**LAPORAN TUGAS SORTING**

***Mata Kuliah Algoritma dan Struktur Data***

*Dosen pengampu:* ***Elly Warni, ST.,MT.***



**Disusun oleh:**

Andi Suci Khairunnisa (D121241085)

DEPARTEMEN TEKNIK INFORMATIKA

FAKULTAS TEKNIK

UNIVERSITAS HASANUDDIN

2025

Tugas Algoritma dan Struktur Data

1. Buatlah sebuah program C yang mengimplementasikan algoritma Bubble Sort **atau** Insertion Sort untuk mengurutkan array dari bilangan bulat dalam urutan menaik. Program akan menerima input berupa jumlah elemen pada array dan elemen-elemen dari array tersebut. Setelah diurutkan, program akan mencetak array yang telah terurut.

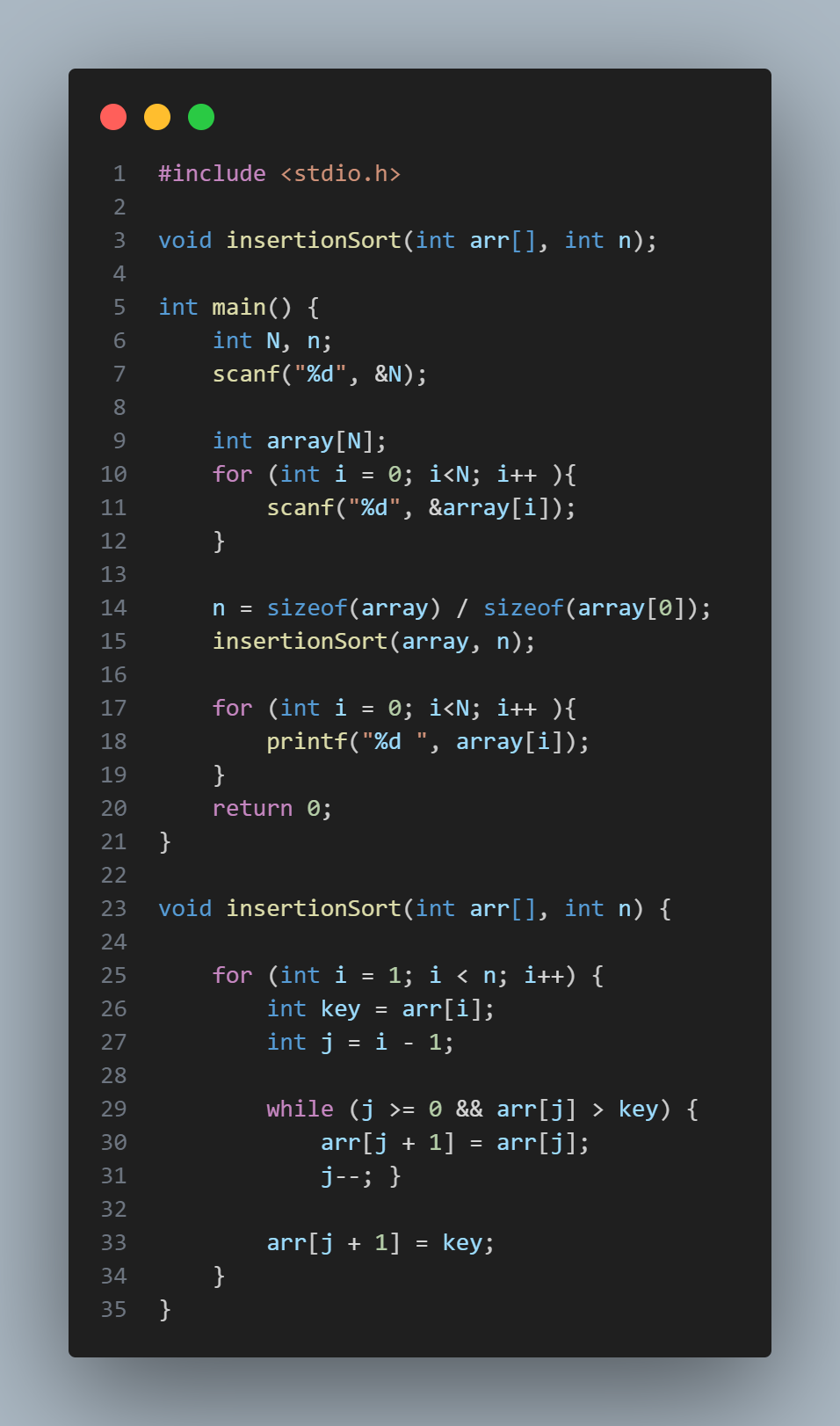
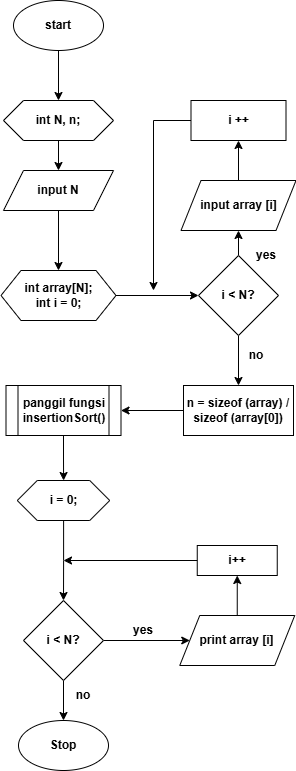
# Spesifikasi:

* + Input pertama adalah integer n, yaitu jumlah elemen pada array.
  + Input berikutnya adalah n bilangan bulat yang merupakan elemen-elemen dari array.
  + Program harus mengurutkan array menggunakan algoritma Bubble Sort **atau** Insertion Sort dan menampilkan array yang telah diurutkan.

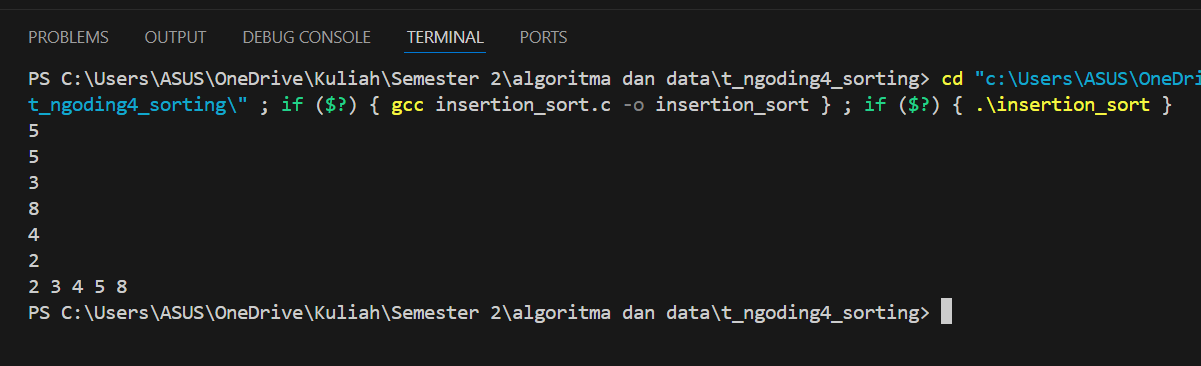
# Format Masukan dan Keluaran:

|  |  |
| --- | --- |
| Input | Output |
| 5  5 3 8 4 2 | 2 3 4 5 8 |

|  |  |
| --- | --- |
| 1. ***Source Code*** | 1. ***Flowchart*** |

* **PENYELESAIAN:**

1. ***Output Source Code***

******

1. ***Penjelasan Alur Program***
2. **Bagian Fungsi Utama (main**)

#include <stdio.h>

* Baris ini adalah perintah untuk menyertakan library *`stdio.h*`, yang berisi fungsi-fungsi standar seperti `*printf()`* untuk mencetak output dan `*scanf()`* untuk membaca input dari pengguna.

void insertionSort(int arr[], int n);

* Ini adalah deklarasi fungsi `*insertionSort`* yang akan kita gunakan untuk mengurutkan array. Fungsi ini menerima dua parameter: *`arr[]`* (array yang akan diurutkan) dan *`n`* (ukuran array). Kata `void` berarti fungsi ini tidak mengembalikan nilai apa pun.

int main() {

int N, n;

scanf("%d", &N);

* Di dalam fungsi `main()`, kita mendeklarasikan dua variabel: `*N*` dan `*n*`.
* *`scanf("%d", &N)`* digunakan untuk membaca input dari pengguna berupa bilangan bulat yang menunjukkan jumlah elemen dalam array (ukuran array).

int array[N];

for (int i = 0; i<N; i++ ){

scanf("%d", &array[i]);

}

* Di sini, kita membuat array bernama `*array*` dengan ukuran `*N*` (sesuai input pengguna).
* Kemudian, kita menggunakan loop `*for*` untuk membaca `*N*` buah bilangan bulat dari pengguna dan menyimpannya ke dalam array. Misalnya, jika `*N = 5`,* maka pengguna akan diminta memasukkan 5 angka.

n = sizeof(array) / sizeof(array[0]);

* Baris ini menghitung ukuran array dengan cara membagi total ukuran array dalam byte *(`sizeof(array)`)* dengan ukuran satu elemen array *(`sizeof(array[0])`).* Hasilnya adalah jumlah elemen dalam array, yang disimpan di variabel `*n*`. Sebenarnya, karena `*N*` sudah ada, kita bisa langsung menggunakan `*N*` sebagai ukuran array, tapi cara ini juga sering digunakan untuk menghitung ukuran array secara dinamis.

insertionSort(array, n);

* + Di sini, kita memanggil fungsi *`insertionSort*` untuk mengurutkan array. Kita mengirimkan array `*array`* dan ukurannya *`n`* sebagai parameter.

for (int i = 0; i<N; i++ ){

printf("%d ", array[i]);

}

return 0;

}

* + Setelah array diurutkan, kita mencetak isi array yang sudah terurut menggunakan loop `*for*` dan fungsi `*printf()`.* Setiap elemen dipisahkan dengan spasi.
  + Terakhir, `*return* *0*` menandakan bahwa program selesai dengan sukses.

---

**2. Bagian Fungsi Insertion Sort**

void insertionSort(int arr[], int n) {

for (int i = 1; i < n; i++) {

* + Fungsi ini menerima array `*arr*[]` dan ukurannya `*n*`.
  + Loop `*for*` dimulai dari indeks `*i = 1`* (elemen kedua) hingga `*i < n*` (elemen terakhir). Mengapa mulai dari 1? Karena kita menganggap elemen pertama (indeks 0) sudah "terurut" sebagai awalan.

int key = arr[i];

int j = i - 1;

* + Di setiap iterasi, kita mengambil elemen saat ini sebagai `*key*` (kunci). Elemen ini akan kita sisipkan ke posisi yang tepat di bagian array yang sudah terurut (dari indeks 0 sampai *`i-1`).*
  + Variabel `*j*` diatur ke indeks sebelumnya *(`i-1*`), karena kita akan membandingkan *`key`* dengan elemen-elemen sebelumnya.

while (j >= 0 && arr[j] > key) {

arr[j + 1] = arr[j];

j--; }

* + Loop *`while*` ini berjalan selama *`j*` masih dalam batas *array (j >= 0*) dan elemen di indeks *`j`* lebih besar dari `*key`*. Jika kondisi ini terpenuhi, kita akan menggeser elemen `*arr[j]`* ke kanan (ke indeks *`j + 1*`), sehingga membuat ruang untuk `*key`* di posisi yang tepat.
  + Setelah menggeser elemen, kita mengurangi `j` untuk memeriksa elemen sebelumnya.

arr[j + 1] = key;

* + Setelah keluar dari loop *`while`,* kita menempatkan `*key`* di posisi yang benar, yaitu di *`arr[j + 1]`.* Ini menyelesaikan proses penyisipan untuk elemen saat ini.
  + Proses ini diulang untuk setiap elemen dalam array hingga seluruh array terurut.

Dengan demikian, fungsi `insertionSort` berhasil mengurutkan array dengan cara menyisipkan setiap elemen ke posisi yang tepat dalam bagian array yang sudah terurut. Algoritma ini efisien untuk array kecil dan memiliki kompleksitas waktu O(n^2) dalam kasus terburuk, tetapi dapat bekerja dengan baik pada data yang hampir terurut.

1. Diberikan sebuah array yang berisi angka-angka acak, tugas Anda adalah mengurutkan array tersebut menggunakan algoritma QuickSort dan menampilkan hasil array yang sudah terurut.

# Deskripsi Input:

Sebuah array dengan elemen-elemen angka integer.

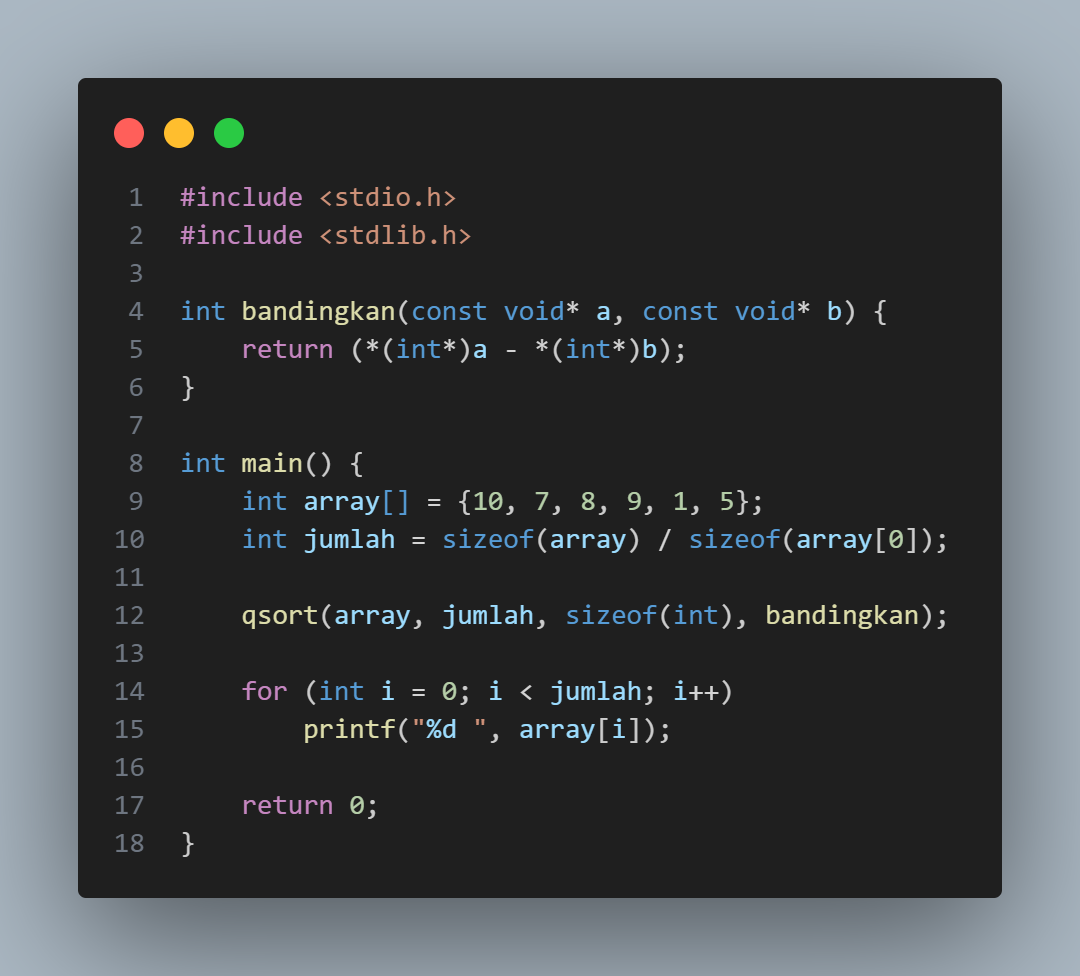
# Deskripsi Output:

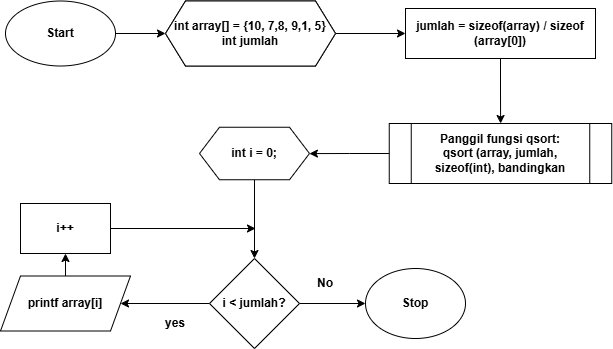
Array yang telah diurutkan dari yang terkecil hingga yang terbesar.

**Format Masukan dan Keluaran:**

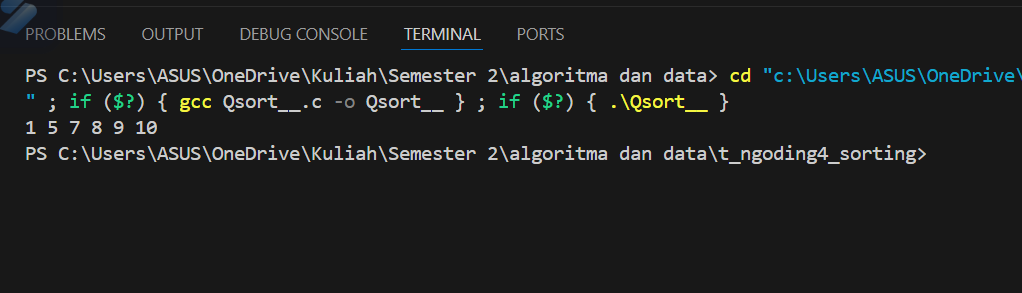
|  |  |
| --- | --- |
| Input | Output |
| 10, 7, 8, 9, 1, 5 | 1 5 7 8 9 10 |

* **PENYELESAIAN:**

1. ***Source Code***
2. ***Flowchart***

****

1. ***Output Source Code***

******

1. ***Penjelasan Alur Program***
2. Header File

* *#include <stdio.h>* : Ini adalah header file yang digunakan untuk fungsi input dan output standar, seperti `printf`.
* *#include <stdlib.h>*: Ini adalah header file yang digunakan untuk fungsi-fungsi umum, termasuk fungsi `qsort` yang kita gunakan untuk mengurutkan array.

2. Fungsi Pembanding:

*int bandingkan(const void\* a, const void\* b) {*

*return (\*(int\*)a - \*(int\*)b);*

* + Di sini kita mendefinisikan sebuah fungsi bernama `*bandingkan*` yang digunakan untuk membandingkan dua elemen.
  + Fungsi ini menerima dua parameter bertipe `*const void*\*`, yang berarti kita tidak tahu tipe data apa yang akan diterima. Ini adalah cara umum untuk membuat fungsi pembanding yang fleksibel.
  + Di dalam fungsi, kita mengubah `*void*\*` menjadi `*int*\*` dengan cara *`\*(int\*)a`* dan *`\*(int\*)b`,* sehingga kita bisa mengakses nilai integer yang sebenarnya.
  + Fungsi ini mengembalikan selisih antara dua angka. Jika hasilnya negatif, berarti *`a`* lebih kecil dari *`b`*, jika positif berarti *`a*` lebih besar dari *`b`*, dan jika nol berarti keduanya sama.

3. Fungsi Utama (main):

int main() {

int array[] = {10, 7, 8, 9, 1, 5};

int jumlah = sizeof(array) / sizeof(array[0]);

* + Di dalam fungsi `*main*`, kita mendeklarasikan sebuah array integer bernama `*array`* yang berisi beberapa angka.
  + *`int jumlah = sizeof(array) / sizeof(array[0]);`*: Di sini kita menghitung jumlah elemen dalam array. `*sizeof(array)`* memberikan ukuran total dari array dalam byte, dan `*sizeof(array[0])`* memberikan ukuran dari satu elemen array. Dengan membagi keduanya, kita mendapatkan jumlah elemen dalam array.

4. Mengurutkan Array:

qsort(array, jumlah, sizeof(int), bandingkan);

* + *`qsort`* adalah fungsi dari `*stdlib.h*` yang digunakan untuk mengurutkan array.
  + Parameter yang diberikan adalah:
  + *`array`*: array yang ingin kita urutkan.
  + *`jumlah`*: jumlah elemen dalam array.
  + *`sizeof(int)`*: ukuran dari setiap elemen dalam array (dalam byte).
  + *`bandingkan`*: fungsi yang kita buat sebelumnya untuk membandingkan dua elemen.

5. Mencetak Hasil:

for (int i = 0; i < jumlah; i++)

printf("%d ", array[i]);

- Di sini kita menggunakan loop *`for`* untuk mencetak setiap elemen dari array yang sudah diurutkan.

- *`printf("%d ", array[i]);`* akan mencetak setiap elemen integer di array diikuti dengan spasi.

6. Mengakhiri Program:

* + `*return 0;`* menandakan bahwa program telah selesai dijalankan dengan sukses.