2023/2024(2)
EF234201 Data Structure

Lecture #3a

Array: Sorting

Misbakhul Munir IRFAN SUBAKTI

司馬伊凡

Мисбакхул Мунир Ирфан Субакти

Data Sorting: Why?

- Data sorting in a data structure is very important for data of the numeric or character type.
- Sorting can be done in ascending and descending order.
- Sorting is the process of rearranging data that has previously been arranged in a certain pattern so that it is arranged regularly according to certain rules.
- Example:
  - Random Data: 5 6 8 1 3 25 10
  - Ascending: 1 3 5 6 8 10 25
  - Descending: 25 10 8 6 5 3 1

Data Sorting: The Method

- Sorting based on comparison (comparison-based sorting)
  - Bubble sort, exchange sort
- Sorting based on priority (priority queue sorting method)
  - Selection sort, heap sort (using tree)
- Sorting based on insertion and keeping sorted (insert and keep sorted method)
  - Insertion sort, tree sort
- Sorting based on divide and conquer (divide and conquer method)
  - Quick sort, merge sort
- Decreasing increment sorting (diminishing increment sort method)
  - Shell sort (the development of insertion sort)

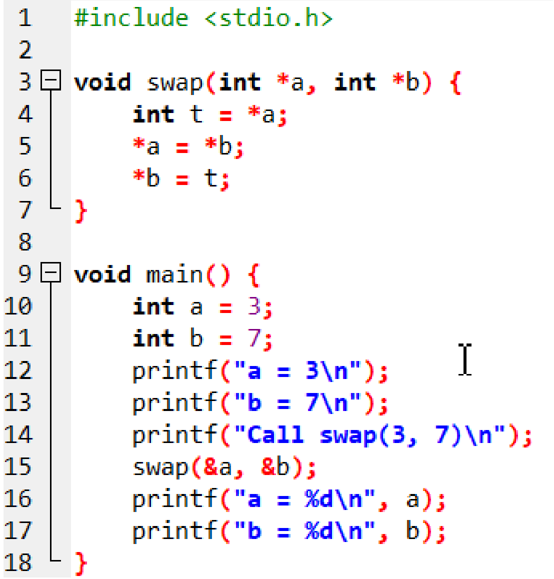
Array Declaration

- Declare:

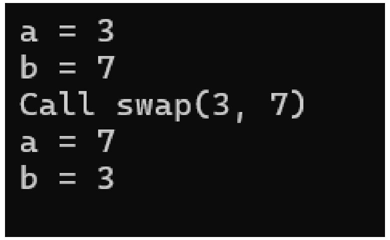
```
int data[100];
int n; // The amount of data
```

- Function to exchange the 2 items of data (by reference):

```
void swap(int *a, int *b) {
    int t = *a;
    *a = *b;
    *b = t;
}
```



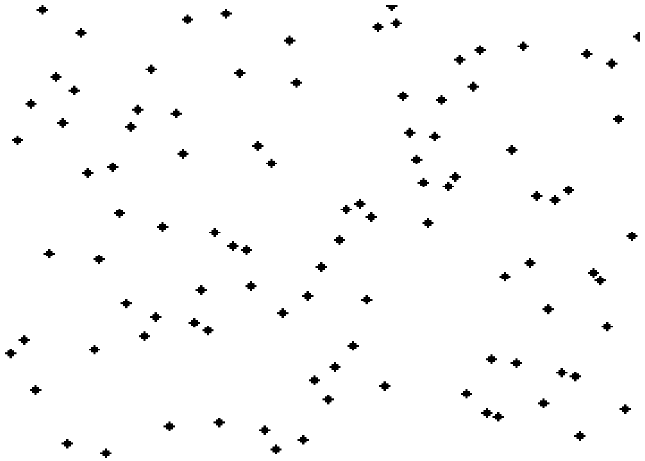
```c
#include <stdio.h>

void swap(int *a, int *b) {
    int t = *a;
    *a = *b;
    *b = t;
}

void main() {
    int a = 3;
    int b = 7;
    printf("a = 3\n");
    printf("b = 7\n");
    printf("Call swap(3, 7)\n");
    swap(&a, &b);
    printf("a = %d\n", a);
    printf("b = %d\n", b);
}
```



```
a = 3
b = 7
Call swap(3, 7)
a = 7
b = 3
```

Bubble Sort

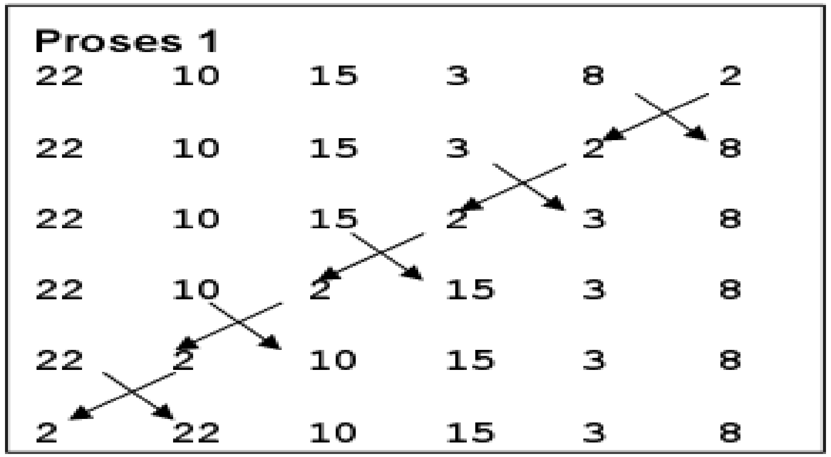
- The easiest sorting method
- It is given the name “bubble” because the sorting process gradually moves to the right position, like bubbles coming out of a fizzy glass.
- Bubble Sort sorts data by comparing the current element with the next element



Bubble Sort (continued)

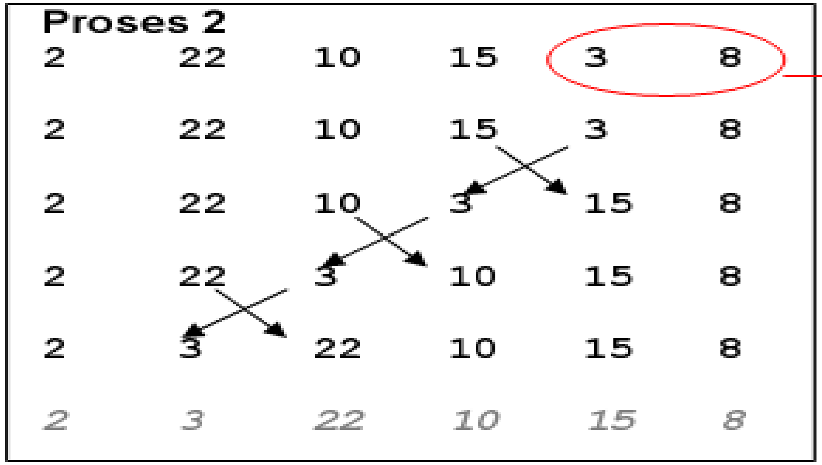
- Ascending ordering: If the current element is greater than the next element then the two elements are swapped.
- Descending ordering: If the current element is smaller than the next element, then the two elements are swapped.
- This algorithm seems to shift elements one by one from right to left or left to right, depending on the sorting type, ascending or descending.
- When one process has been completed, bubble sort will repeat the process, and so on up to *n*-1 iterations.
- When does it stop? Bubble sort stops when the entire array has been checked and no further exchanges can be made, and the desired sort is achieved.

Bubble Sort (continued)



Proses 1

22	10	15	3	8	2
22	10	15	3	2	8
22	10	15	2	3	8
22	10	2	15	3	8
22	2	10	15	3	8
2	22	10	15	3	8



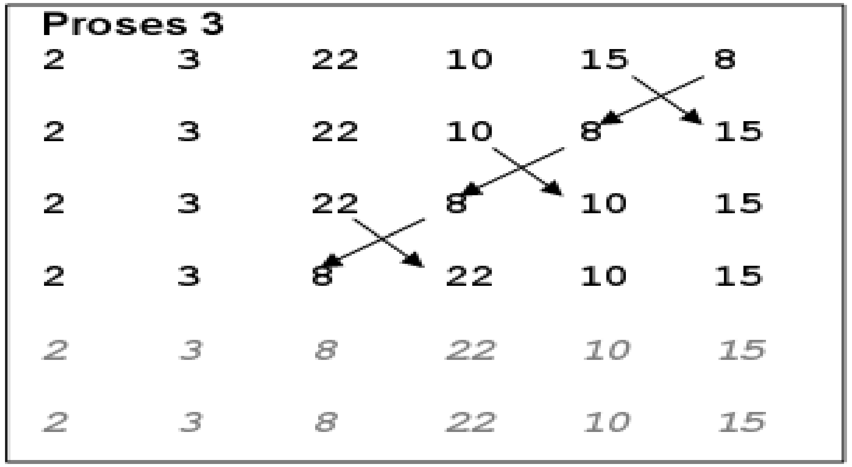
Proses 2

2	22	10	15	3	8
2	22	10	15	3	8
2	22	10	3	15	8
2	22	3	10	15	8
2	3	22	10	15	8
2	*3*	*22*	*10*	*15*	*8*

Tidak ada penukaran,
karena 3 < 8

Pegurutan berhenti di sini!

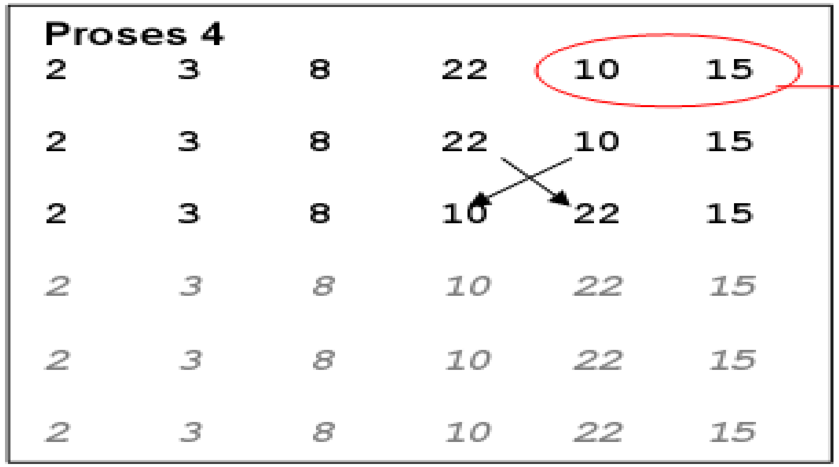
Bubble Sort (continued)



Proses 3

2	3	22	10	15	8
2	3	22	10	8	15
2	3	22	8	10	15
2	3	8	22	10	15
2	*3*	*8*	*22*	*10*	*15*
2	*3*	*8*	*22*	*10*	*15*

Pegurutan berhenti di sini!



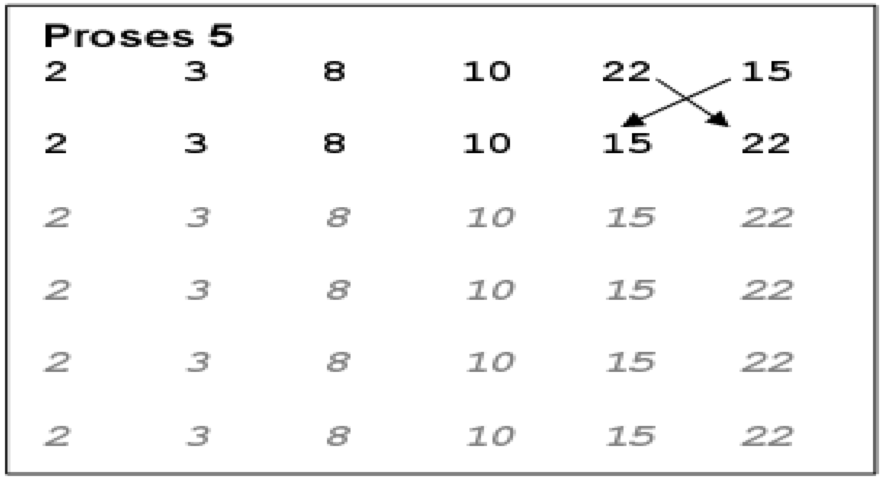
Proses 4

2	3	8	22	10	15
2	3	8	22	10	15
2	3	8	10	22	15
2	*3*	*8*	*10*	*22*	*15*
2	*3*	*8*	*10*	*22*	*15*
2	*3*	*8*	*10*	*22*	*15*

Tidak ada penukaran, karena 10 < 15

Pegurutan berhenti di sini!

Bubble Sort (continued)



Proses 5

2 3 8 10 22 15

2 3 8 10 15 22

2 3 8 10 15 22

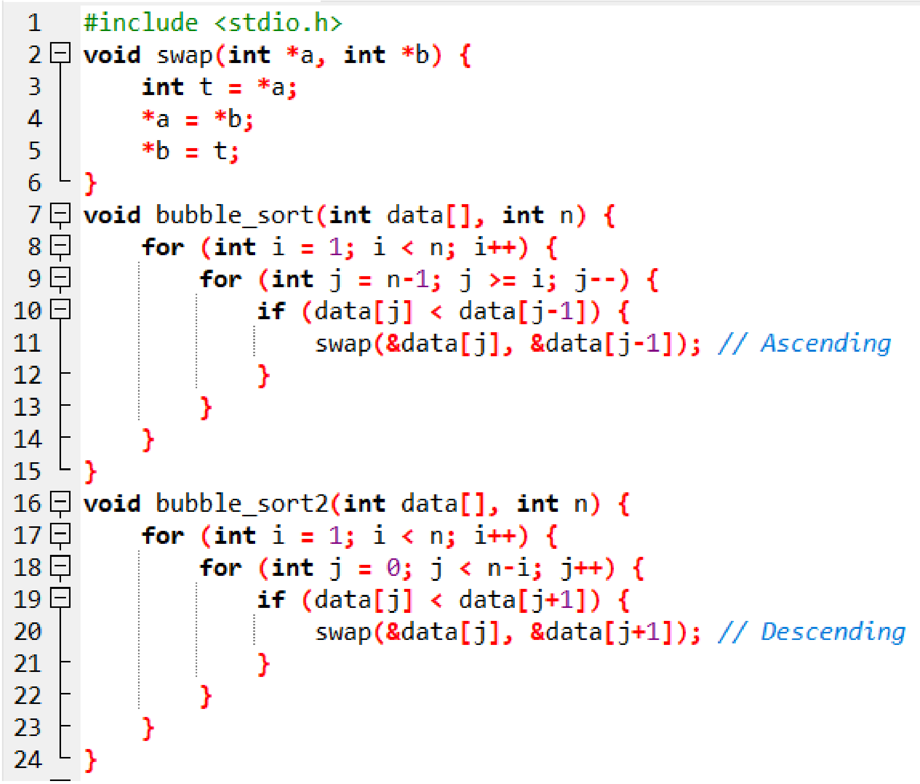
2 3 8 10 15 22

2 3 8 10 15 22

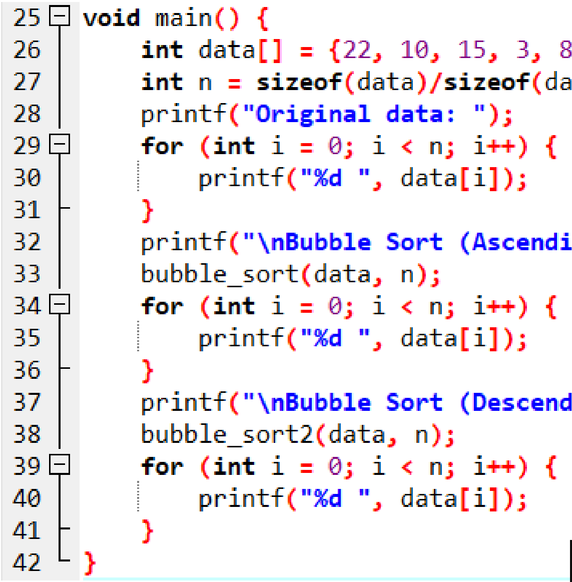
2 3 8 10 15 22

→ Pegurutan berhenti di sini!

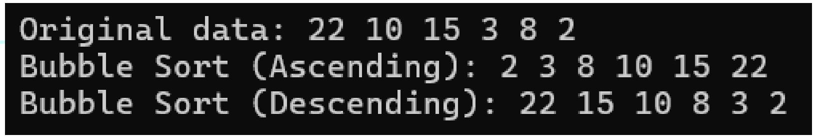
Bubble Sort (continued)



```c
#include <stdio.h>
void swap(int *a, int *b) {
    int t = *a;
    *a = *b;
    *b = t;
}
void bubble_sort(int data[], int n) {
    for (int i = 1; i < n; i++) {
        for (int j = n-1; j >= i; j--) {
            if (data[j] < data[j-1]) {
                swap(&data[j], &data[j-1]); // Ascending
            }
        }
    }
}
void bubble_sort2(int data[], int n) {
    for (int i = 1; i < n; i++) {
        for (int j = 0; j < n-i; j++) {
            if (data[j] < data[j+1]) {
                swap(&data[j], &data[j+1]); // Descending
            }
        }
    }
}
```



```c
void main() {
    int data[] = {22, 10, 15, 3, 8, 2};
    int n = sizeof(data)/sizeof(data[0]);
    printf("Original data: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", data[i]);
    }
    printf("\nBubble Sort (Ascending): ");
    bubble_sort(data, n);
    for (int i = 0; i < n; i++) {
        printf("%d ", data[i]);
    }
    printf("\nBubble Sort (Descending): ");
    bubble_sort2(data, n);
    for (int i = 0; i < n; i++) {
        printf("%d ", data[i]);
    }
}
```



```
Original data: 22 10 15 3 8 2
Bubble Sort (Ascending): 2 3 8 10 15 22
Bubble Sort (Descending): 22 15 10 8 3 2
```

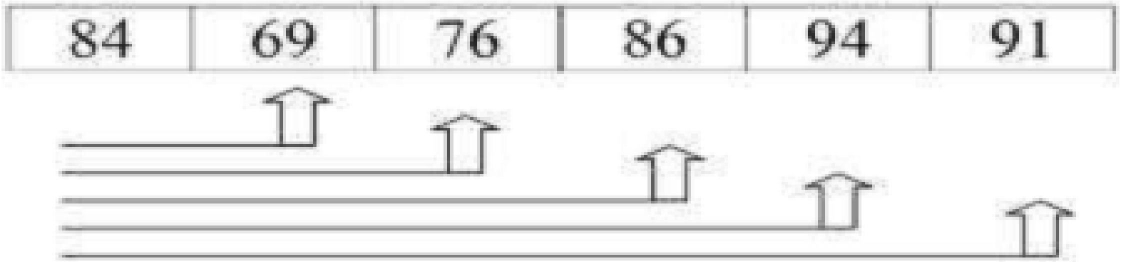
Bubble Sort (continued)

- With the procedure above, the data is sorted in *ascending* order, to sort it in *descending* order, please change the section:
  - if (data[j] < data[j-1]) { … to
  - if (data[j] > data[j-1]) {
- Likewise, the data is sorted in *descending* order, to sort it in *ascending* order, please change the section:
  - if (data[j] < data[j+1]) { { … to
  - if (data[j] > data[j+1]) { {
- The bubble sort is an easy algorithm to program, but it is slower than many other sorting methods/algorithms

Exchange Sort

- Very similar to Bubble Sort
- Many say Bubble Sort is the same as Exchange Sort
- Differentiation: in terms of how to compare the elements.
  - Exchange Sort compares *an element* with *other elements* in the array, and exchanges elements if necessary. So there is an element that is always the central element (*pivot*).
  - Meanwhile, Bubble Sort will compare the *first/last element* with the *previous/after element*, and then that element will become the center (*pivot*) to be compared with the previous/after element again, and so on.

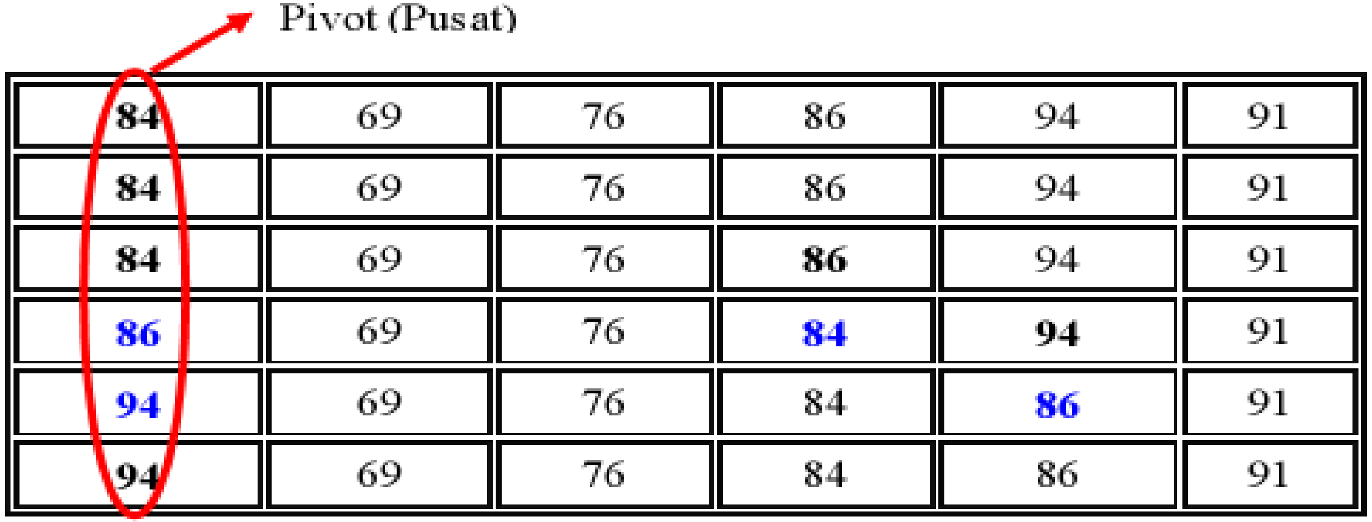
Exchange Sort (continued)



84	69	76	86	94	91

Proses 1

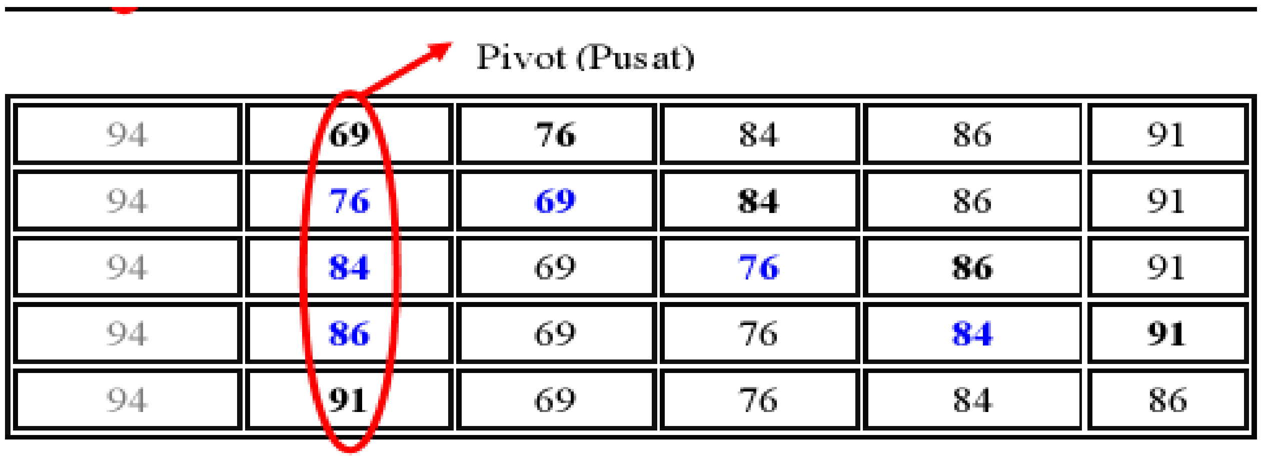
Pivot (Pusat)



84	69	76	86	94	91
84	69	76	86	94	91
84	69	76	**86**	94	91
86	69	76	**84**	**94**	91
94	69	76	84	**86**	91
94	69	76	84	86	91

Exchange Sort (continued)

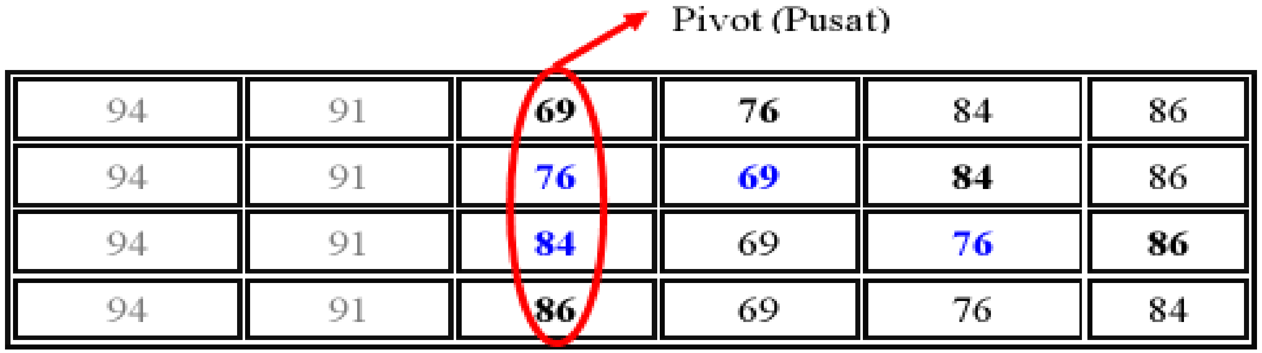
Proses 2



Pivot (Pusat)

94	**69**	**76**	84	86	91
94	**76**	**69**	**84**	86	91
94	**84**	69	**76**	**86**	91
94	**86**	69	76	**84**	**91**
94	**91**	69	76	84	86

Proses 3

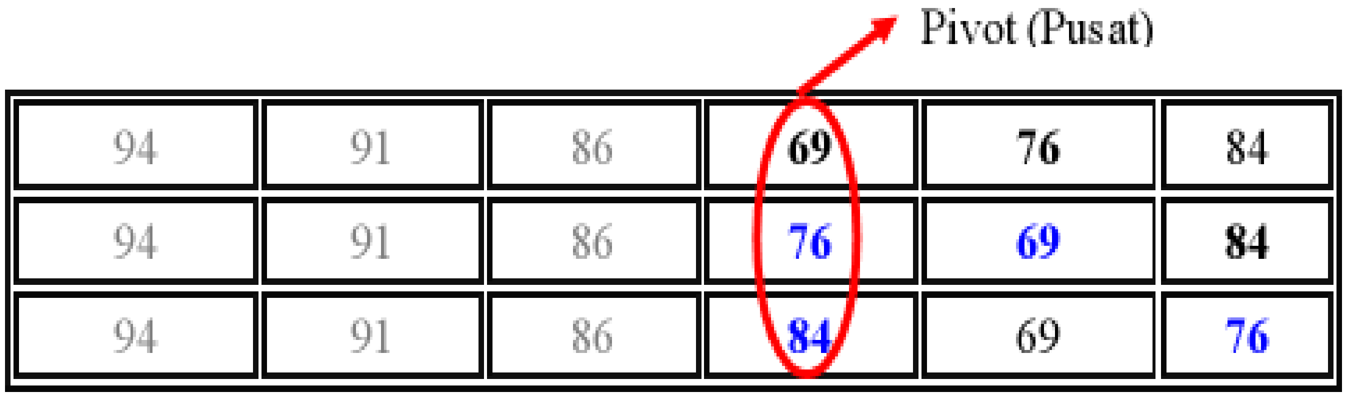


Pivot (Pusat)

94	91	**69**	**76**	84	86
94	91	**76**	**69**	**84**	86
94	91	**84**	69	**76**	**86**
94	91	**86**	69	76	84

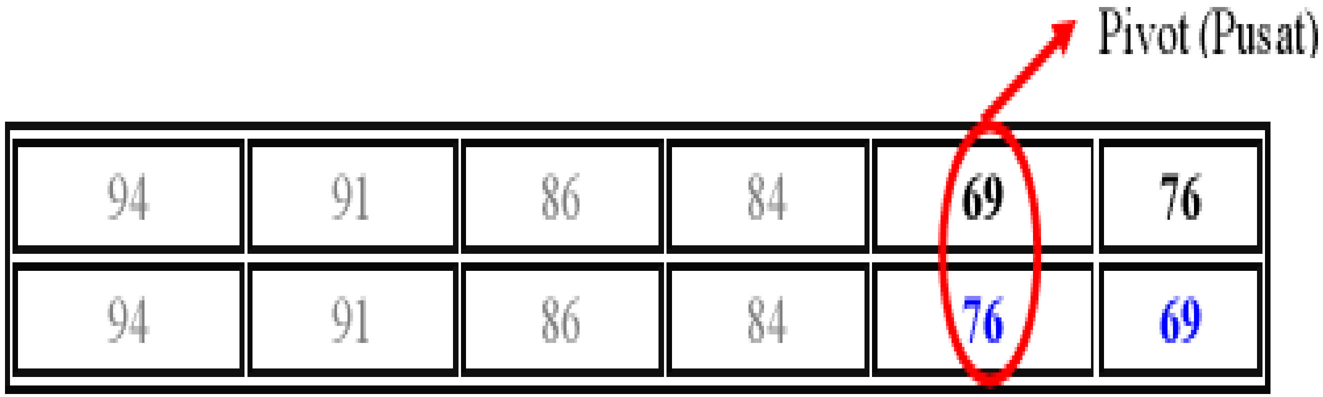
Exchange Sort (continued)

Proses 4


Pivot (Pusat)

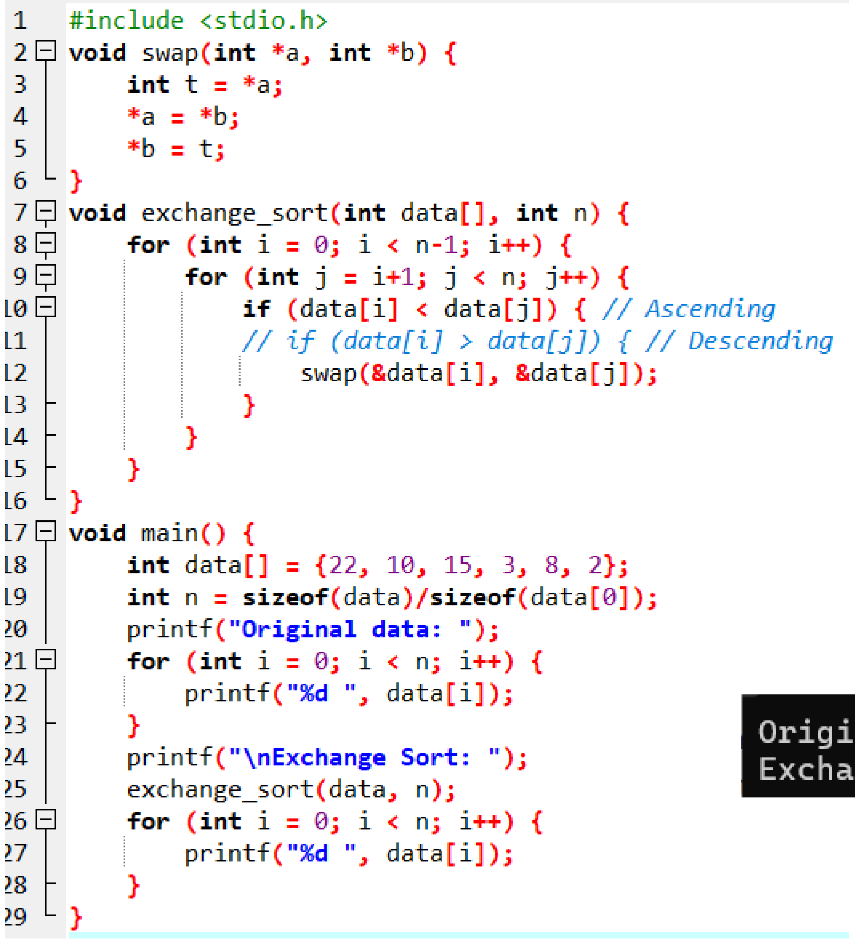
94	91	86	69	76	84
94	91	86	76	69	84
94	91	86	84	69	76

Proses 5

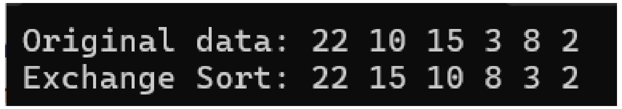

Pivot (Pusat)

94	91	86	84	69	76
94	91	86	84	76	69

Exchange Sort (continued)



```c
#include <stdio.h>
void swap(int *a, int *b) {
    int t = *a;
    *a = *b;
    *b = t;
}
void exchange_sort(int data[], int n) {
    for (int i = 0; i < n-1; i++) {
        for (int j = i+1; j < n; j++) {
            if (data[i] < data[j]) { // Ascending
            // if (data[i] > data[j]) { // Descending
                swap(&data[i], &data[j]);
            }
        }
    }
}
void main() {
    int data[] = {22, 10, 15, 3, 8, 2};
    int n = sizeof(data)/sizeof(data[0]);
    printf("Original data: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", data[i]);
    }
    printf("\nExchange Sort: ");
    exchange_sort(data, n);
    for (int i = 0; i < n; i++) {
        printf("%d ", data[i]);
    }
}
```

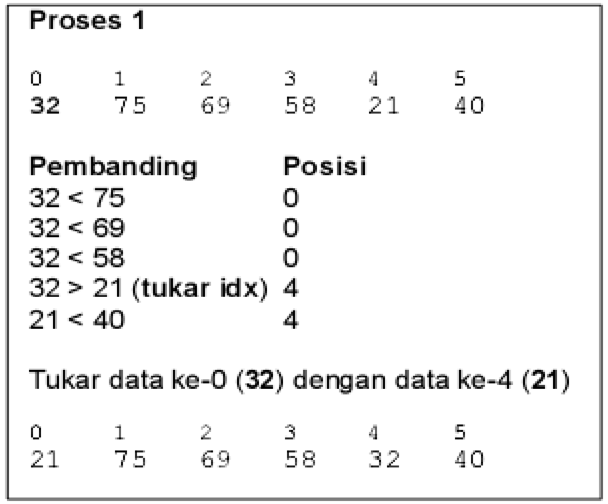


```
Original data: 22 10 15 3 8 2
Exchange Sort: 22 15 10 8 3 2
```

Selection Sort

- It is a combination of sorting and searching
- For each process, it will look for unsorted elements that have the smallest or largest value and will be swapped to the right position in the array.
- For example, for the first round, the data with the smallest value will be searched and this data will be placed in the smallest index (data[0]), in the second round the second smallest data will be searched for, and it will be placed in the second index (data[1]).
- During the process, comparisons and changes are *only made* to the comparison *index*, physical data exchange occurs at the *end* of the process.

Selection Sort (continued)



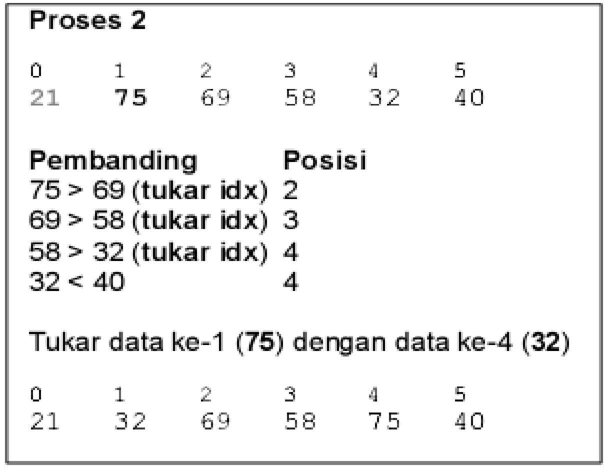
Proses 1

0	1	2	3	4	5
32	75	69	58	21	40

Pembanding	**Posisi**
32 < 75	0
32 < 69	0
32 < 58	0
32 > 21 (**tukar idx**)	4
21 < 40	4

Tukar data ke-0 (**32**) dengan data ke-4 (**21**)

0	1	2	3	4	5
21	75	69	58	32	40



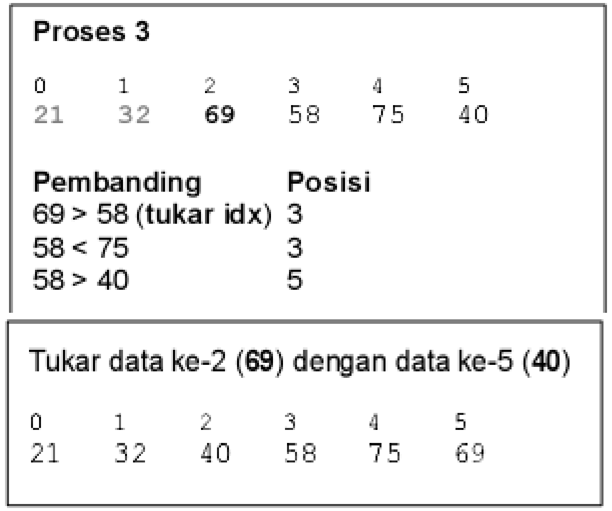
Proses 2

0	1	2	3	4	5
21	**75**	69	58	32	40

Pembanding	**Posisi**
75 > 69 (**tukar idx**)	2
69 > 58 (**tukar idx**)	3
58 > 32 (**tukar idx**)	4
32 < 40	4

Tukar data ke-1 (**75**) dengan data ke-4 (**32**)

0	1	2	3	4	5
21	32	69	58	75	40



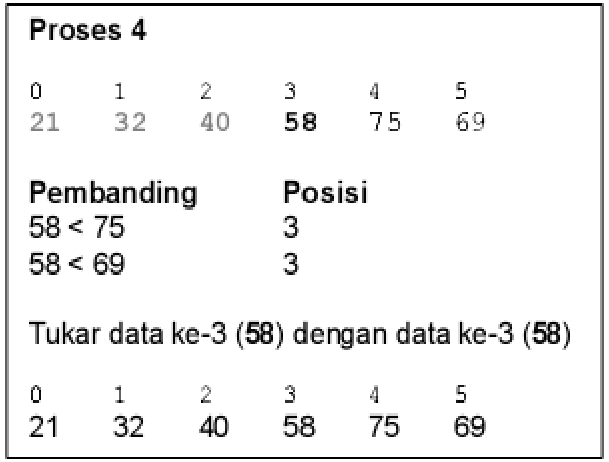
Proses 3

0	1	2	3	4	5
21	32	**69**	58	75	40

Pembanding	**Posisi**
69 > 58 (**tukar idx**)	3
58 < 75	3
58 > 40	5

Tukar data ke-2 (**69**) dengan data ke-5 (**40**)

0	1	2	3	4	5
21	32	40	58	75	69



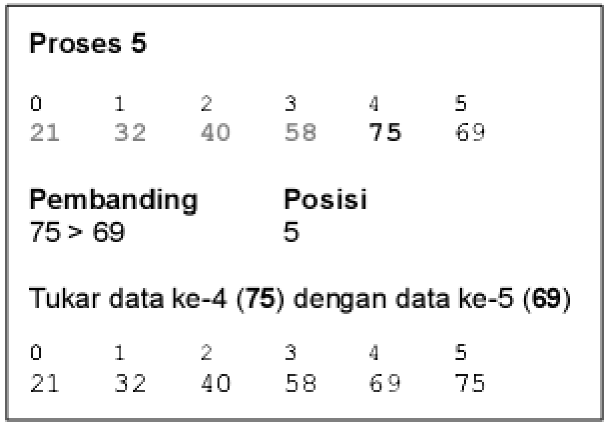
Proses 4

0	1	2	3	4	5
21	32	40	**58**	75	69

Pembanding	**Posisi**
58 < 75	3
58 < 69	3

Tukar data ke-3 (**58**) dengan data ke-3 (**58**)

0	1	2	3	4	5
21	32	40	58	75	69



Proses 5

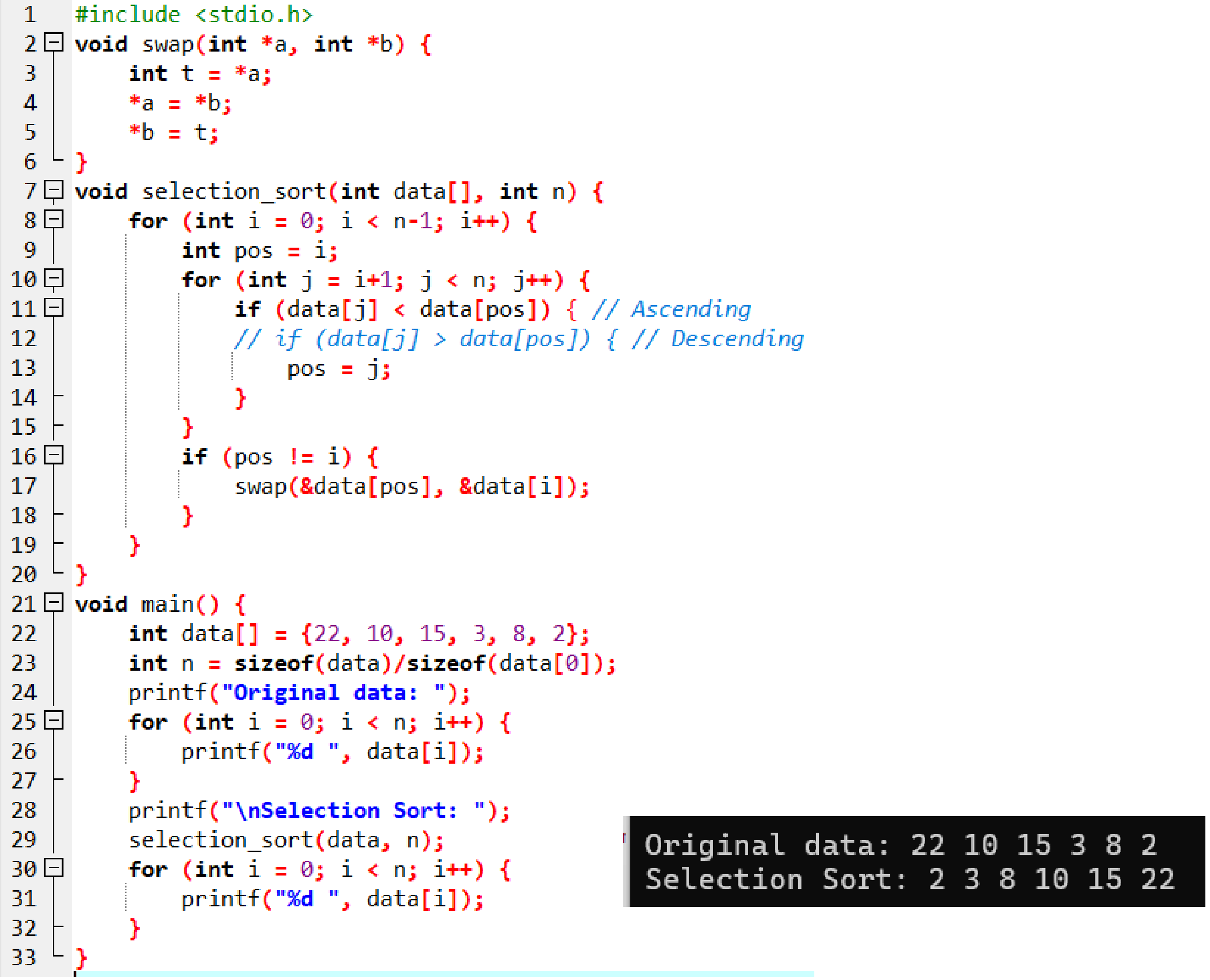
0	1	2	3	4	5
21	32	40	58	**75**	69

Pembanding	**Posisi**
75 > 69	5

Tukar data ke-4 (**75**) dengan data ke-5 (**69**)

0	1	2	3	4	5
21	32	40	58	69	75

Selection Sort (continued)

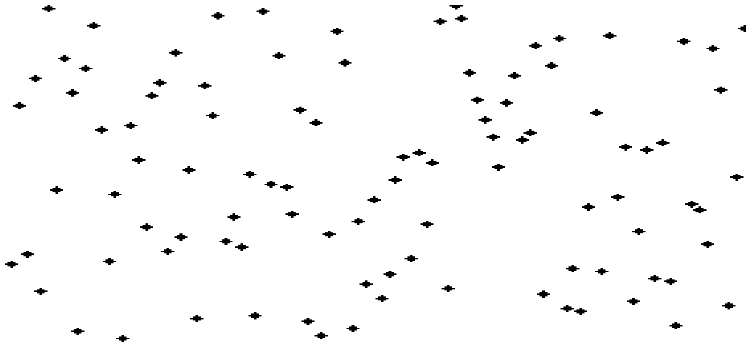


```c
#include <stdio.h>
void swap(int *a, int *b) {
    int t = *a;
    *a = *b;
    *b = t;
}
void selection_sort(int data[], int n) {
    for (int i = 0; i < n-1; i++) {
        int pos = i;
        for (int j = i+1; j < n; j++) {
            if (data[j] < data[pos]) { // Ascending
            // if (data[j] > data[pos]) { // Descending
                pos = j;
            }
        }
        if (pos != i) {
            swap(&data[pos], &data[i]);
        }
    }
}
void main() {
    int data[] = {22, 10, 15, 3, 8, 2};
    int n = sizeof(data)/sizeof(data[0]);
    printf("Original data: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", data[i]);
    }
    printf("\nSelection Sort: ");
    selection_sort(data, n);
    for (int i = 0; i < n; i++) {
        printf("%d ", data[i]);
    }
}
```

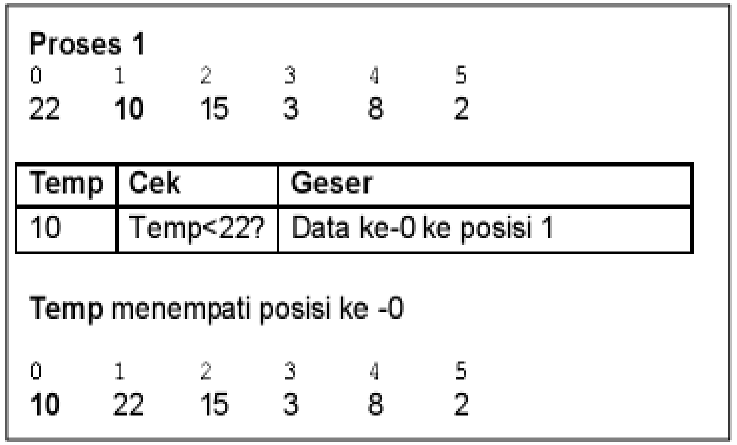
```
Original data: 22 10 15 3 8 2
Selection Sort: 2 3 8 10 15 22
```

Insertion Sort

- Similar to the way people *sort cards*, one by one the cards are taken out and inserted into their proper places.
- Sorting starts from the 2nd data to the last data, if *smaller* data is found, it will be placed (*inserted*) in the correct position.
- When inserting an element, the other elements will shift to the back



Insertion Sort (continued)



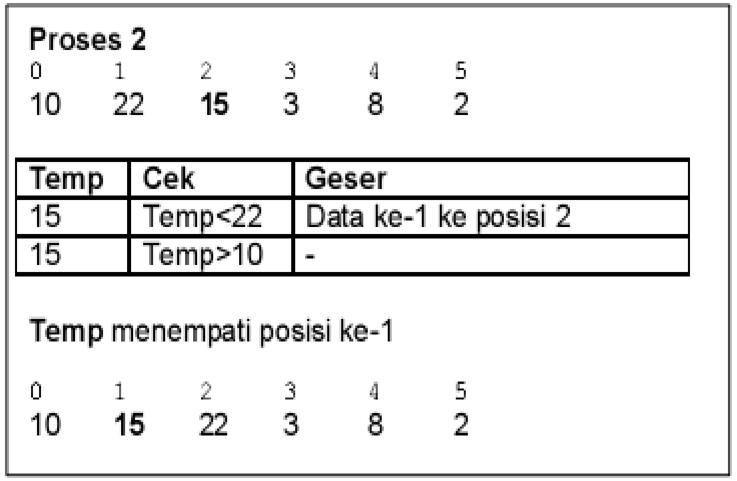
Proses 1

0	1	2	3	4	5
22	**10**	15	3	8	2

Temp	Cek	Geser
10	Temp<22?	Data ke-0 ke posisi 1

Temp menempati posisi ke -0

0	1	2	3	4	5
10	22	15	3	8	2



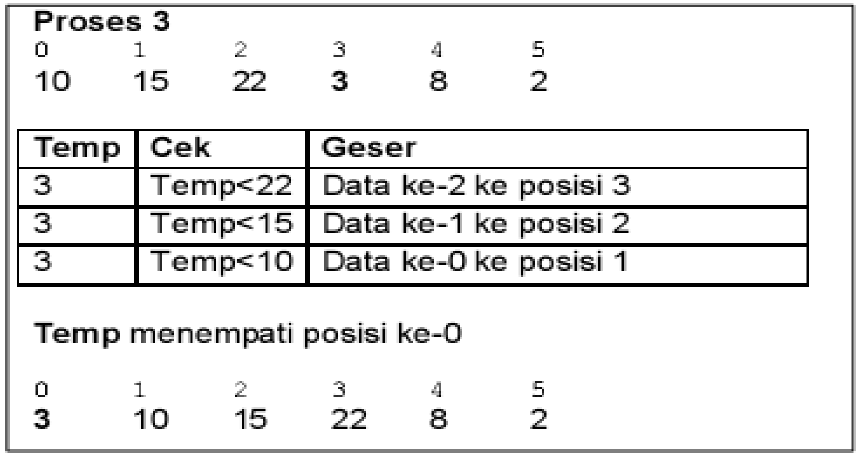
Proses 2

0	1	2	3	4	5
10	22	**15**	3	8	2

Temp	Cek	Geser
15	Temp<22	Data ke-1 ke posisi 2
15	Temp>10	-

Temp menempati posisi ke-1

0	1	2	3	4	5
10	**15**	22	3	8	2



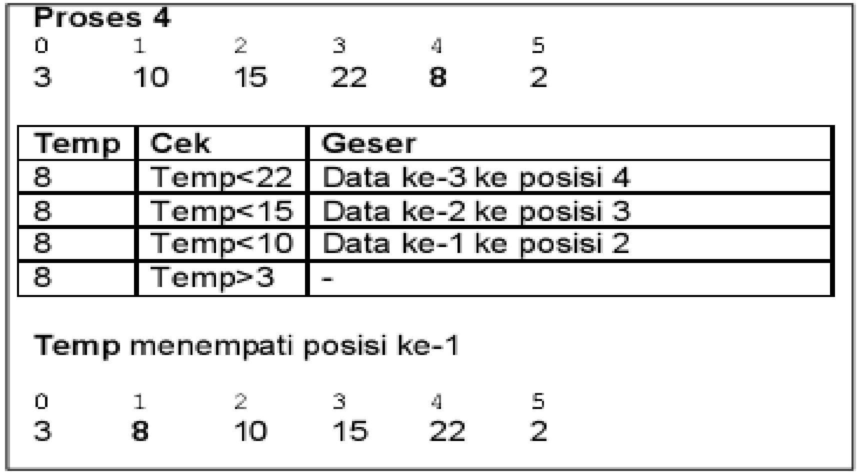
Proses 3

0	1	2	3	4	5
10	15	22	**3**	8	2

Temp	Cek	Geser
3	Temp<22	Data ke-2 ke posisi 3
3	Temp<15	Data ke-1 ke posisi 2
3	Temp<10	Data ke-0 ke posisi 1

Temp menempati posisi ke-0

0	1	2	3	4	5
3	10	15	22	8	2



Proses 4

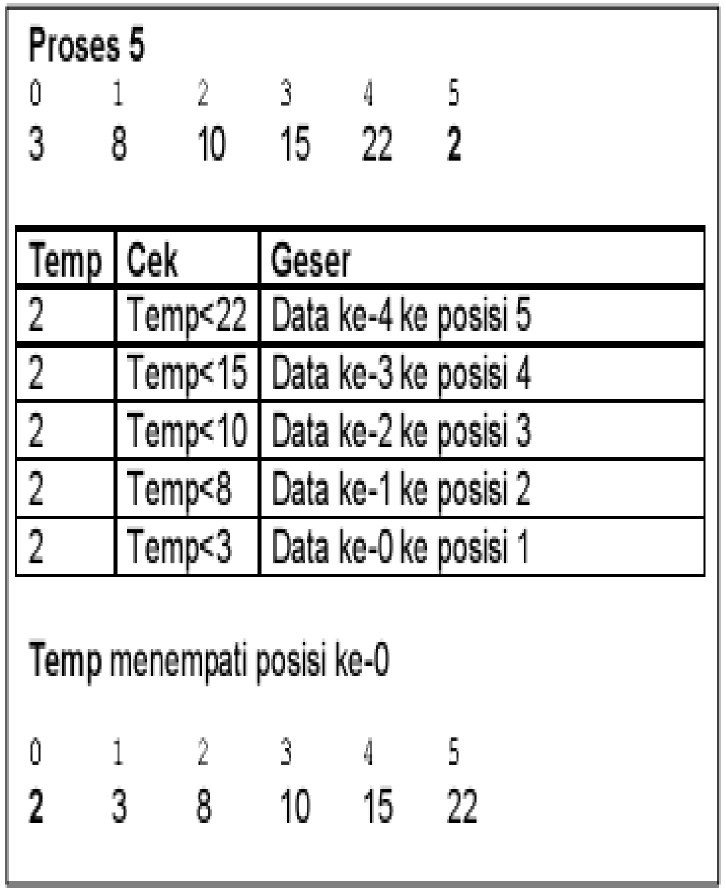
0	1	2	3	4	5
3	10	15	22	**8**	2

Temp	Cek	Geser
8	Temp<22	Data ke-3 ke posisi 4
8	Temp<15	Data ke-2 ke posisi 3
8	Temp<10	Data ke-1 ke posisi 2
8	Temp>3	-

Temp menempati posisi ke-1

0	1	2	3	4	5
3	**8**	10	15	22	2

Insertion Sort (continued)



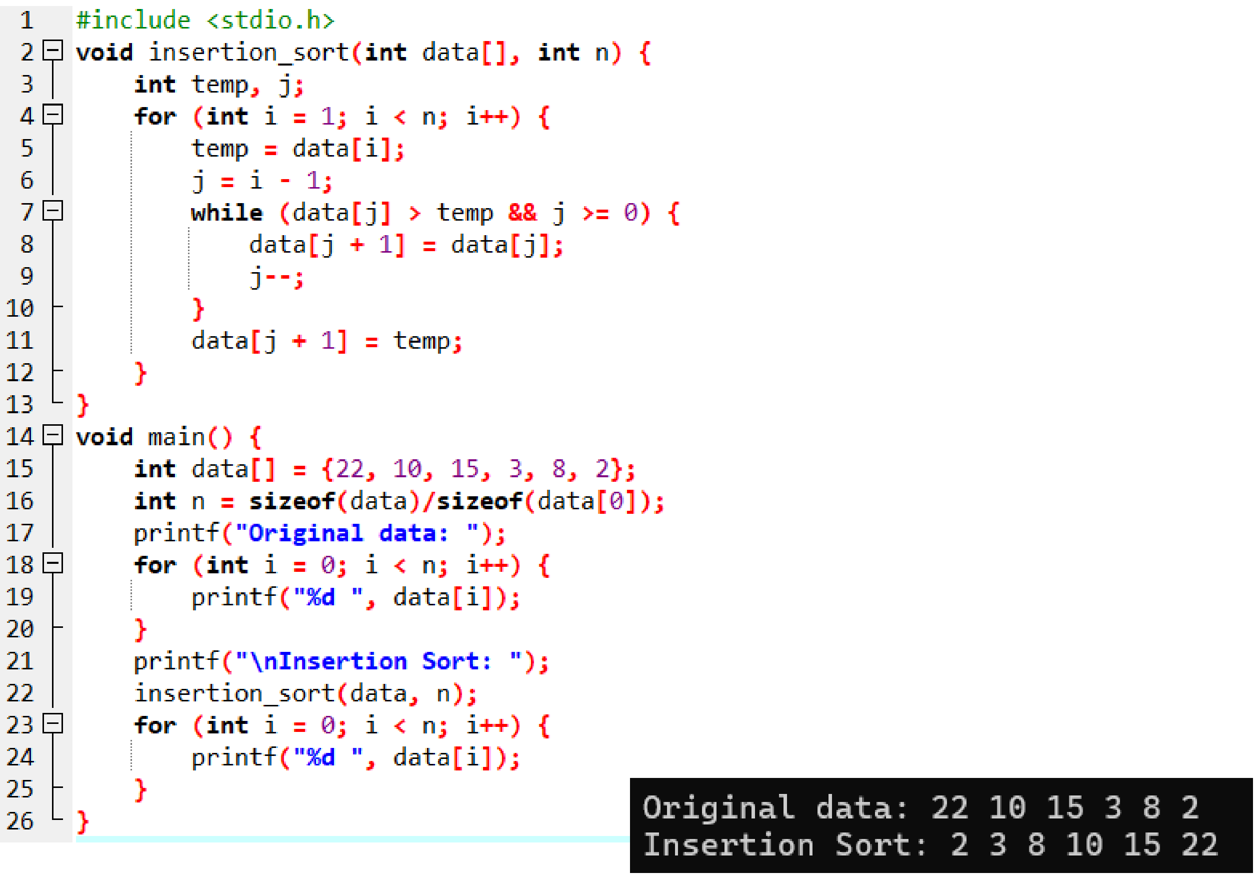
Proses 5

0	1	2	3	4	5
3	8	10	15	22	2

Temp	Cek	Geser
2	Temp<22	Data ke-4 ke posisi 5
2	Temp<15	Data ke-3 ke posisi 4
2	Temp<10	Data ke-2 ke posisi 3
2	Temp<8	Data ke-1 ke posisi 2
2	Temp<3	Data ke-0 ke posisi 1

Temp menempati posisi ke-0

0	1	2	3	4	5
2	3	8	10	15	22



```c
#include <stdio.h>
void insertion_sort(int data[], int n) {
    int temp, j;
    for (int i = 1; i < n; i++) {
        temp = data[i];
        j = i - 1;
        while (data[j] > temp && j >= 0) {
            data[j + 1] = data[j];
            j--;
        }
        data[j + 1] = temp;
    }
}
void main() {
    int data[] = {22, 10, 15, 3, 8, 2};
    int n = sizeof(data)/sizeof(data[0]);
    printf("Original data: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", data[i]);
    }
    printf("\nInsertion Sort: ");
    insertion_sort(data, n);
    for (int i = 0; i < n; i++) {
        printf("%d ", data[i]);
    }
}
```

```
Original data: 22 10 15 3 8 2
Insertion Sort: 2 3 8 10 15 22
```

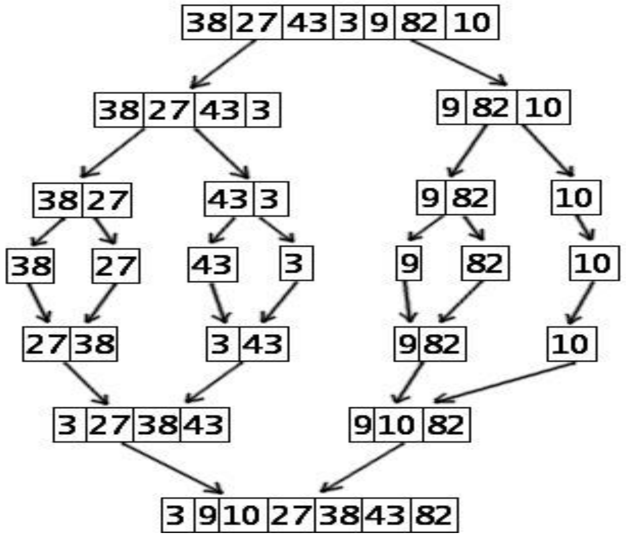
Comparison

- Speed Comparison Table of Data Sorting Methods
- For 10,000 data on a Pentium II 450 MHz computer

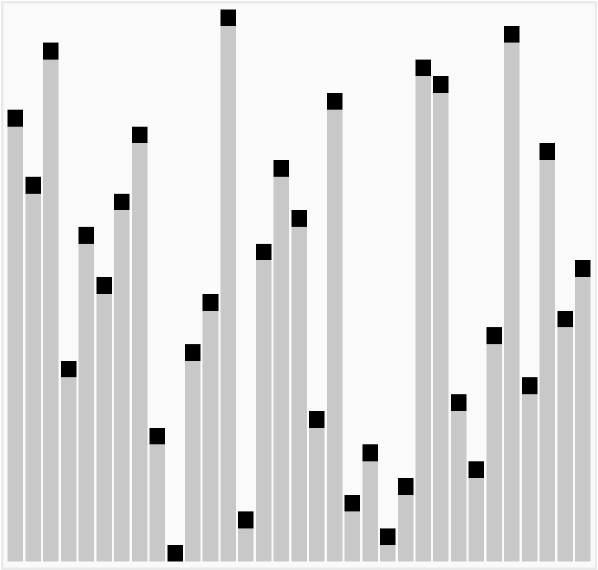
Method	Time (second)		
	Random data	Ascending data	Descending data
Bubble Sort	11.2	1.32	19.77
Insertion Sort	1.09	0.00	2.25
Selection Sort	1.32	1.32	19.77

More methods

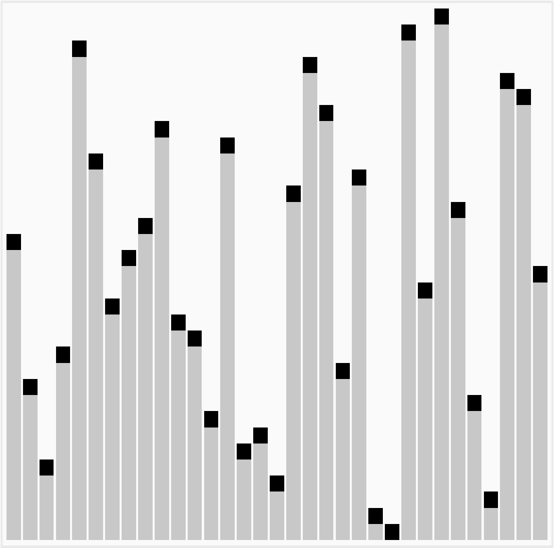
- Merge Sort



Heap Sort



Quick Sort



Exercise

- Look for 3 other sorting methods and write them down in the paper along with source code, methods and analysis and every sorting method that exists!
- Make all the above procedures into the complete programs!