

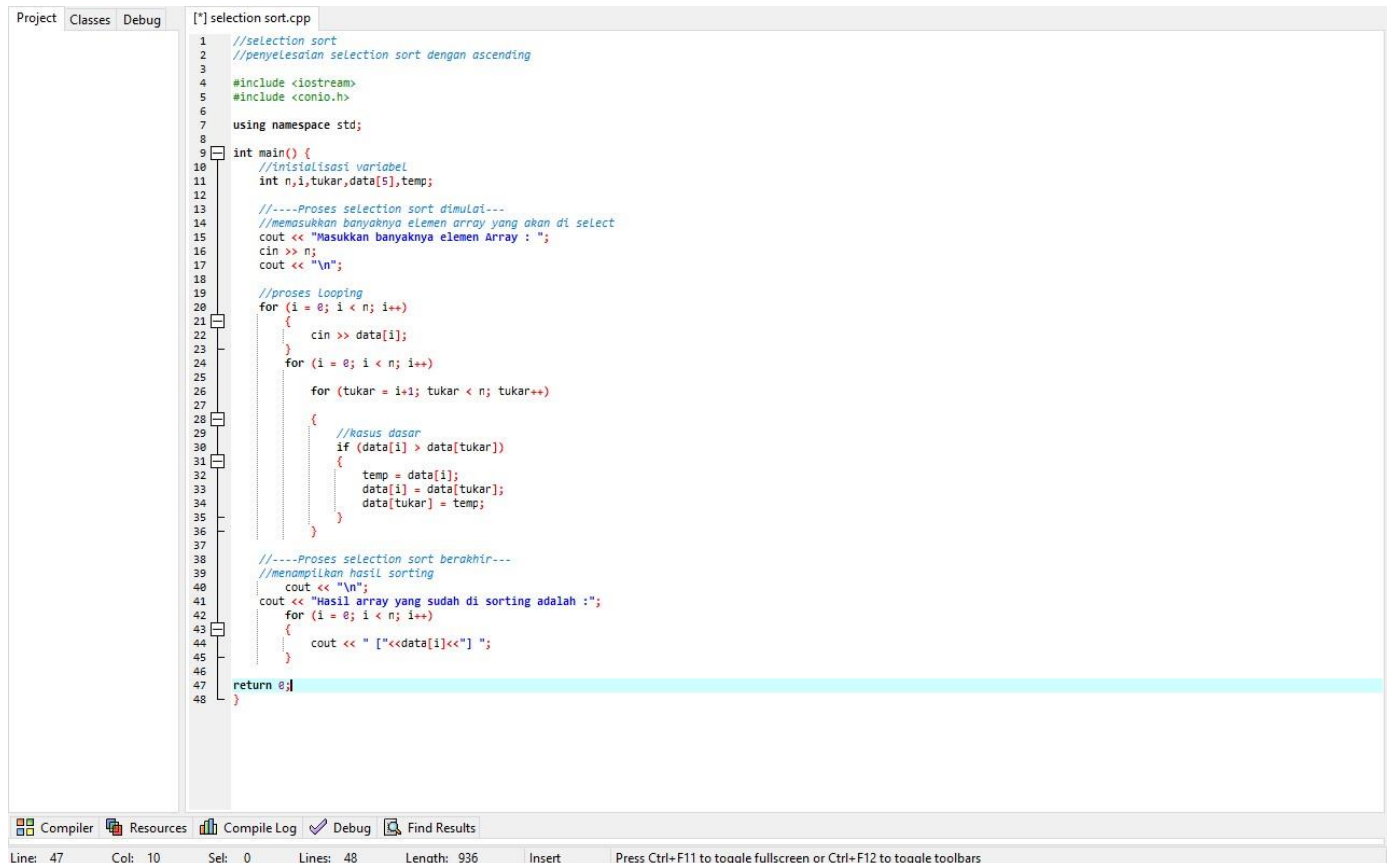
Nama : Ummiyatun

NIM : 21091397039

## LAPORAN INDIVIDU

### //SELECTION SORT

#### 1. Screenshot kode sorting



```
1 //selection sort
2 //penyelesaian selection sort dengan ascending
3
4 #include <iostream>
5 #include <conio.h>
6
7 using namespace std;
8
9 int main() {
10     //inisialisasi variabel
11     int n,i,tukar,data[5],temp;
12
13     //----Proses selection sort dimulai---
14     //memasukkan banyaknya elemen array yang akan di select
15     cout << "Masukkan banyaknya elemen Array : ";
16     cin >> n;
17     cout << "\n";
18
19     //proses looping
20     for (i = 0; i < n; i++)
21     {
22         cin >> data[i];
23         for (t = i + 1; t < n; t++)
24         {
25             if (data[i] > data[t])
26             {
27                 //khusus dasar
28                 temp = data[i];
29                 data[i] = data[t];
30                 data[t] = temp;
31             }
32         }
33     }
34
35     //----Proses selection sort berakhir---
36     //menampilkan hasil sorting
37     cout << "\n";
38     cout << "Hasil array yang sudah di sorting adalah : ";
39     for (i = 0; i < n; i++)
40     {
41         cout << " ["<<data[i]<<"] ";
42     }
43
44     return 0;
45 }
```

Gambar di atas merupakan *source code selection sort* dengan proses *ascending*, yaitu proses pengurutan data dari nilai terkecil ke nilai terbesar. Dalam kode tersebut, batas data yang bisa diinput adalah 5 yang ditandai dengan inisialisasi variabel `data[5]`.

Algoritma ini mencari nilai terkecil dari semua elemen array. Setelah berhasil menemukan nilai terkecil, maka *selection sort* ini akan membandingkan dan menukarkan nilai terkecil tersebut ke posisi pertama dalam list sehingga nilai terkecil tersebut sudah berada di posisi yang seharusnya. Nilai yang berada di posisi pertama tersebut akan dipindahkan ke posisi di mana nilai terkecil tadi berada. Proses ini akan terus berulang hingga semua elemen array berhasil terurut dengan benar. Ketika melakukan proses pengulangan, nilai di posisi pertama tadi tidak perlu dilibatkan kembali karena sudah berada di posisi yang seharusnya. Jadi proses pengulangan berikutnya hanya dilakukan pada sisa elemen yang belum terurut.

## 2. Screenshot hasil run kodingan

```
C:\Users\Dell\Documents\smt2 unesa\Prak struktur data\selection sort.exe
Masukkan banyaknya elemen Array : 5
4
20
3
9
13

Hasil array yang sudah di sorting adalah : [3] [4] [9] [13] [20]
-----
Process exited after 27.99 seconds with return value 0
Press any key to continue . . .
```

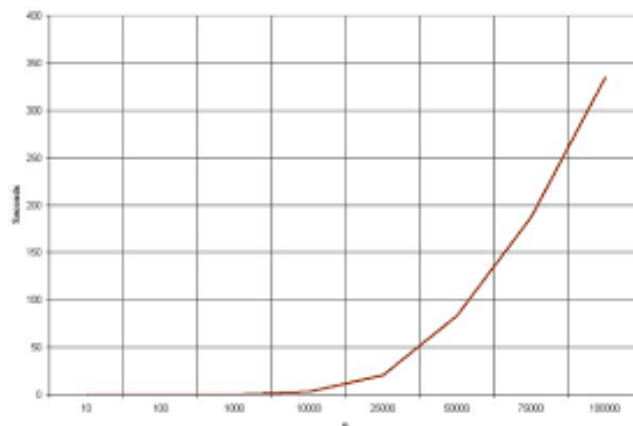
Terdapat 5 elemen array yang belum terurut yaitu: 4, 20, 3, 9, 13. Dengan menerapkan algoritma *selection sort* yaitu proses *ascending*, maka elemen-elemen tersebut akan diatur ulang sehingga elemen array tersebut berhasil terurut dengan benar. Sesuai gambar di atas, maka hasil elemen yang sudah disorting dengan benar yaitu: [3] [4] [9] [13] [20].

## 3. Big O selection sort

Operasi perbandingan elemennya :

$$T(n) = (n - 1) + (n - 2) + \dots + 2 + 1 = \sum_{i=1}^{n-1} n - i$$
$$= \frac{n(n-1)}{2} = O(n^2)$$

Kompleksitas waktu selection sort adalah  $O(n^2)$ . Karena terdapat proses perulangan bersarang (*nested loop*).



Gambar 1. Grafik kompleksitas selection sort (sumber: [informatika.stei.itb.ac.id /MakalahStrukdis0910-074.pdf](http://informatika.stei.itb.ac.id/MakalahStrukdis0910-074.pdf))

- Big O saat  $n=1$   
→  $O(n^2) = O(1^2) = 1$
- Big O saat  $n=5$   
→  $O(n^2) = O(5^2) = 25$
- Big O saat  $n=10$   
→  $O(n^2) = O(10^2) = 100$

#### **4. Kelebihan dan kekurangan *selection sort***

- Kelebihan *selection sort*
  - a) Algoritma ini sangat mudah untuk diimplementasikan
  - b) Mudah dalam menentukan data maksimum / minimum
  - c) Kompleksitasnya relatif lebih kecil
  - d) Waktu pengurutannya singkat.
- Kekurangan *selection sort*
  - a) Perlu dihindari untuk penggunaan data dalam jumlah besar, karena akan menyebabkan kompleksitas yang lebih tinggi dan kurang praktis
  - b) Membutuhkan metode tambahan.