Nama : Alvin Febrianto
NIM : 21091397031
Kelas : 2021 A

Laporan Individu Bubble Sort

1. Program C++ Bubble Sort

A. Fungsi Void Bubble Sort

```
// Implementasi Bubble Sort pada Program C++
#include <iostream>
#include <conio.h>
using namespace std;

// Fungsi untuk mengimplementasikan bubble sort
void Bubble_Sort(int array[], int n) {
      /* Deklarasi variabel i dan j untuk perulangan serta variabel
         temporary untuk penukaran sementara */
      int i, j, temporary;
      // Looping untuk mengakses setiap elemen array
      for (i = 0; i < n; i++) {
            for (j = 0; j < n - i - 1; j++) {
                  // Membandingkan 2 elemen array yang berdekatan
                  if (array[j] > array[j + 1]) {
                        // Menukar elemen jika belum sesuai urutan
                        temporary = array[j];
                        array[j] = array[j + 1];
                        array[j + 1] = temporary;
                  }
            }
      }
}
```

B. Fungsi Main

```
// Fungsi utama program
int main() {
	// Deklarasi variabel
	int array[100], n, i;

	// Input nilai untuk variabel n
	cout << "Masukkan batas jumlah array: ";
	cin >> n;
	cout << endl;

	// Input nilai untuk variabel array
	cout << "Masukkan " << n << " elemen array: \n";
	for (i = 0; i < n; i++) {
		cin >> array[i];
	}

	/* Memanggil fungsi Bubble_Sort dengan memasukkan variabel array
	   dan n */
	Bubble_Sort(array, n);
	// Menampilkan hasil pengurutan
	cout << "\nHasil array yang sudah disorting adalah: \n";
	for (i = 0; i < n; i++) {
		cout << "[" << array[i] << "]";
	}

	getch();
}
```

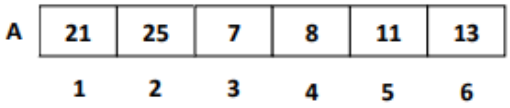
C. Output Program

```
Masukkan jumlah angka dalam array: 5

Masukkan 5 elemen array:
4 20 3 9 13

Hasil array yang sudah disorting adalah:
[3] [4] [9] [13] [20]
```

2. Bubble Sort

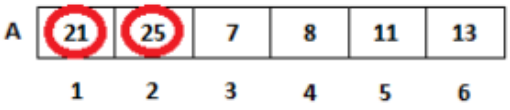
A. Konsep Bubble Sort

Konsep pengurutan ini dilakukan dengan cara membandingkan setiap elemen dengan elemen setelahnya. Untuk membentuk urutan menaik atau menurun, elemen dipertukarkan sesuai aturan. Pembandingan seluruh elemen dilakukan sebanyak (N-1) kali. Contoh: Misalnya kita akan mengurutkan sebuah array A yang memiliki 6 elemen. Dalam urutan menaik, maka elemen pertama harus lebih kecil dari elemen terakhir.



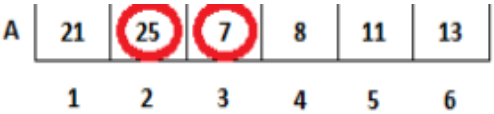
1) Langkah 1

- Dimulai dengan mengakses indeks pertama dari array dan membandingkannya dengan indeks setelahnya (indeks kedua).

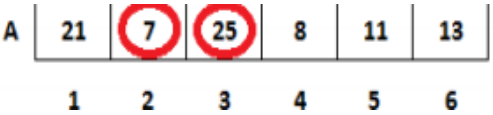


Karena elemen pertama tidak lebih besar dari elemen kedua, maka tidak dilakukan pertukaran.

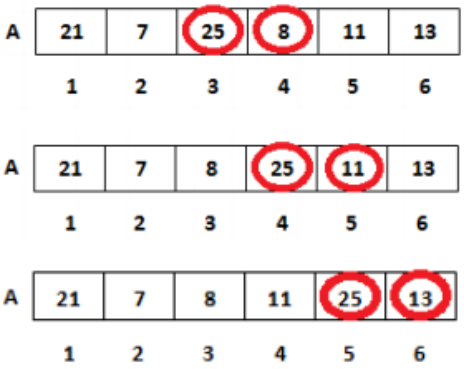
- Kemudian dimulai kembali dengan membandingkan indeks kedua dengan indeks setelahnya (indeks ketiga).



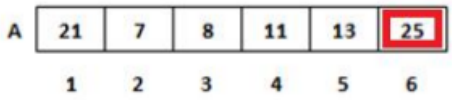
Karena elemen kedua lebih besar dari elemen ketiga, maka dilakukan pertukaran.



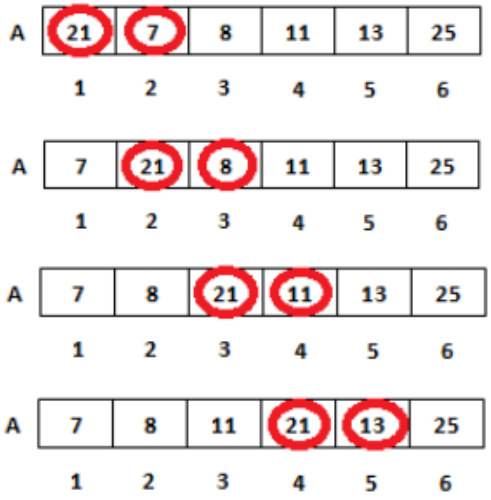
- Selanjutnya, dilakukan dengan aturan yang sama. Apabila nilai di suatu indeks lebih besar dari nilai di indeks setelahnya, maka pertukaran nilai akan dilakukan.



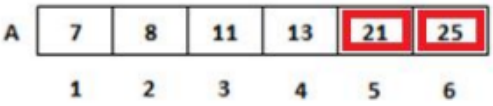
- Setelah dilakukan pembandingan sebanyak 5 kali, akhirnya didapatkan hasil pengurutan langkah pertama sebagai berikut.



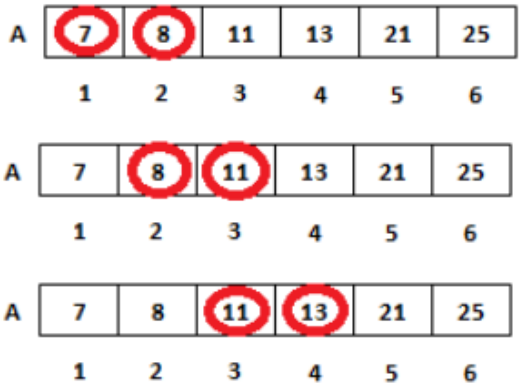
2) Langkah 2
   - Didapatkan hasil pembandingan sebagai berikut:



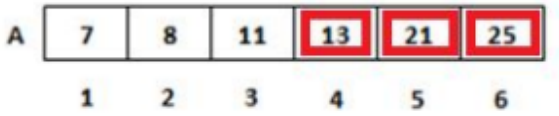
   - Maka, setelah dilakukan pembandingan sebanyak 4 kali, akhirnya didapatkan hasil pengurutan langkah kedua sebagai berikut.



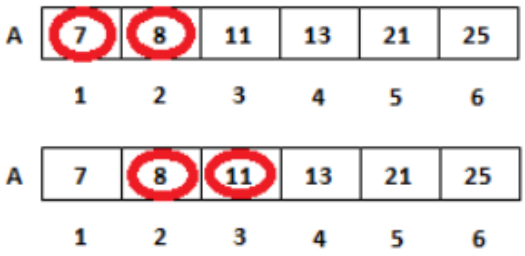
3) Langkah 3
   - Didapatkan hasil pembandingan sebagai berikut



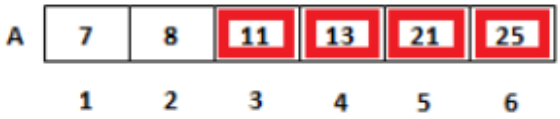
- Maka, setelah dilakukan pembandingan sebanyak 3 kali, akhirnya didapatkan hasil pengurutan langkah ketiga sebagai berikut.



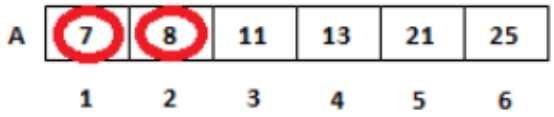
4) Langkah 4
   - Didapatkan hasil pembandingan sebagai berikut :



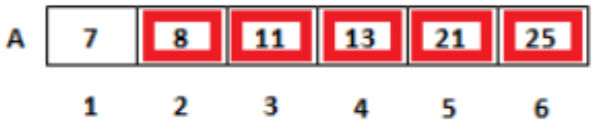
   - Maka, setelah dilakukan pembandingan sebanyak 2 kali, akhirnya didapatkan hasil pengurutan langkah keempat sebagai berikut.



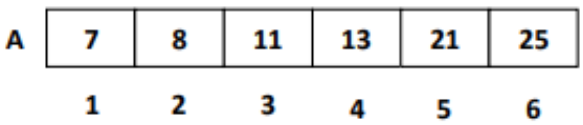
5) Langkah 5
   - Didapatkan hasil pembandingan sebagai berikut :



   - Setelah dilakukan pembandingan sebanyak 1 kali, didapatkan hasil pengurutan langkah kelima sebagai berikut



   - Maka, pada akhirnya array dapat dipastikan terurut setelah langkah kelima dikerjakan.



B. Kompleksitas Bubble Sort

Kompleksitas algoritma bubble sort dapat dilihat dari beberapa jenis kasus, yaitu *best-case*, *average-case*, dan *worst-case*.

a) Kondisi *Best-Case*

Dalam kasus ini, data yang akan disorting telah terurut sebelumnya, sehingga proses perbandingan hanya dilakukan sebanyak (n-1) kali, dengan satu kali pass. Proses perbandingan dilakukan hanya untuk memverifikasi keurutan data. Contoh *best-case* dapat dilihat pada pengurutan data (1 2 3 4) di bawah ini.

Pass Pertama

(1 2 3 4) menjadi (1 2 3 4)
(1 2 3 4) menjadi (1 2 3 4)
(1 2 3 4) menjadi (1 2 3 4)

Dari proses di atas, dapat dilihat bahwa tidak terjadi pertukaran posisi satu kalipun, sehingga tidak dilakukan pass selanjutnya. Perbandingan elemen dilakukan sebanyak tiga kali. Proses perbandingan pada kondisi ini hanya dilakukan sebanyak (n-1) kali. Persamaan Big O yang diperoleh dari proses ini adalah O(n).

b) Kondisi *Average-Case*

Pada kondisi *average-case*, jumlah pass ditentukan dari elemen mana yang mengalami penggeseran ke kiri paling banyak. Hal ini dapat ditunjukkan oleh proses pengurutan suatu *array*, misalkan (1 8 6 2). Dari (1 8 6 2), dapat dilihat bahwa yang akan mengalami proses penggeseran paling banyak adalah elemen 2, yaitu sebanyak dua kali.

Pass Pertama

(1 8 6 2) menjadi (1 8 6 2)
(1 8 6 2) menjadi (1 6 8 2)
(1 6 8 2) menjadi (1 6 2 8)

Pass Kedua

(1 6 2 8) menjadi (1 6 2 8)
(1 6 2 8) menjadi (1 2 6 8)
(1 2 6 8) menjadi (1 2 6 8)

Pass Ketiga

(1 2 6 8) menjadi (1 2 6 8)
(1 2 6 8) menjadi (1 2 6 8)
(1 2 6 8) menjadi (1 2 6 8)

Dari proses pengurutan di atas, dapat dilihat bahwa untuk mengurutkan data diperlukan dua kali pass, ditambah satu kali pass untuk verifikasi. Dengan kata lain, jumlah proses perbandingan dapat dihitung sebagai berikut.

$$Jumlah\ proses = x^2+x$$

Dalam persamaan di atas, x adalah jumlah penggeseran terbanyak. Dalam hal ini, x tidak pernah lebih besar dari n, sehingga dapat disimpulkan bahwa notasi Big O nya adalah $O(n^2)$.

c) Kondisi *Worst-Case*

Dalam kasus ini, data terkecil berada pada ujung *array*. Contoh *worst-case* dapat dilihat pada pengurutan data (4 3 2 1) di bawah ini.

Pass Pertama

(4 3 2 1) menjadi (3 4 2 1)
(3 4 2 1) menjadi (3 2 4 1)
(3 2 4 1) menjadi (3 2 1 4)

Pass Kedua

(3 2 1 4) menjadi (2 3 1 4)
(2 3 1 4) menjadi (2 1 3 4)
(2 1 3 4) menjadi (2 1 3 4)

Pass Ketiga

(2 1 3 4) menjadi (1 2 3 4)
(1 2 3 4) menjadi (1 2 3 4)
(1 2 3 4) menjadi (1 2 3 4)

Pass Keempat

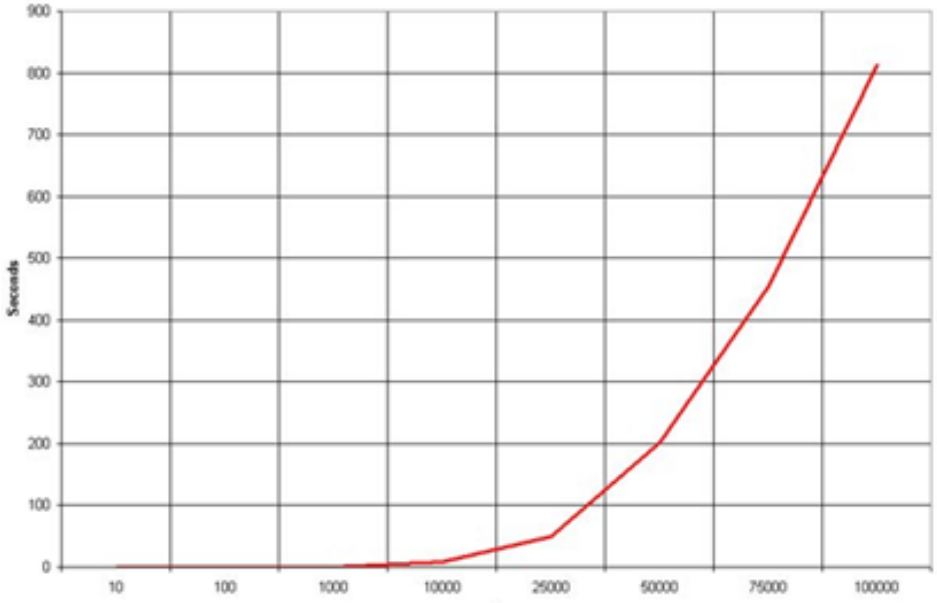
(1 2 3 4) menjadi (1 2 3 4)
(1 2 3 4) menjadi (1 2 3 4)
(1 2 3 4) menjadi (1 2 3 4)

Dari langkah pengurutan di atas, terlihat bahwa setiap kali melakukan satu pass, data terkecil akan bergeser ke arah awal sebanyak satu langkah. Dengan kata lain, untuk menggeser data terkecil dari urutan keempat menuju urutan pertama, dibutuhkan pass sebanyak tiga kali, ditambah satu kali pass untuk verifikasi. Sehingga jumlah proses pada kondisi *worst-case* dapat dirumuskan sebagai berikut.

$$Jumlah\ proses = n^2+n$$

Dalam persamaan di atas, n adalah jumlah elemen yang akan diurutkan. Sehingga notasi Big O yang didapat adalah $O(n^2)$.

C. Grafik Kompleksitas Bubble Sort



3. Kelebihan dan Kekurangan Bubble Sort

A. Kelebihan:

- Metode pengurutan yang paling simpel.
- Algoritma yang mudah dipahami.
- Definisi terurut terdapat dengan jelas dalam algoritma.
- Cocok untuk pengurutan data dengan elemen kecil yang telah terurut.

B. Kekurangan:

- Metode pengurutan yang paling tidak efisien.
- Tidak efektif dalam pengurutan data berskala besar.
- Langkah pengurutan yang terlalu panjang.