

**LAPORAN PRAKTIKUM
ALGORITMA & STRUKTUR DATA
MODUL 5**



SORTING

Oleh:

Noor Khalisa NIM. 2410817220012

**PROGRAM STUDI TEKNOLOGI INFORMASI
FAKULTAS TEKNIK
UNIVERSITAS LAMBUNG MANGKURAT
MEI 2025**

LEMBAR PENGESAHAN
LAPORAN PRAKTIKUM ALGORITMA & STRUKTUR DATA
MODUL 5

Laporan Praktikum Algoritma & Struktur Data Modul 5: Sorting ini disusun sebagai syarat lulus mata kuliah Praktikum Algoritma & Struktur Data. Laporan Praktikum ini dikerjakan oleh:

Nama Praktikan : Noor Khalisa
NIM : 24101817220012

Menyetujui,
Asisten Praktikum

Mengetahui,
Dosen Penanggung Jawab Praktikum

Muhammad Fauzan Ahsani
NIM. 2310817310009

Muti'a Maulida, S.Kom., M.TI.
NIP. 198810272019032013

DAFTAR ISI

LEMBAR PENGESAHAN	2
DAFTAR ISI.....	3
DAFTAR GAMBAR	4
DAFTAR TABEL	5
SOAL	6
A. Source Code	6
B. Output.....	12
C. Pembahasan.....	15

DAFTAR GAMBAR

Gambar 1 Insertion Sort	12
Gambar 2 Merge Sort.....	13
Gambar 3 Shell Sort	13
Gambar 4 Quick Sort	14
Gambar 5 Bubble Sort.....	14
Gambar 6 Selection Sort	15
Gambar 7 Exit	15

DAFTAR TABEL

Tabel 1 Source Code Soal	6
--------------------------------	---

SOAL

Buat Program Sederhana Menggunakan Nama dan Angka NIM masing-masing:

- Insertion Sort (Nama)
- Merge Sort (Nama)
- Shell Sort (Nama)
- Quick Sort (NIM)
- Bubble Sort (NIM)
- Selection Sort (NIM)



A. Source Code

Tabel 1 Source Code Soal

1	#include <iostream>
2	#include <functional>
3	#include <chrono>
4	#include <string>
5	#include <algorithm>
6	#include <iomanip>
7	
8	using namespace std;
9	
10	string name, id;
11	

```

12 void timeSort(const function<void()>& sortFunc,
13 const string& sortName) {
14     auto start = chrono::high_resolution_clock::now();
15     sortFunc();
16     auto end = chrono::high_resolution_clock::now();
17     chrono::duration<double> duration = end - start;
18     cout << sortName << " took " << fixed <<
19     setprecision(10) << duration.count() << " seconds\n";
20 }
21 void merge(string& str, int left, int mid, int
22 right){
23     int leftSize = mid - left + 1;
24     int rightSize = right - mid;
25     char* leftSide = new char[leftSize];
26     char* rightSide = new char[rightSize];
27     for(int i = 0; i < leftSize; i++)
28         leftSide[i] = str[left + i];
29     for(int j = 0; j < rightSize; j++)
30         rightSide[j] = str[mid + 1 + j];
31
32     int l = 0, r = 0, k = left;
33     while(l < leftSize && r < rightSize){
34         if(leftSide[l] <= rightSide[r]){
35             str[k++] = leftSide[l++];
36         } else {
37             str[k++] = rightSide[r++];
38         }
39     }
40     while (l < leftSize)
41         str[k++] = leftSide[l++];
42     while (r < rightSize)
43         str[k++] = rightSide[r++];
44
45     delete[] leftSide;
46     delete[] rightSide;
47 }
48
49 void mergeSortRecursive(string& str, int left, int
50 right) {
51     if(left < right) {
52         int mid = left + (right - left) / 2;

```

```

52         mergeSortRecursive(str, left, mid);
53         mergeSortRecursive(str, mid + 1, right);
54         merge(str, left, mid, right);
55     }
56 }
57
58 void insertionSort(string& str) {
59     int n = str.length();
60     for(int i = 1; i < n; i++) {
61         char key = str[i];
62         int j = i - 1;
63         while(j >= 0 && str[j] > key) {
64             str[j + 1] = str[j];
65             j--;
66         }
67         str[j + 1] = key;
68     }
69 }
70
71 void mergeSort(string& str) {
72     mergeSortRecursive(str, 0, str.length() - 1);
73 }
74
75 void shellSort(string& str){
76     int n = str.length();
77     for (int gap = n / 2; gap > 0; gap /= 2) {
78         for (int i = gap; i < n; i++) {
79             char temp = str[i];
80             int j;
81             for (j = i; j >= gap && str[j - gap] >
temp; j -= gap) {
82                 str[j] = str[j - gap];
83             }
84             str[j] = temp;
85         }
86     }
87 }
88
89 void bubbleSort(string& str) {
90     int n = str.length();
91     bool swapped;
92     for(int i = 0; i < n - 1; i++) {
93         swapped = false;
94         for (int j = 0; j < n - i - 1; j++) {
95             if (str[j] > str[j + 1]) {

```



```

96         swap(str[j], str[j + 1]);
97         swapped = true;
98     }
99 }
100     if (!swapped) break; // If no two elements
    were swapped, the array is sorted
101 }
102 }
103
104 int partition(string& str, int low, int high) {
105     char pivot = str[high];
106     int i = (low - 1);
107     for (int j = low; j <= high - 1; j++) {
108         if (str[j] < pivot) {
109             i++;
110             swap(str[i], str[j]);
111         }
112     }
113     swap(str[i + 1], str[high]);
114     return (i + 1);
115 }
116
117 void quickSortRecursive(string& str, int low, int
    high) {
118     if (low < high) {
119         int pi = partition(str, low, high);
120         quickSortRecursive(str, low, pi - 1);
121         quickSortRecursive(str, pi + 1, high);
122     }
123 }
124
125 void quickSort(string& str) {
126     quickSortRecursive(str, 0, str.length() - 1);
127 }
128
129 void selectionSort(string& str){
130     int n = str.length();
131     for (int i = 0; i < n - 1; i++) {
132         int minIndex = i;
133         for (int j = i + 1; j < n; j++) {
134             if (str[j] < str[minIndex]) {
135                 minIndex = j;
136             }
137         }
138         swap(str[i], str[minIndex]);

```

```

139     }
140 }
141
142 int main(){
143     int menu;
144     string name, id;
145     do{
146         cout << "\n-----" <<
endl;
147         cout << "|-----SORTING-----|" << endl;
148         cout << "-----" << endl;
149         cout << "1. Insertion Sort" << endl;
150         cout << "2. Merge Sort" << endl;
151         cout << "3. Shell Sort" << endl;
152         cout << "4. Quick Sort" << endl;
153         cout << "5. Bubble Sort" << endl;
154         cout << "6. Selection Sort" << endl;
155         cout << "7. Exit" << endl;
156         cout << "-----" << endl;
157         cout << "Masukkan Pilihan: ";
158         cin >> menu;
159         cin.ignore();
160
161         if(menu >= 1 && menu <= 3) {
162             cout << "Masukkan Nama: ";
163             getline(cin, name);
164         } else if(menu >= 4 && menu <= 6) {
165             cout << "Masukkan NIM: ";
166             getline(cin, id);
167         }
168         string nama, nim;
169         switch(menu) {
170             case 1:
171                 cout << "\nInsertion Sort" << endl;
172                 nama = name;
173
174                 cout << "Nama Awal: " << name <<
endl;
175                 timeSort([&
{ insertionSort(nama); }, "Insertion Sort");
176                 cout << "Nama Terurut: " << nama <<
endl;
177                 break;
178             case 2:
179                 cout << "\nMerge Sort" << endl;

```

```

180         nama = name;
181
182         cout << "Nama Awal: " << name <<
endl;
183         timeSort([&] { mergeSort(nama); },
"Merge Sort");
184         cout << "Nama Terurut: " << nama <<
endl;
185         break;
186     case 3:
187         cout << "\nShell Sort" << endl;
188         nama = name;
189
190         cout << "Nama Awal: " << name <<
endl;
191         timeSort([&] { shellSort(nama); },
"Shell Sort");
192         cout << "Nama Terurut: " << nama <<
endl;
193         break;
194     case 4:
195         cout << "\nQuick Sort" << endl;
196         nim = id;
197
198         cout << "\nNIM Awal: " << id << endl;
199         timeSort([&] { quickSort(nim); },
"Quick Sort");
200         cout << "NIM Terurut: " << nim <<
endl;
201         break;
202     case 5:
203         cout << "\nBubble Sort" << endl;
204         nim = id;
205
206         cout << "\nNIM Awal: " << id << endl;
207         timeSort([&] { bubbleSort(nim); },
"Bubble Sort");
208         cout << "NIM Terurut: " << nim <<
endl;
209         break;
210     case 6:
211         cout << "\nSelection Sort" << endl;
212         nim = id;
213
214         cout << "\nNIM Awal: " << id << endl;

```

```

215         timeSort([&
{ selectionSort(nim); }, "Selection Sort");
216         cout << "NIM Terurut: " << nim <<
endl;
217         break;
218         case 7:
219             cout << "Exiting program." << endl;
220             break;
221     }
222     } while(menu != 7);
223     return 0;
224 }

```

B. Output

```

-----
|-----SORTING-----|
-----
1. Insertion Sort
2. Merge Sort
3. Shell Sort
4. Quick Sort
5. Bubble Sort
6. Selection Sort
7. Exit
-----
Masukkan Pilihan: 1
Masukkan Nama: Noor Khalisa

Insertion Sort
Nama Awal: Noor Khalisa
Insertion Sort took 0.0000048000 seconds
Nama Terurut: KNaahiloors

```

Gambar 1 Insertion Sort

```

-----
|-----SORTING-----|
-----
1. Insertion Sort
2. Merge Sort
3. Shell Sort
4. Quick Sort
5. Bubble Sort
6. Selection Sort
7. Exit
-----
Masukkan Pilihan: 2
Masukkan Nama: Noor Khalisa

Merge Sort
Nama Awal: Noor Khalisa
Merge Sort took 0.0000333000 seconds
Nama Terurut: KNaahiloors

```

Gambar 2 Merge Sort

```

-----
|-----SORTING-----|
-----
1. Insertion Sort
2. Merge Sort
3. Shell Sort
4. Quick Sort
5. Bubble Sort
6. Selection Sort
7. Exit
-----
Masukkan Pilihan: 3
Masukkan Nama: Noor Khalisa

Shell Sort
Nama Awal: Noor Khalisa
Shell Sort took 0.0000051000 seconds
Nama Terurut: KNaahiloors

```

Gambar 3 Shell Sort

```

-----
|-----SORTING-----|
-----
1. Insertion Sort
2. Merge Sort
3. Shell Sort
4. Quick Sort
5. Bubble Sort
6. Selection Sort
7. Exit
-----
Masukkan Pilihan: 4
Masukkan NIM: 2410817220012

Quick Sort

NIM Awal: 2410817220012
Quick Sort took 0.0000046000 seconds
NIM Terurut: 0001112222478

```

Gambar 4 Quick Sort

```

-----
|-----SORTING-----|
-----
1. Insertion Sort
2. Merge Sort
3. Shell Sort
4. Quick Sort
5. Bubble Sort
6. Selection Sort
7. Exit
-----
Masukkan Pilihan: 5
Masukkan NIM: 2410817220012

Bubble Sort

NIM Awal: 2410817220012
Bubble Sort took 0.0000081000 seconds
NIM Terurut: 0001112222478

```

Gambar 5 Bubble Sort

```

-----
|-----SORTING-----|
-----
1. Insertion Sort
2. Merge Sort
3. Shell Sort
4. Quick Sort
5. Bubble Sort
6. Selection Sort
7. Exit
-----
Masukkan Pilihan: 6
Masukkan NIM: 2410817220012

Selection Sort

NIM Awal: 2410817220012
Selection Sort took 0.0000052000 seconds
NIM Terurut: 0001112222478

```

Gambar 6 Selection Sort

```

-----
|-----SORTING-----|
-----
1. Insertion Sort
2. Merge Sort
3. Shell Sort
4. Quick Sort
5. Bubble Sort
6. Selection Sort
7. Exit
-----
Masukkan Pilihan: 7
Exiting program.
PS C:\Users\ACER\Kuliah TI\Semes

```

Gambar 7 Exit

C. Pembahasan

Struktur Kode Umum

Header Files (#include) adalah pustaka yang berisi kode-kode siap pakai.

- `<iostream>` digunakan untuk input dan output standar (seperti `cout` untuk mencetak ke konsol dan `cin` untuk membaca input dari konsol).
- `<functional>` menyediakan fungsionalitas untuk objek fungsi, seperti `std::function` yang digunakan dalam fungsi `timeSort`.
- `<chrono>` digunakan untuk mengukur waktu eksekusi kode dengan presisi tinggi.
- `<string>` menyediakan kelas `std::string` untuk manipulasi string.
- `<algorithm>` berisi berbagai algoritma standar, termasuk `std::swap` yang digunakan dalam berbagai algoritma pengurutan.
- `<iomanip>` digunakan untuk memanipulasi format output, seperti `std::fixed` dan `std::setprecision` untuk mengatur presisi angka decimal.

`using namespace std;` adalah deklarasi `using` yang membawa semua nama dari namespace `std` ke cakupan global sehingga bisa menulis `cout`, `cin`, dan `string` secara langsung.

`string name, id` mendeklarasikan dua variabel global bertipe `string` bernama `name` dan `id`.

Fungsi `timeSort`

Fungsi `timeSort` berfungsi untuk mengukur seberapa cepat sebuah algoritma pengurutan berjalan.

Parameter:

- `const function<void()>& sortFunc` menerima fungsi yang akan dijalankan dan diukur waktunya. Fungsi yang diteruskan tidak boleh memiliki parameter dan tidak mengembalikan nilai.
- `const string& sortName` menerima nama dari algoritma pengurutan sebagai string, yang akan dicetak sebagai bagian dari output waktu.

Cara Kerja:

1. Mencatat waktu awal (start) sebelum memanggil sortFunc.
2. Mengeksekusi sortFunc().
3. Mencatat waktu akhir (end) setelah sortFunc selesai.
4. Menghitung durasi (duration) dengan mengurangi start dari end.
5. Mencetak sortName dan duration dengan durasi ditampilkan dalam detik dengan 10 angka di belakang koma.

Algoritma Pengurutan

1. Fungsi merge

Fungsi merge bertugas untuk menggabungkan dua bagian (sub-array) yang sudah terurut dari sebuah string menjadi satu bagian yang terurut penuh. Ini adalah bagian inti dari algoritma Merge Sort.

Parameter:

- string& str referensi ke string utama yang sedang diurutkan (karakter-karakternya akan dimodifikasi).
- int left, int mid, int right menentukan rentang indeks dalam str yang akan digabungkan. left adalah awal sub-array kiri, mid adalah akhir sub-array kiri (dan mid + 1 adalah awal sub-array kanan), dan right adalah akhir sub-array kanan.

Cara Kerja:

1. Menciptakan dua array sementara (leftSide dan rightSide) untuk menyimpan karakter dari dua sub-array yang akan digabungkan.
 2. Menyalin karakter dari str ke dalam array-array sementara ini.
 3. Membandingkan karakter dari leftSide dan rightSide satu per satu. Karakter yang lebih kecil disalin Kembali ke posisi yang benar di str.
 4. Setelah salah satu array sementara habis, sisa karakter dari array yang lain disalin ke str.
- #### **2. Fungsi mergeSortRecursive**

Fungsi `mergeSortRecursive` berfungsi untuk memecah masalah pengurutan string menjadi masalah-masalah kecil yang lebih mudah diselesaikan, lalu menggabungkan hasilnya. Ini adalah bagian rekursif dari Merge Sort.

Parameter:

- `string& str` referensi ke string utama yang sedang diurutkan (karakter-karakternya akan dimodifikasi). Ini adalah string yang akan diteruskan ke panggilan rekursif pertama `mergeSortRecursive`.
- `int left` mewakili indeks awal (paling kiri) dari segmen string yang akan diurutkan dalam panggilan rekursif saat ini. `int right` mewakili indeks akhir (paling kanan) dari segmen string yang akan diurutkan dalam panggilan rekursif saat ini.

Cara Kerja:

1. Kondisi Dasar: Jika `left` tidak kurang dari `right`, artinya segmen hanya memiliki satu karakter atau kosong. Maka segmen tersebut sudah terurut dan fungsi berhenti.
 2. Divide (Pecah): Menghitung indeks tengah (`mid`) untuk membagi string menjadi dua bagian.
 3. Conquer (Taklukkan): Memanggil dirinya sendiri secara rekursif untuk mengurutkan bagian kiri (`left` hingga `mid`) dan bagian kanan (`mid + 1` hingga `right`).
 4. Combine (Gabungkan): Setelah kedua bagian rekursif selesai diurutkan, fungsi `merge` dipanggil untuk menggabungkan dua sub-array yang sudah terurut menjadi satu segmen yang terurut.
3. Fungsi `mergeSort`
- Fungsi `mergeSort` berfungsi sebagai titik masuk utama untuk memulai proses pengurutan menggunakan algoritma Merge Sort. Fungsi ini hanya memanggil fungsi `mergeSortRecursive` untuk mengurutkan seluruh string, mulai dari indeks pertama hingga indeks terakhir (`str.length() - 1`).

Parameter: string& str referensi ke string utama yang sedang diurutkan.

Cara Kerja:

Hanya memanggil mergeSortRecursive untuk mengurutkan string dari indeks 0 hingga indeks terakhir (`str.length() - 1`). Ini adalah fungsi pembungkus yang memudahkan panggilan awal.

4. Fungsi insertionSort

Fungsi insertionSort berfungsi untuk mengurutkan string dengan cara menyisipkan setiap karakter ke posisi yang benar di dalam bagian string yang sudah terurut.

Parameter: string& str referensi ke string utama yang sedang diurutkan.

Cara Kerja:

1. Iterasi dimulai dari karakter kedua ($i = 1$).
2. Setiap karakter (`str[i]`) dianggap sebagai key yang akan disisipkan.
3. Loop mundur ($j = i - 1$) membandingkan key dengan karakter-karakter di bagian yang sudah terurut di sebelah kirinya.
4. Jika karakter di sebelah kiri lebih besar dari key, karakter tersebut digeser satu posisi ke kanan (`str[j + 1] = str[j]`) untuk memberi ruang.
5. Proses pergeseran berlanjut sampai posisi yang benar untuk key ditemukan (yaitu, j menjadi negative atau `str[j]` tidak lagi lebih besar dari key).
6. key kemudian ditempatkan pada posisi yang benar (`str[j + 1] = key`).

5. Fungsi shellSort

Fungsi shellSort berfungsi untuk mengurutkan string dengan membandingkan dan menukar karakter yang berjauhan terlebih dahulu, lalu secara bertahap mengurangi jarak perbandingan tersebut. Ini adalah versi yang lebih cepat dari Insertion Sort.

Parameter: string& str referensi ke string utama yang sedang diurutkan.

Cara Kerja:

1. Loop luar mengontrol ukuran gap (jarak antar elemen yang dibandingkan), dimulai dari $n/2$ dan terus dibagi 2 hingga menjadi 1.
 2. Loop dalam melakukan semacam “Insertion Sort” pada sub-list di mana elemen-elemennya dipisahkan oleh gap yang ditentukan.
 3. Karakter (temp) dibandingkan dengan elemen-elemen di temp – gap, temp 2*gap, dll. Lalu digeser jika lebih besar, sampai posisi yang benar ditemukan.
 4. Setelah gap menjadi 1, algoritma pada dasarnya adalah Insertion Sort pada data yang sudah sebagian besar terurut, sehingga lebih efisien.
6. Fungsi bubbleSort

Fungsi bubbleSort berfungsi untuk mengurutkan string dengan berulang kali membandingkan dan menukar pasangan karakter yang berdekatan jika urutannya salah.

Parameter: string& str referensi ke string utama yang sedang diurutkan.

Cara Kerja:

1. Loop luar mengontrol jumlah *pass* melalui string. *Pass* adalah satu siklus penuh perbandingan dan potensi penukaran elemen yang berdekatan, tujuannya untuk memindahkan elemen terbesar yang belum diurutkan ke posisi yang benar di setiap akhir siklus.
 2. Di setiap *pass*, loop dalam membandingkan setiap karakter (str[j]) dengan karakter di sebelahnya (str[j + 1]).
 3. Jika str[j] lebih besar dari (str[j + 1]), kedua karakter ditukar menggunakan std::swap.
 4. Sebuah variabel swapped digunakan sebagai optimasi: jika tidak ada penukaran yang terjadi dalam satu pass penuh, berarti string sudah terurut dan algoritma berhenti lebih awal.
7. Fungsi partition

Fungsi partition berfungsi untuk mengatur ulang bagian dari string sedemikian rupa sehingga semua karakter yang lebih kecil dari sebuah “pivot” (karakter

pembandingan) berada di sebelah kirinya, dan semua karakter yang lebih besar berada di sebelah kanannya. Ini adalah bagian inti dari algoritma Quick Sort.

Parameter:

- string& str referensi ke string utama yang sedang diurutkan.
- int low, int high menentukan rentang indeks dalam str yang akan dipartisi.

Cara Kerja:

1. Memilih elemen terakhir (str[high]) sebagai pivot.
2. Menggunakan indeks i untuk melacak batas elemen yang lebih kecil dari pivot.
3. Iterasi dari low hingga high-1. Jika sebuah karakter (str[j]) ditemukan lebih kecil dari pivot, i ditingkatkan dan str[j] ditukar dengan str[i].
4. Setelah iterasi, pivot ditukar ke posisi yang benar (i + 1), sehingga berada di antara elemen yang lebih kecil dan lebih besar.

8. Fungsi quickSortRecursive

Fungsi quickSortRecursive berfungsi untuk mengurutkan string dengan memecahnya menjadi bagian-bagian yang lebih kecil, mengurutkan bagian-bagian tersebut secara independent, dan kemudian secara otomatis menyatukannya karena proses partisi. Ini adalah bagian rekursif dari Quick Sort.

Parameter:

- string& str referensi ke string utama yang sedang diurutkan.
- int low, int high menentukan rentang indeks dalam str yang akan dipartisi.

Cara Kerja:

1. Kondisi Dasar: Jika low tidak kurang dari high, segmen sudah berurutan dan fungsi berhenti.
2. Divide (Pecah) & Conquer (Taklukkan): Memanggil fungsi partition untuk mengatur ulang segmen saat ini dan mendapatkan indeks pivot (pi).

3. Conquer (Taklukkan): Memanggil dirinya sendiri secara rekursif untuk mengurutkan sub-array di sebelah kiri pivot (low hingga $pi - 1$) dan di sebelah kanan pivot ($pi + 1$ hingga high).

9. Fungsi quickSort

Fungsi quickSort berfungsi sebagai titik masuk utama untuk memulai proses pengurutan menggunakan algoritma Quick Sort. Fungsi ini hanya memanggil fungsi quickSortRecursive untuk mengurutkan seluruh string, mulai dari indeks pertama hingga indeks terakhir (`str.length() - 1`).

Parameter: `string& str` referensi ke string utama yang sedang diurutkan.

Cara Kerja: hanya memanggil quickSortRecursive untuk mengurutkan string dari indeks 0 hingga indeks terakhir,

10. Fungsi selectionSort

Fungsi selectionSort berfungsi untuk mengurutkan string dengan berulang kali mencari karakter terkecil dari bagian yang belum diurutkan dan memindahkannya ke posisi awal bagian yang belum terurut tersebut.

Parameter: `string& str` referensi ke string utama yang sedang diurutkan.

Cara Kerja:

1. Loop luar (*i*) melintasi string dari awal hingga karakter kedua terakhir.
2. Di setiap iteraasi *i*, ia mengasumsikan `str[i]` adalah yang terkecil (`minIndex = i`).
3. Loop dalam (*j*) mencari sisa string (*i + 1* hingga akhir) untuk menemukan karakter yang paling kecil. Jika karakter yang lebih kecil ditemukan, `minIndex` diperbarui.
4. Setelah loop dalam selesai, `minIndex` menunjuk ke karakter terkecil di bagian yang belum diurutkan. Karakter ini kemudian ditukar dengan `str[i]` menggunakan `std::swap`, menempatkan elemen terkecil pada posisi yang benar.

Fungsi main

Fungsi main adalah tempat program mulai berjalan, fungsi ini mengelola interaksi dengan user dan alur eksekusi. Fungsi ini terus berjalan dalam sebuah lingkaran (do-while) sampai user memilih keluar.

int menu adalah variabel yang akan menyimpan pilihan angka yang akan dimasukkan user dari menu yang ditampilkan.

string name, id adalah variabel yang akan menyimpan nama atau NIM yang akan dimasukkan oleh user untuk diurutkan.

Alur kerja dalam main():

1. Pengulangan Menu dan Input (Loop do-while)

Program memulai dengan `do { ... } while(menu != 7);`. Artinya semua kode di dalam blok `do` akan selalu dijalankan minimal satu kali. Setelah itu, program akan terus mengulang bagian tersebut selama nilai menu (pilihan user) bukan 7. Pilihan 7 adalah cara untuk keluar dari program.

2. Menampilkan Menu

Di setiap awal `do-while`, beberapa perintah `cout` akan menampilkan menu pilihan algoritma pengurutan serta opsi untuk keluar dari program.

3. Menerima Pilihan User

`cin >> menu` akan menunggu user mengetikkan angka pilihan dan menekan enter. Angka itu akan disimpan ke variabel `menu`. Pentingnya `cin.ignore()` di sini adalah untuk membersihkan sisa enter yang ditekan, agar input teks berikutnya (Nama atau NIM) tidak terlewat atau error.

4. Meminta Input Data (Nama atau NIM)

Setelah pilihan menu dibaca, program akan memutuskan data mana yang akan diminta.

- Jika user memilih algoritma 1, 2, dan 3 (untuk Nama), program akan menampilkan "Masukkan Nama:" dan memakai `getline(cin, name)` untuk membaca seluruh baris nama yang dimasukkan user.

- Jika user memilih algoritma 4, 5, dan 6 (untuk NIM), program akan menampilkan “Masukkan NIM:” dan memakai `getline(cin, id)` untuk membaca NIM yang dimasukkan.

5. Menyiapkan Data untuk Pengurutan

Nama atau NIM yang dimasukkan akan disalin ke variabel sementara (nama atau nim). Ini dilakukan agar data asli tetap tersimpan untuk ditampilkan sebagai “Nama Awal” atau “NIM Awal”, sementara salinannya yang akan diubah oleh algoritma pengurutan.

6. Memilih dan Menjalankan Algoritma (Blok switch)

Blok `switch(menu)` akan menjalankan kode yang spesifik sesuai dengan nilai menu yang dipilih user.

- Untuk pilihan 1 sampai 6: Program menampilkan nama algoritma, data asli, lalu memanggil `timeSort`. `timeSort` akan menjalankan algoritma pengurutan pada data yang sudah disalin, sambil mengukur berapa lama waktu yang diperlukan. Setelah selesai, program akan menampilkan data yang sudah terurut beserta waktu eksekusinya. Perintah `break` mengakhiri proses untuk pilihan ini dan kembali ke menu utama.
- Untuk pilihan 7 (keluar): Program menampilkan pesan “Exiting Program.” Dan kemudian berhenti berulang.

7. Mengakhiri Program

Setelah memilih 7, program akan selesai sepenuhnya dan mengembalikan nilai 0, yang menandakan eksekusi berhasil.