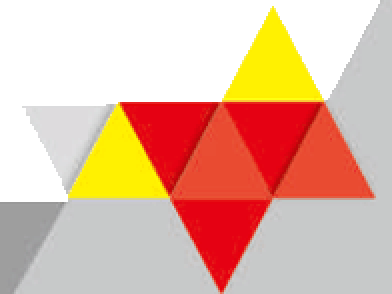
4. QUICK SORT



Universitas Dinamika

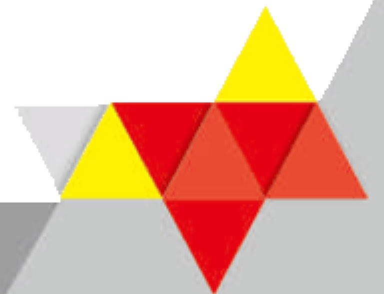
QUICK SORT



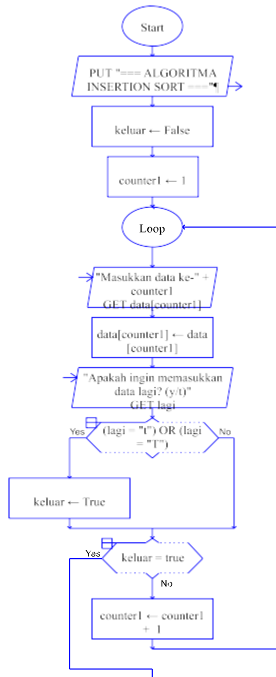


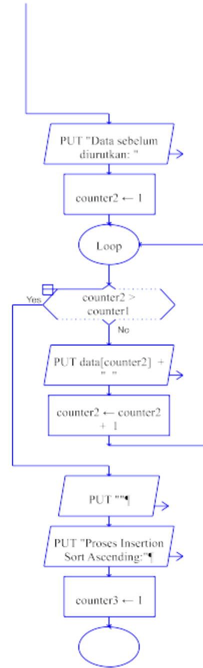
QUICK SORT

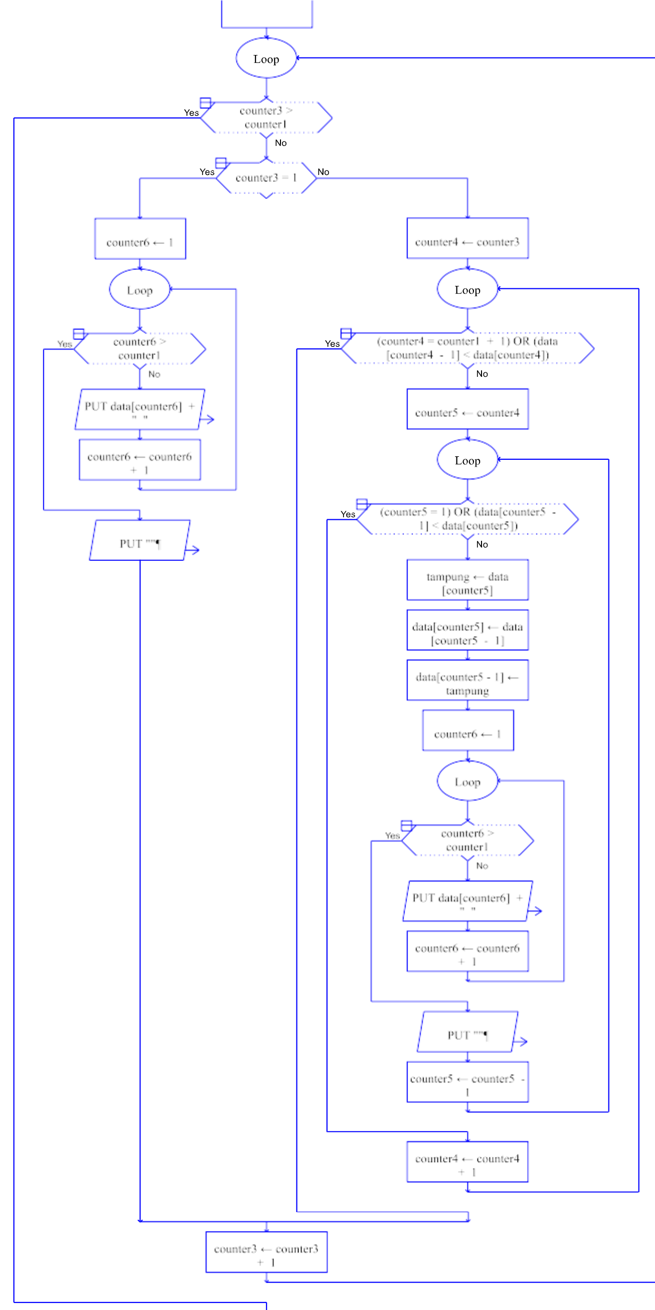
```
1) algorithm QuickSort(list)
2)     Pre:  list ≠ ∅
3)     Post: list has been sorted into values of ascending order
4)     if list.Count = 1 // already sorted
5)        return list
6)     end if
7)     pivot ← MedianValue(list)
8)     for i ← 0 to list.Count−1
9)        if list[i] = pivot
10)          equal.Insert(list[i])
11)       end if
12)       if list[i] < pivot
13)          less.Insert(list[i])
14)       end if
15)       if list[i] > pivot
16)          greater.Insert(list[i])
17)       end if
18)    end for
19)    return Concatenate(QuickSort(less), equal, QuickSort(greater))
20) end Quicksort
```

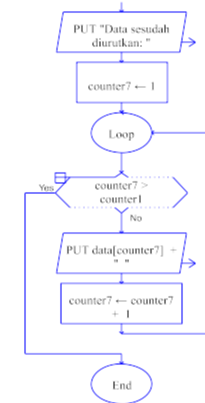


QUICK SORT ASCENDING



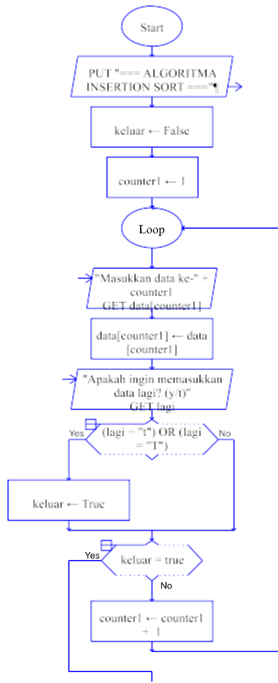


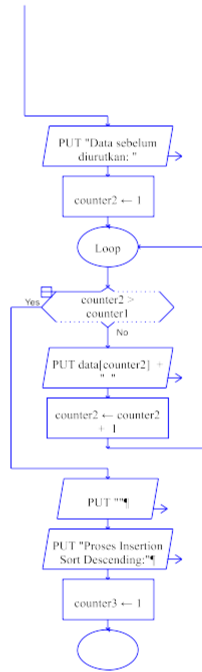


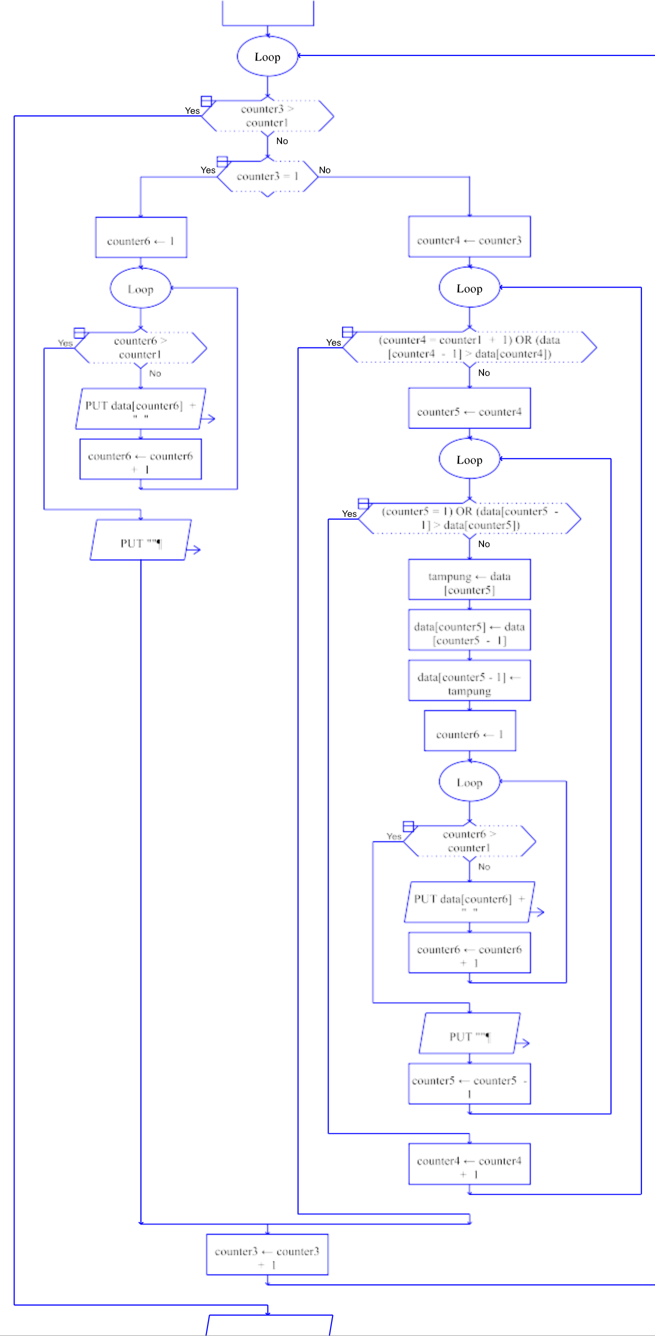


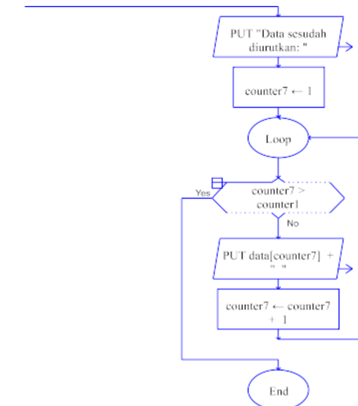
UNIVERSITAS Dinamika

QUICK SORT DESCENDING



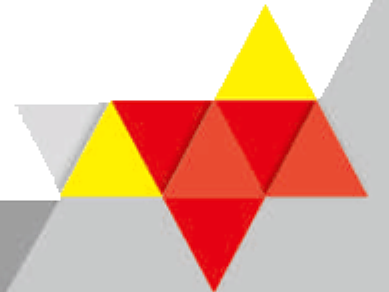






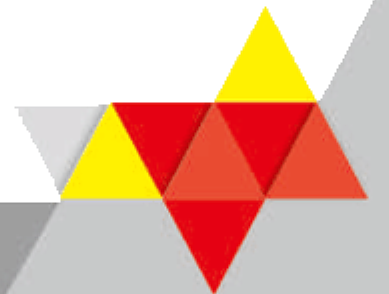
Universitas Dinamika

5. INSERTION SORT



INSERTION SORT

- Metode Penyisipan
- Insertion sort menjadikan bagian kiri dari array terurut sampai keseluruhan dari array terurut.
- Algoritma ini diawali dengan membaca nilai-nilai di sebelah kanan dan membandingkan nilai-nilai di sebelah kiri, setelah ketemu tempatnya maka nilai yang dibaca ini disisipkan.



INSERTION SORT

3	10	4	6	8	9	7	2	1	5

Nilai paling kiri(3) dapat dikatakan diurutkan dengan dirinya sendiri. Sehingga, kita tidak butuh untuk melakukan sesuatu terhadap nilai ini.

INSERTION SORT

3	10	4	6	8	9	7	2	1	5

3	10	4	6	8	9	7	2	1	5

Cek dan lihat nilai kedua(10) apakah lebih kecil dari nilai pertama(3). Jika ya, tukar kedua nilai tersebut. Akan tetapi dalam hal ini kita tidak menukarnya.

Universitas Dinamika

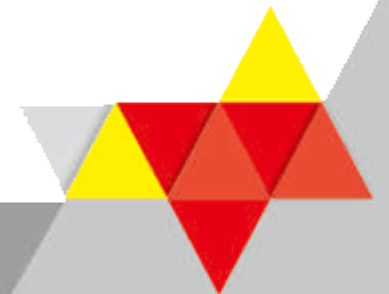
INSERTION SORT

3	10	4	6	8	9	7	2	1	5

Angka biru/wilayah abu-abu (dua nilai pertama) secara relatif sudah dalam kondisi terurut.

3	10	4	6	8	9	7	2	1	5

Selanjutnya, kita butuh menyisipkan nilai ketiga(4) dalam daerah abu-abu agar setelah penyisipan, daerah abu-abu tetap secara relatif dalam kondisi terurut.



INSERTION SORT

Pertama : Simpan nilai ketiga dalam sebuah variabel.

4

3	10		6	8	9	7	2	1	5

Kedua : Lihat nilai di sebelah kirinya (10), jika lebih besar geser ke kanan. Langkah ini diulang sampai ditemukan nilai lebih kecil.

4

3		10	6	8	9	7	2	1	5

Ketiga : Sisipkan nilai 4 pada posisi yang sesuai.

3	4	10	6	8	9	7	2	1	5

INSERTION SORT

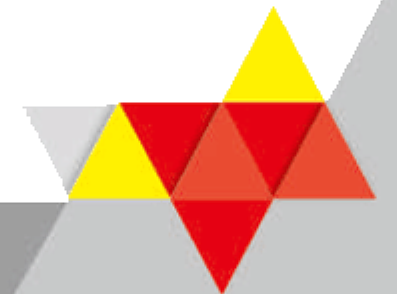
Sekarang ketiga nilai pertama secara relatif dalam kondisi terurut, dan kita telah menyisipkan nilai keempat dalam ketiga nilai tsb, sehingga keempat nilai pertama secara relatif telah terurut.

3	4	6	10	8	9	7	2	1	5

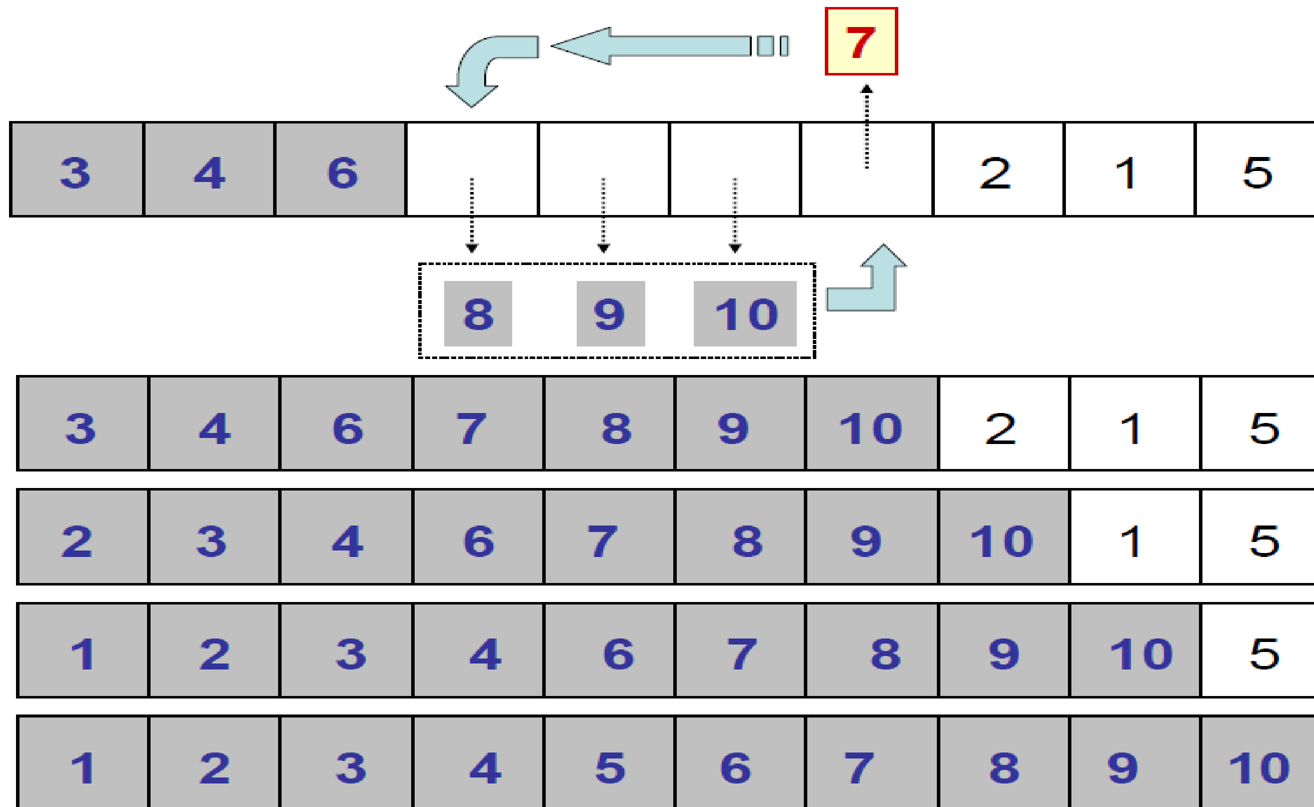
Ulangi proses sampai nilai terakhir telah disisipkan.

3	4	6	8	10	9	7	2	1	5

3	4	6	8	9	10	7	2	1	5

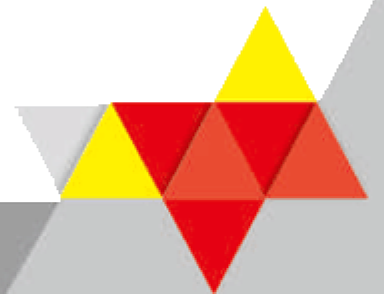


INSERTION SORT



Algoritma

1. Ulangi langkah 2-6 untuk j=1 s/d n-1
2. i <- j – 1
3. tmp <- data[j]
4. Ulangi langkah 5-6 selama i>=0 dan tmp<data[i]
5. data[i+1]=data[i]
6. i--
7. data[i+1]=tmp



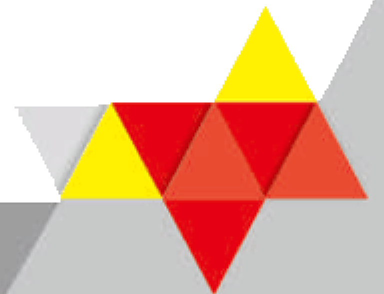
Insertion Sort → Analysis

- Assuming there are n elements in the array, we must index through n - 1 entries.
- For each entry, we may need to examine and shift up to n - 1 other entries, resulting in a $\boldsymbol{O}(n^2)$ algorithm.
- The insertion sort is an *in-place* sort. That is, we sort the array in-place. No extra memory is required.
- The insertion sort is also a *stable* sort. Stable sorts retain the original ordering of keys when identical keys are present in the input data.

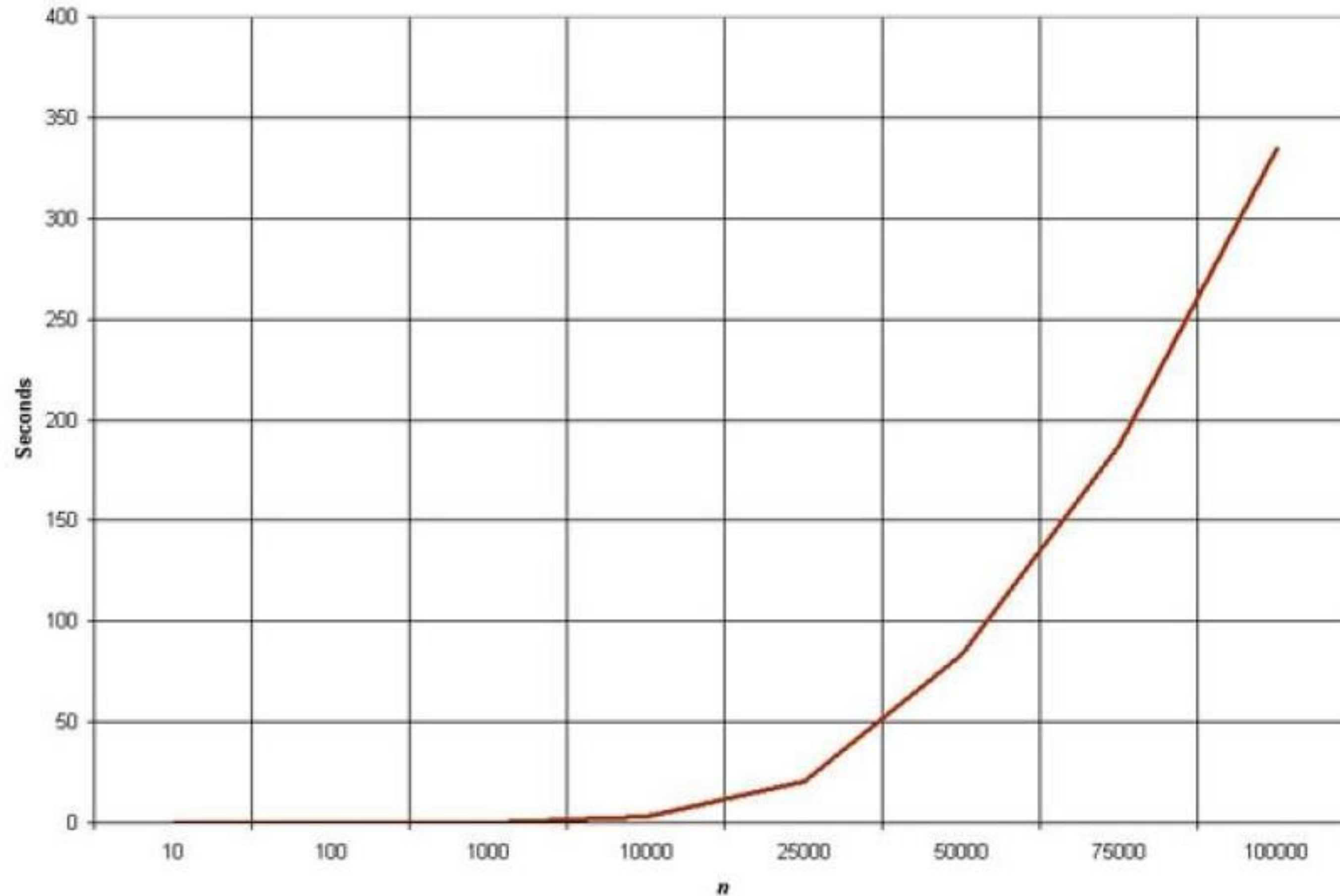
INSERTION SORT

Insertion Sort → Analysis

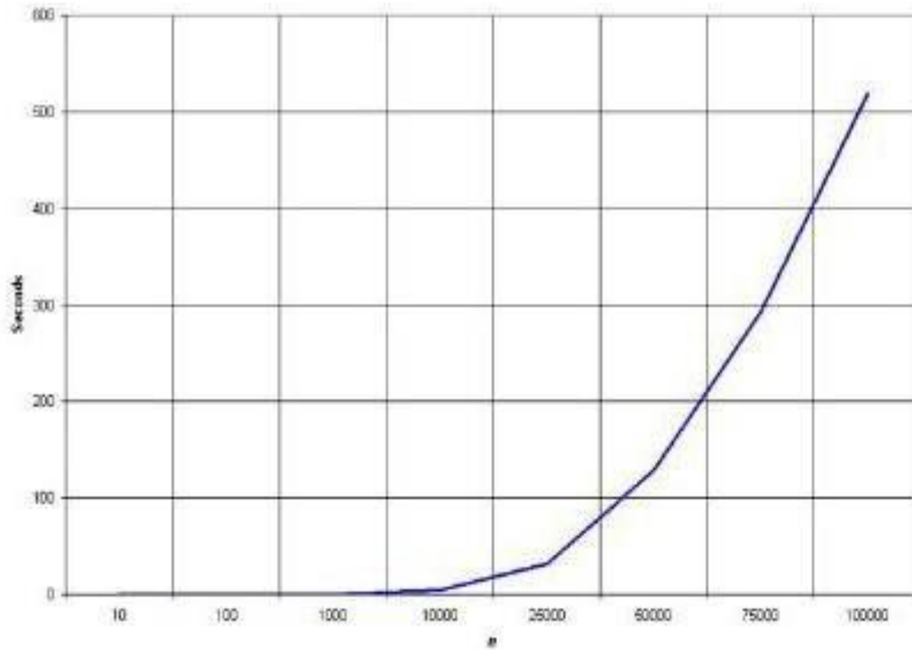
- **best case:**
  - about N comparisons, 0 moves, (input already sorted)
- **worst case:**
  - about $N^2/2$ comparisons, $N^2/2$ moves, (input sorted in reverse order)
- **average case:**
  - about $N^2/4$ comparisons, $N^2/4$ moves
- **notes:**
  - very efficient when input is "almost sorted";
  - moving a record is about half the work of exchanging two records, so average case is equivalent to roughly $N^2/8$ exchanges.

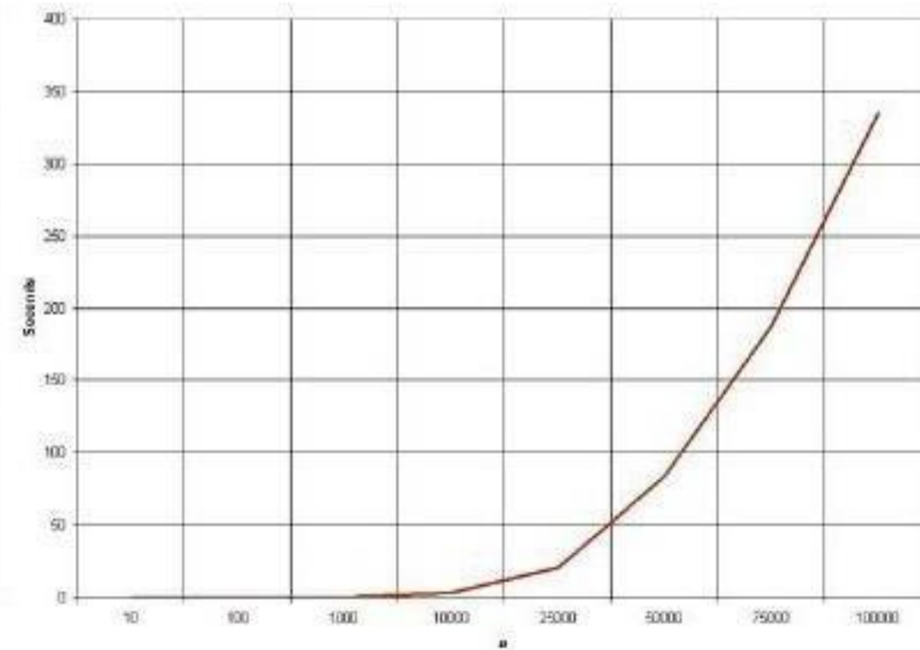


INSERTION SORT

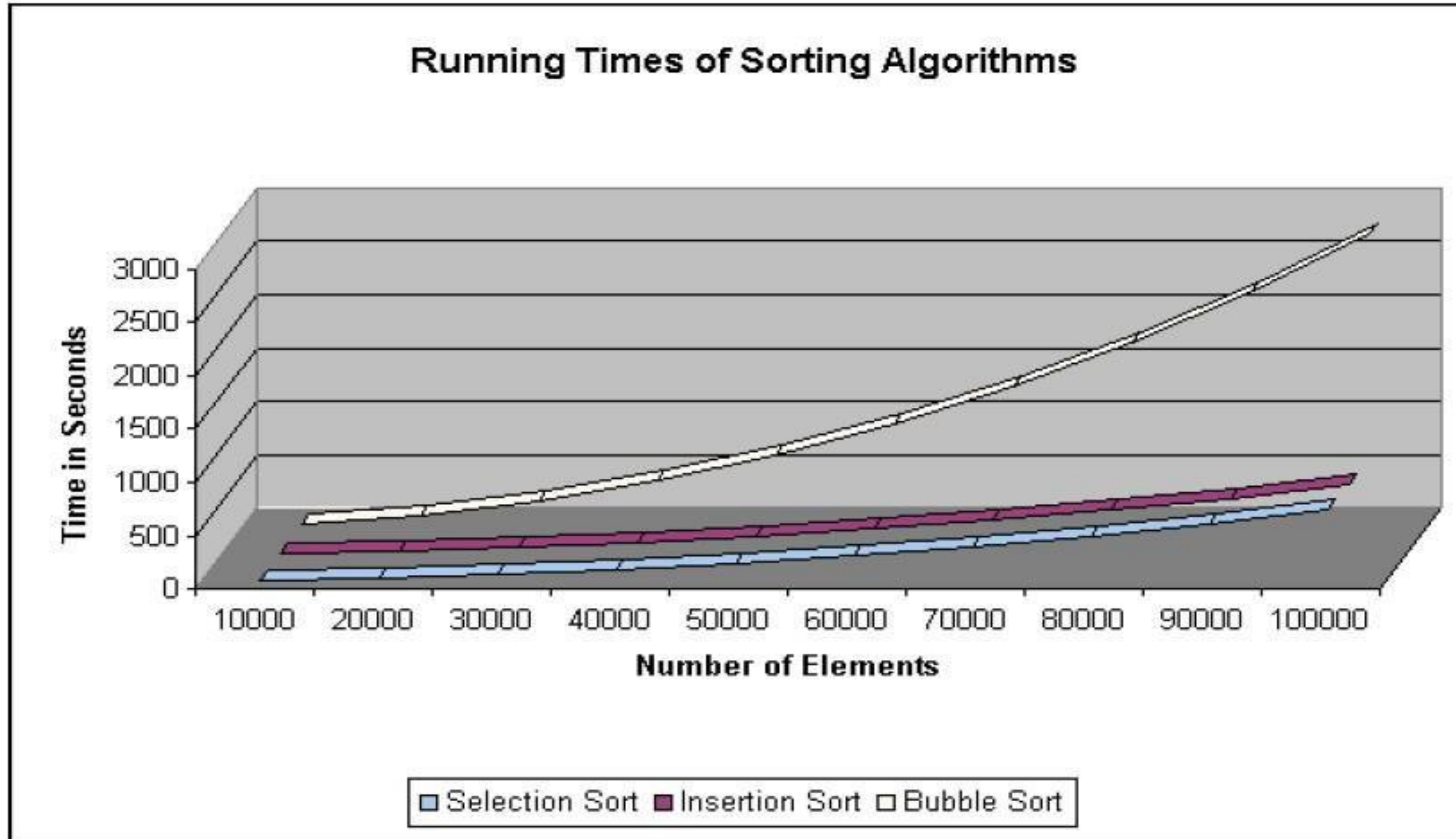


INSERTION SORT

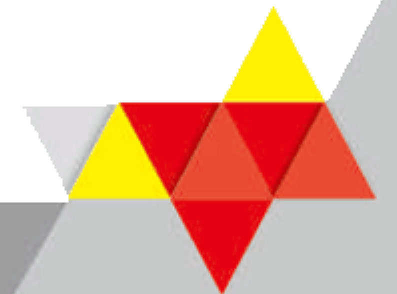




INSERTION SORT

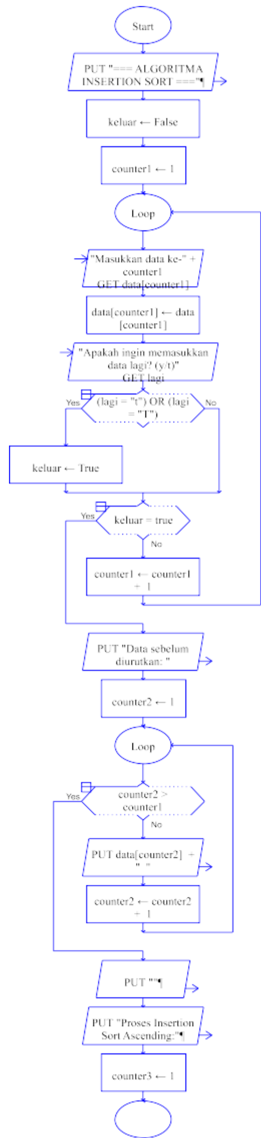


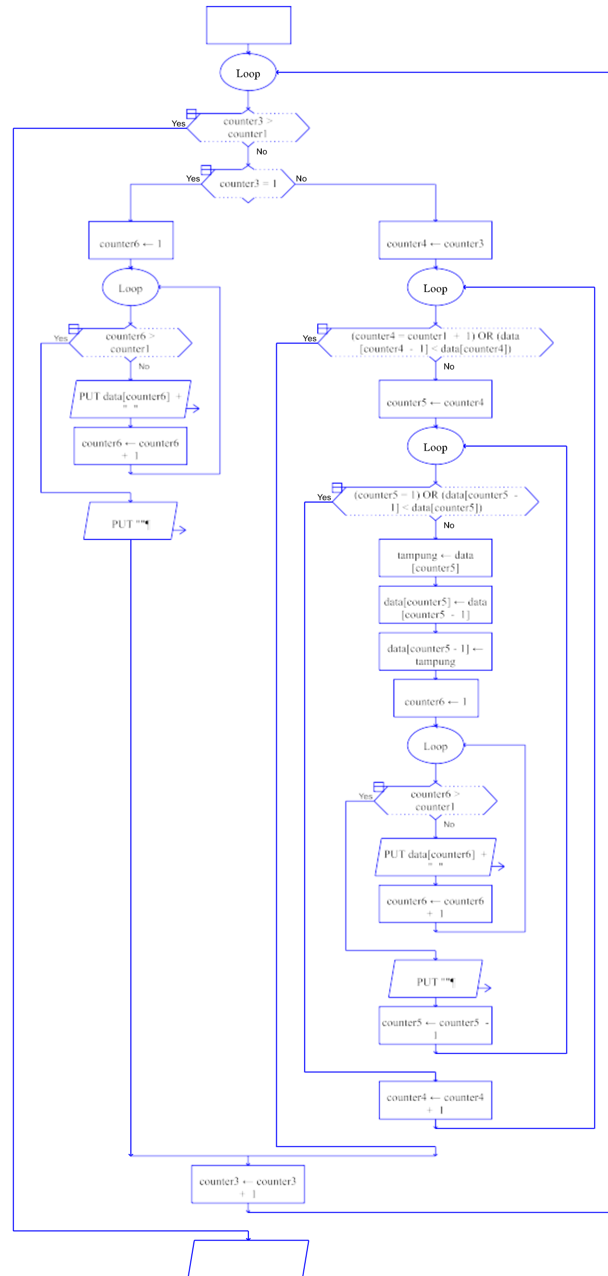
Selection Sort yields a 60% performance improvement over the Bubble Sort

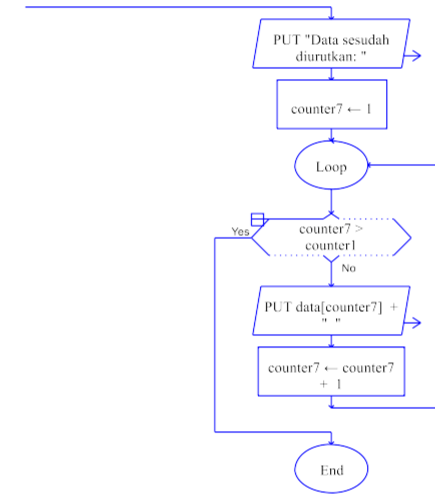


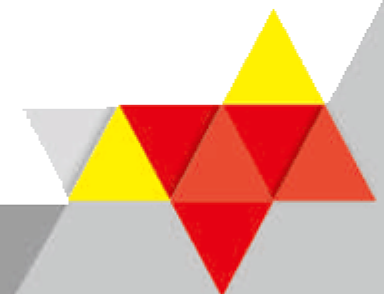
UNIVERSITAS Dinamika

INSERTION SORT ASCENDING



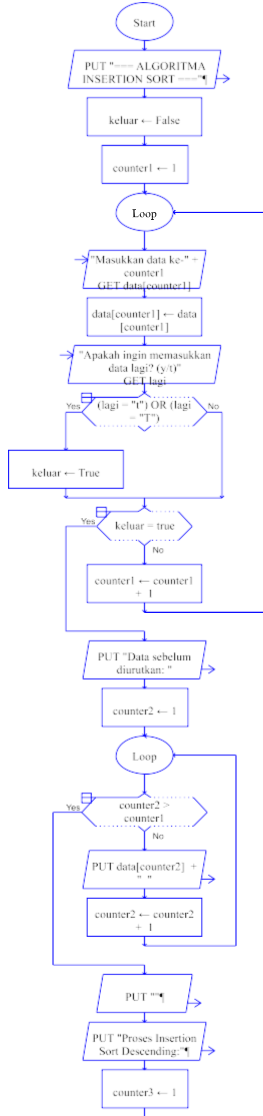


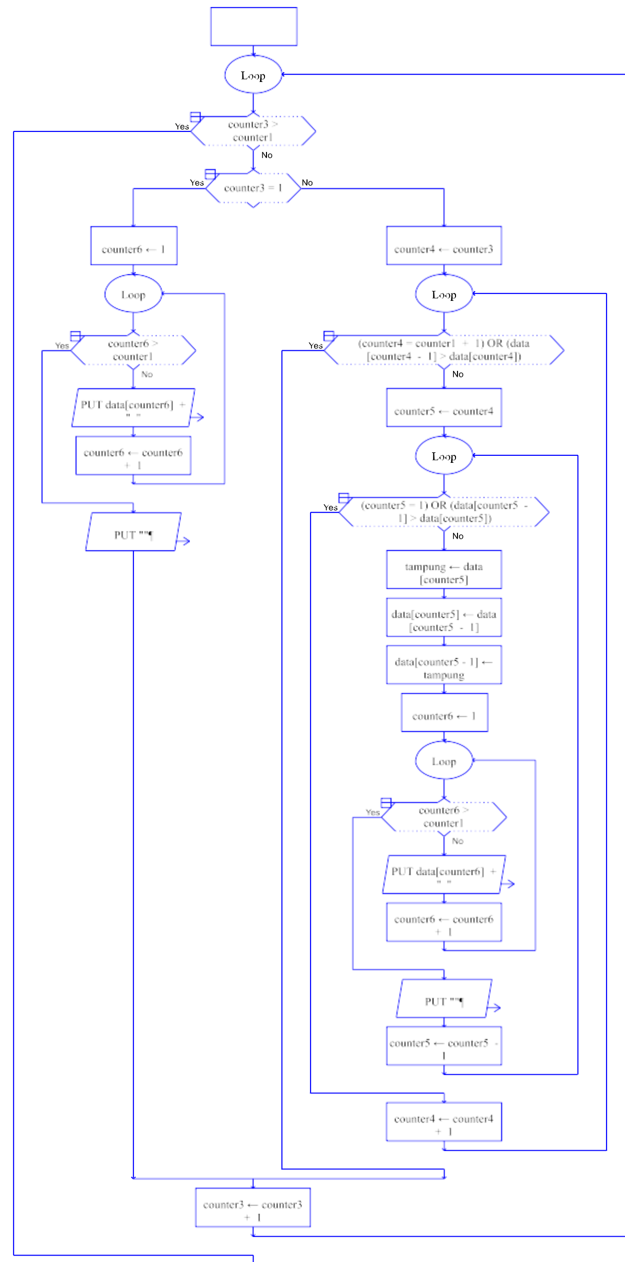


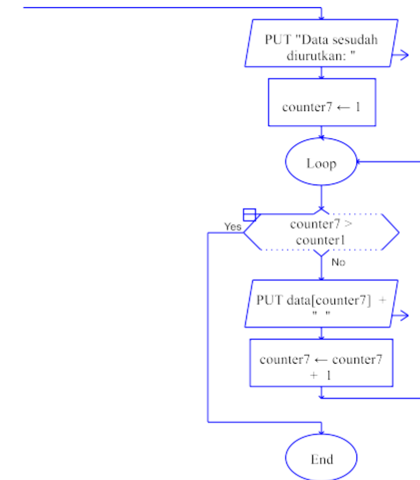


UNIVERSITAS Dinamika

INSERTION SORT DESCENDING



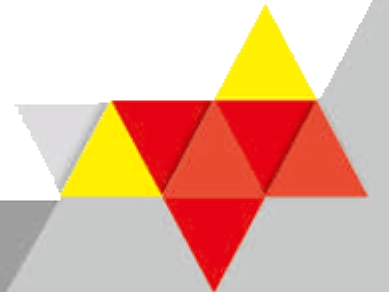




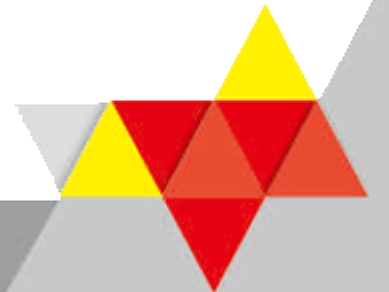
UNIVERSITAS Dinamika

PERBANDINGAN SORTING

- Bubble sort uses about $N^2/2$ comparisons and $N^2/2$ exchanges on the average and in the worst case.
- Selection sort uses about $N^2/2$ comparisons and N exchanges on the average, twice as many in the worst case.
- Insertion sort uses about $N^2/4$ comparisons and $N^2/8$ exchanges and is linear for "almost sorted" files.

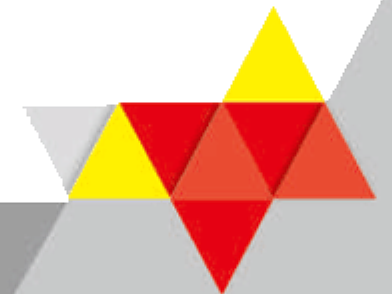


6. SHELL SORT



SHELL SORT

- Metode ini disebut juga dengan metode pertambahan menurun *(diminishing increment sort)*. Metode ini dikembangkan oleh Donald L. Shell pada tahun 1959, sehingga sering disebut dengan Metode Shell Sort.
- Metode ini mengurutkan data dengan cara membandingkan suatu data dengan data lain yang memiliki jarak tertentu – sehingga membentuk sebuah sub-list-, kemudian dilakukan penukaran bila diperlukan



SHELL SORT

- Jarak yang dipakai didasarkan pada *increment value* atau *sequence number* k
- Misalnya Sequence number yang dipakai adalah 5,3,1. Tidak ada pembuktian di sini bahwa bilangan-bilangan tersebut adalah sequence number terbaik
- Setiap sub-list berisi setiap elemen ke-k dari kumpulan elemen yang asli

SHELL SORT

- Contoh: Jika k = 5 maka sub-list nya adalah sebagai berikut :

  - s[0] s[5] s[10] ...

  - s[1] s[6] s[11] ...

  - s[2] s[7] s[12] ...

  - dst

- Begitu juga jika k = 3 maka sub-list nya adalah:

  - s[0] s[3] s[6] ...

  - s[1] s[4] s[7] ...

  - dst

SHELL SORT

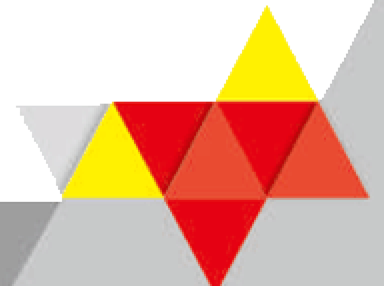
- Buatlah sub-list yang didasarkan pada jarak (Sequence number) yang dipilih
- Urutkan masing-masing sub-list tersebut
- Gabungkan seluruh sub-list

Let's see this algorithm in action

SHELL SORT

- Urutkan sekumpulan elemen di bawah ini , misalnya diberikan sequence number : 5, 3, 1

30	62	53	42	17	97	91	38
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]



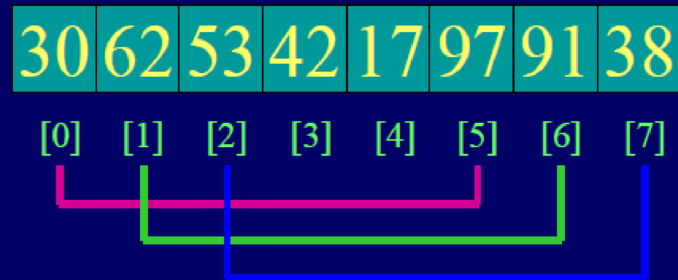
SHELL SORT

Proses Shell Sort k=5

30 62 53 42 17 97 91 38

Step 1: Buat sub list k = 5

S[0] S[5]
S[1] S[6]
S[2] S[7]
S[3]



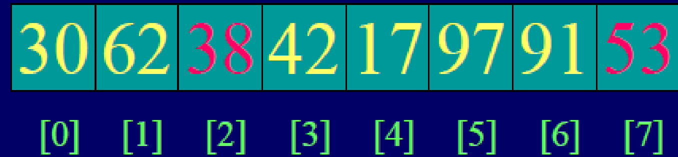
Step 2 - 3: Urutkan sub list & gabungkan

S[0] < S[5] **30, 97 OK**

S[1] < S[6] **62, 91 OK**

S[2] > S[7] **53, 38 not OK.**

Swap them **38, 53**



UNIVERSITAS Dinamika

SHELL SORT

Proses Shell Sort utk k=3

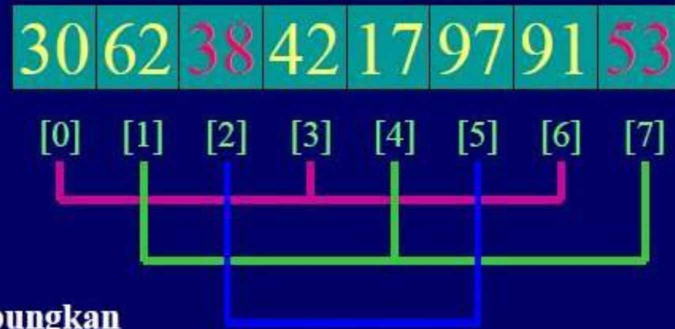
30 62 53 42 17 97 91 38

Step 1: Buat sub list k = 3

S[0] S[3] S[6]

S[1] S[4] S[7]

S[2] S[5]



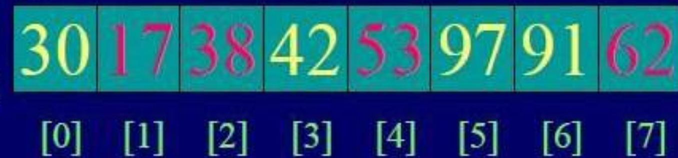
Step 2 - 3: Urutkan sub list & gabungkan

S[0] S[3] S[6] **30, 42, 91 OK**

S[1] S[4] S[7] **62, 17, 53 not OK**

SORT them **17, 53, 62**

S[2] S[5] **38, 97 OK**



SHELL SORT

Shell Sort Process k=1

30 62 53 42 17 97 91 38

Step 1: Buat sub list k =1

S[0]	S[1]	S[2]	S[3]	S[4]	S[5]	S[6]	S[7]
30	17	38	42	53	97	91	62
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]

Step 2 - 3: Urutkan sub list & gabungkan

Sorting akan seperti insertion sort

17	30	38	42	53	62	91	97
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]

DONE

Pemilihan Sequence Number

- Disarankan jarak mula-mula dari data yang akan dibandingkan adalah: **`N / 2`**.
- Pada proses berikutnya, digunakan jarak **`(N / 2) / 2`** atau **`N / 4`**.
- Pada proses berikutnya, digunakan jarak **`(N / 4) / 2`** atau **`N / 8`**.
- Demikian seterusnya sampai jarak yang digunakan adalah 1.

Urutan prosesnya...

- Untuk jarak N/2 :

  - Data pertama (i=0) dibandingkan dengan data dengan jarak N / 2. Apabila data pertama lebih besar dari data ke N / 2 tersebut maka kedua data tersebut ditukar.

  - Kemudian data kedua (i=1) dibandingkan dengan jarak yang sama yaitu N / 2 = elemen ke-(i+N/2)

  - Demikian seterusnya sampai seluruh data dibandingkan sehingga semua data ke-i selalu lebih kecil daripada data ke-(i + N / 2).
- Ulangi langkah-langkah di atas untuk jarak = N / 4 → lakukan pembandingan dan pengurutan sehingga semua data ke-i lebih kecil daripada data ke-(i + N / 4).
- Ulangi langkah-langkah di atas untuk jarak = N / 8 → lakukan pembandingan dan pengurutan sehingga semua data ke-i lebih kecil daripada data ke-(i + N / 8).
- Demikian seterusnya sampai jarak yang digunakan adalah 1 atau data sudah terurut (`did_swap = false`)

Algoritma Metode Shell Sort

1. **Jarak ← N**
2. Selama **Jarak > 1** kerjakan baris 3 sampai dengan 12
3. **Jarak ← Jarak / 2.**
4. **did_swap ← true**
5. Kerjakan baris 6 sampai dengan 12 selama **did_swap = true**
6. **did_swap ← false**
7. **i ← 0**
8. Selama **i < (N – Jarak)** kerjakan baris 9 sampai dengan 12
9. Jika **Data[i] > Data[i + Jarak]** kerjakan baris 10 dan 11
10. **tukar(Data[i], Data[i + Jarak])**
11. **did_swap ← true**
12. **i ← i + 1**

Analisis Metode Shell Sort

- Running time dari metode Shell Sort bergantung pada pemilihan sequence number-nya.
- Disarankan untuk memilih sequence number dimulai dari N/2, kemudian membaginya lagi dengan 2, seterusnya hingga mencapai 1.
- Shell sort menggunakan 3 nested loop, untuk merepresentasikan sebuah pengembangan yang substansial terhadap metode insertion sort

Pembandingan Running time (millisecond) antara insertion and Shell

N	insertion	Shellsort
1000	122	11
2000	483	26
4000	1936	61
8000	7950	153
16000	32560	358

Ref: Mark Allan Wiess
(Florida International University)

UNIVERSITAS Dinamika

7. RADIX SORT

UNIVERSITAS Dinamika

RADIX SORT

- Unlike the sorting algorithms described previously radix sort uses buckets to sort items, each bucket holds items with a particular property called a key.
- Normally a bucket is a queue, each time radix sort is performed these buckets are emptied starting the smallest key bucket to the largest.
- When looking at items within a list to sort we do so by isolating a special key, e.g. in the example we are about to show we have a maximum of three keys for all items, that is the highest key we need to look at is hundreds.
- Because we are dealing with, in this example base 10 numbers we have at any one point 10 possible key values 0..9 each of which has their own bucket

Universitas Dinamika

RADIX SORT

- Before we show you this first simple version of radix sort let us clarify what we mean by isolating keys.
- Given the number 102 if we look at the first key, the ones then we can see we have two of
- them, progressing to the next key - tens we can see that the number has zero of them, finally we can see that the number has a single hundred.
- The number used as an example has in total three keys:
  1. Ones
  2. Tens
  3. Hundreds

RADIX SORT

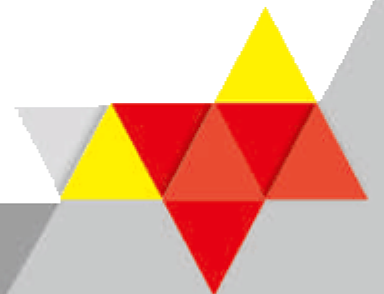
- For further clarification what if we wanted to determine how many thousands the number 102 has?
- Clearly there are none, but often looking at a number as final like we often do it is not so obvious so when asked the question how many thousands does 102 have you should simply pad the number with a zero in that location, e.g. 0102 here it is more obvious that the key value at the thousands location is zero.
- The last thing to identify before we actually show you a simple implementation of radix sort that works on only positive integers, and requires you to specify the maximum key size in the list is that we need a way to isolate a special key at any one time.
- The solution is actually very simple, but its not often you want to isolate a key in a number so we will spell it out clearly here.

RADIX SORT

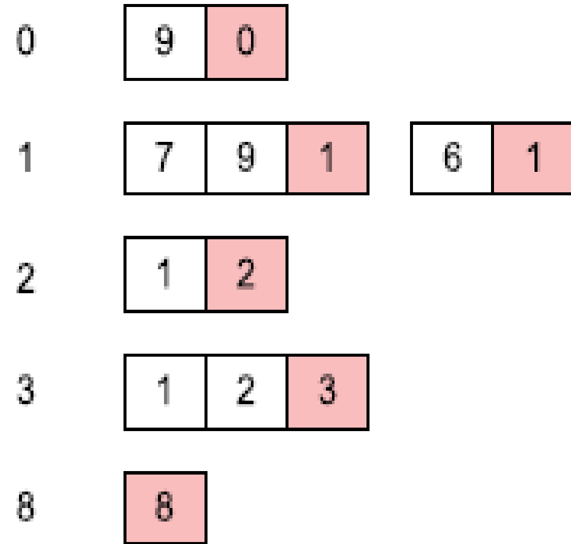
- A key can be accessed from any integer with the following expression:
  key ← (number / keyToAccess) % 10.
- As a simple example lets say that we want to access the tens key of the number 1290, the tens column is key 10 and so after substitution yields key ← (1290 = 10) % 10 = 9.
- The next key to look at for a number can be attained by multiplying the last key by ten working left to right in a sequential manner.
- The value of key is used in the following algorithm to work out the index of an array of queues to enqueue the item into.

RADIX SORT

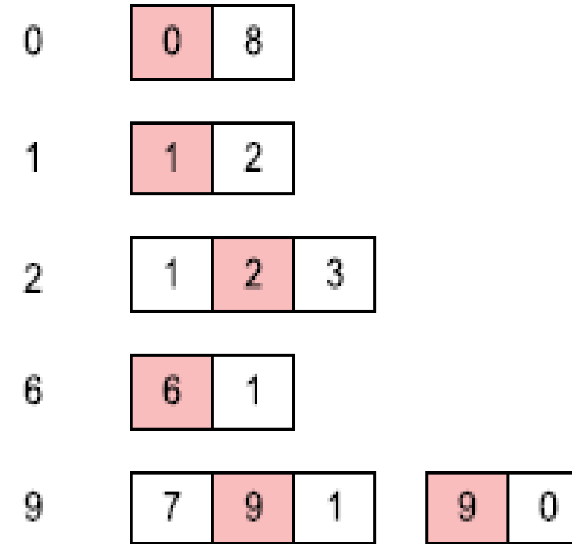
```
1) algorithm Radix(list, maxKeySize)
2)     Pre: list ≠ ∅
3)          maxKeySize ≥ 0 and represents the largest key size in the list
4)     Post: list has been sorted
5)     queues ← Queue[10]
6)     indexOfKey ← 1
7)     fori ← 0 to maxKeySize − 1
8)        foreach item in list
9)           queues[GetQueueIndex(item, indexOfKey)].Enqueue(item)
10)       end foreach
11)       list ← CollapseQueues(queues)
12)       ClearQueues(queues)
13)       indexOfKey ← indexOfKey ∗ 10
14)    end for
15)    return list
16) end Radix
```



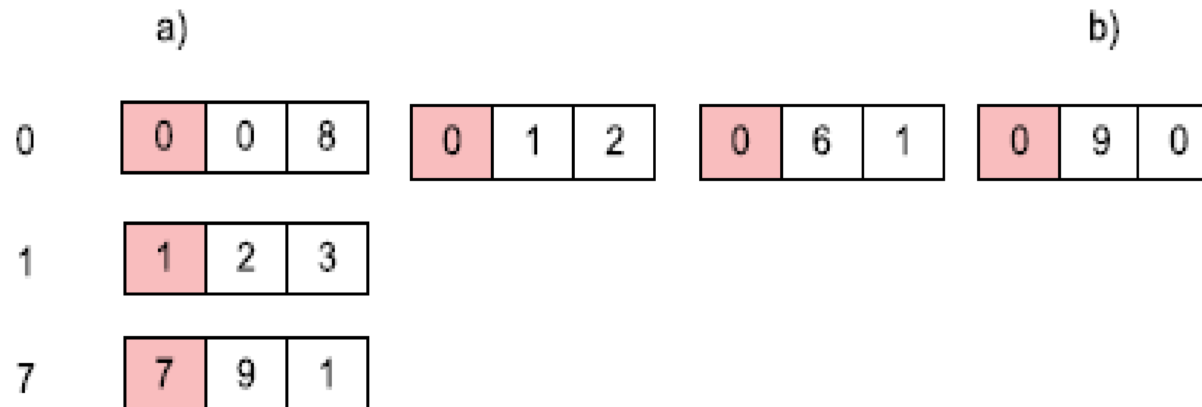
RADIX SORT

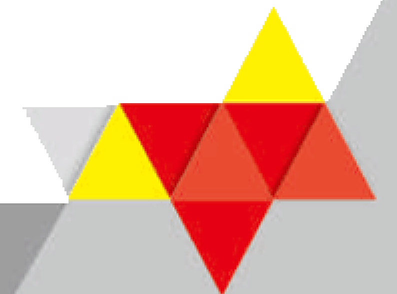


a)



b)





SELESAI