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



**Sorting**

|  |  |
| --- | --- |
| **Oleh:** | |
| **Rizki Adhitiya Maulana** | **NIM. 2410817110014** |

**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 : Rizki Adhitiya Maulana

NIM : 2410817110014

|  |  |
| --- | --- |
| Menyetujui,  Asisten Praktikum  Muhammad Fauzan Ahsani  NIM. 2310817310009 | Mengetahui,  Dosen Penanggung Jawab Praktikum  Muti’a Maulida, S.Kom., M.TI.  NIP. 198810272019032013 |

# DAFTAR ISI

[LEMBAR PENGESAHAN i](#_Toc199324363)

[DAFTAR ISI ii](#_Toc199324364)

[DAFTAR TABEL iii](#_Toc199324365)

[DAFTAR GAMBAR iv](#_Toc199324366)

[SOAL 1 1](#_Toc199324367)

[A Output Program 2](#_Toc199324368)

[B Output Program 13](#_Toc199324369)

[C Pembahasan 20](#_Toc199324370)

[TAUTAN GITHUB 32](#_Toc199324371)

# DAFTAR TABEL

# DAFTAR GAMBAR

[Gambar 1, Tampilan Contoh Soal Menu Sorting 1](#_Toc199352903)

[Gambar 2 Tampilan Menu Sorting 13](#_Toc199352904)

[Gambar 3 Pilihan 1 Pada Menu Sorting 13](#_Toc199352905)

[Gambar 4 Masukkan Nama Pada Pilihan 1 13](#_Toc199352906)

[Gambar 5 Tampilan Hasil Dari Insertion Sort 14](#_Toc199352907)

[Gambar 6 Pilihan 2 Pada Menu Sorting 14](#_Toc199352908)

[Gambar 7 Masukkan Nama Pada Pilihan 2 14](#_Toc199352909)

[Gambar 8 Tampilan Hasil Dari Merge Sort 15](#_Toc199352910)

[Gambar 9 Pilihan 3 Pada Menu Sorting 15](#_Toc199352911)

[Gambar 10 Masukkan Nama Pada Pilihan 3 15](#_Toc199352912)

[Gambar 11 Tampilan Hasil Dari Shell Sort 16](#_Toc199352913)

[Gambar 12 Pilihan 4 Pada Menu Sorting 16](#_Toc199352914)

[Gambar 13 Masukkan ID Pada Pilihan 4 16](#_Toc199352915)

[Gambar 14 Tampilan Hasil Dari Bubble Sort 17](#_Toc199352916)

[Gambar 15 Pilihan 5 Pada Menu Sorting 17](#_Toc199352917)

[Gambar 16 Masukkan ID Pada Pilihan 5 17](#_Toc199352918)

[Gambar 17 Tampilan Hasil Dari Quick Sort 18](#_Toc199352919)

[Gambar 18 Pilihan 6 Pada Menu Sorting 18](#_Toc199352920)

[Gambar 19 Masukkan ID Pada Pilihan 6 18](#_Toc199352921)

[Gambar 20 Tampilan Hasil Dari Selection Sort 19](#_Toc199352922)

[Gambar 21 Pilihan 7 Pada Menu Sorting 19](#_Toc199352923)

[Gambar 22 Tampilan Program Selesai 19](#_Toc199352924)

[Gambar 23 Ilustrasi Insertion Sort 21](#_Toc199352925)

[Gambar 24 Ilustrasi Merge Sort 23](#_Toc199352926)

[Gambar 25 Ilustrasi Shell Sort 25](#_Toc199352927)

[Gambar 26 Ilustrasi Bubblu Sort 27](#_Toc199352928)

[Gambar 27 Ilustrasi Quick Sort 29](#_Toc199352929)

[Gambar 28 Ilustrasi Selection Sort 30](#_Toc199352930)

# SOAL 1

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)



Gambar 1, Tampilan Contoh Soal Menu Sorting

## Output Program

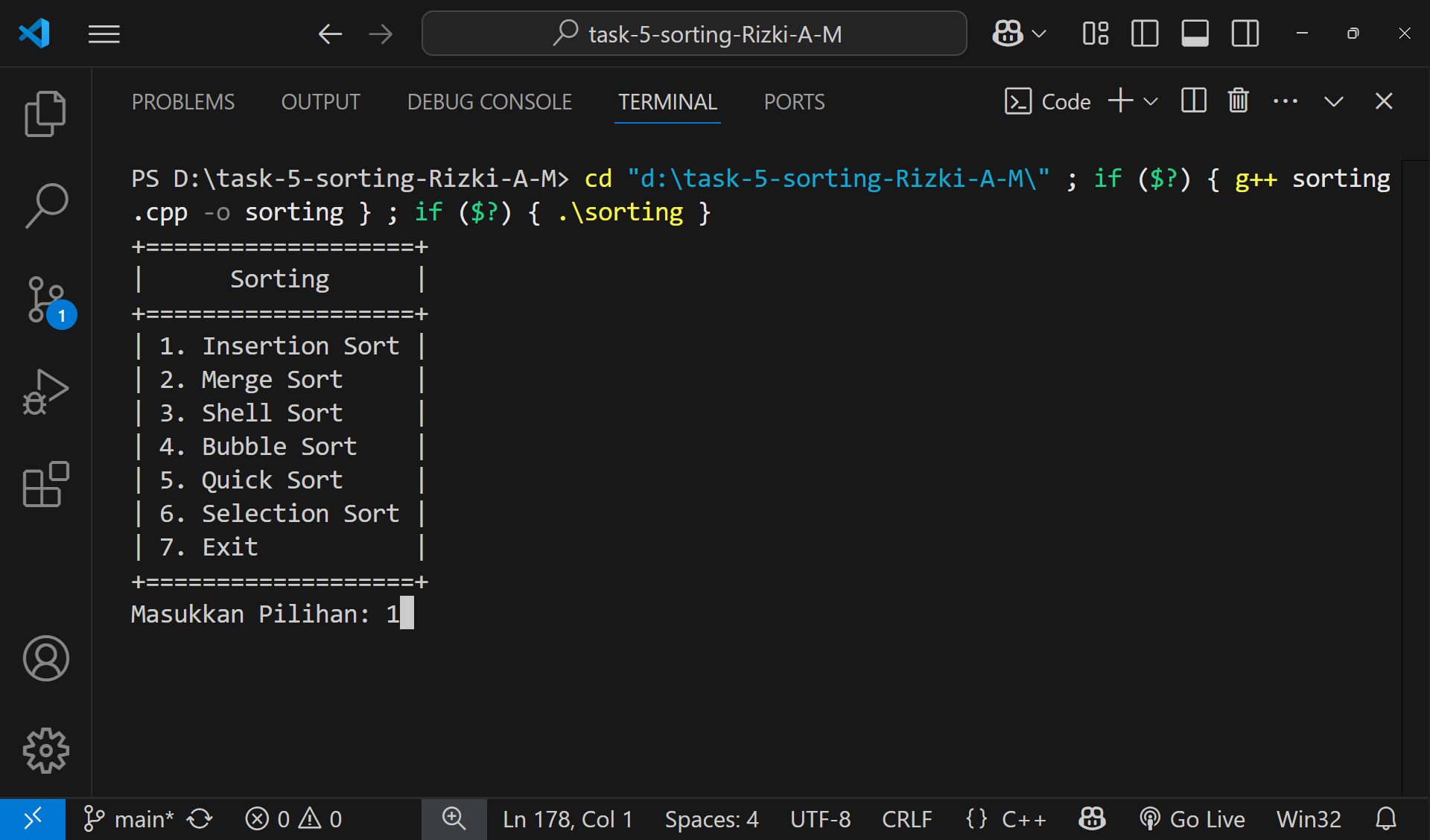
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220  221  222  223  224  225  226  227  228  229  230  231  232  233  234  235  236  237  238  239  240  241  242  243  244  245  246  247  248  249  250  251  252  253  254  255  256  257  258  259  260  261  262  263  264  265  266  267  268  269  270  271  272  273  274  275  276  277  278  279  280  281  282  283 | #include <iostream>  #include <functional>  #include <chrono>  #include <string>  #include <iomanip>  #include <conio.h>  using namespace std;  string name, id;  void timeSort(const function<void()>& sortFunc, const string& sortName)  {      auto start = chrono::high\_resolution\_clock::now();      sortFunc();      auto end = chrono::high\_resolution\_clock::now();      chrono::duration<double> duration = end - start;      cout << fixed << setprecision(10);      cout << sortName << " took " << duration.count() << " seconds\n";  }  void insertionSort(string &str)  {      for (int i = 1; i < str.size(); i++)      {          char key = str[i];          int j = i - 1;          while (j >= 0 && str[j] > key)          {              str[j + 1] = str[j];              j--;          }            str[j + 1] = key;      }  }  void merge(string &str, int left, int mid, int right)  {      int n1 = mid - left + 1;      int n2 = right - mid;      char \*tempL = new char[n1];      char \*tempR = new char[n2];      for (int i = 0; i < n1; i++) tempL[i] = str[left + i];      for (int j = 0; j < n2; j++) tempR[j] = str[mid + 1 + j];      int i = 0, j = 0, k = left;      while (i < n1 && j < n2)      {          if (tempL[i] <= tempR[j])          {              str[k] = tempL[i];              i++;          }          else          {              str[k] = tempR[j];              j++;          }          k++;      }      while (i < n1)      {          str[k] = tempL[i];          i++;          k++;      }      while (j < n2)      {          str[k] = tempR[j];          j++;          k++;      }      delete[] tempL;      delete[] tempR;  }  void mergeSort(string &str, int left, int right)  {      if (left < right)      {          int mid = left + (right - left) / 2;          mergeSort(str, left, mid);          mergeSort(str, mid + 1, right);          merge(str, left, mid, right);      }  }  void shellSort(string &str, int n)  {      for (int gap = n/2; gap > 0; gap /= 2)      {          for (int i = gap; i < n; i++)          {              int temp = str[i];              int j;              for (j = i; j >= gap && str[j - gap] > temp; j -= gap) str[j] = str[j - gap];              str[j] = temp;          }      }  }  void bubbleSort(string &str)  {      for (int i = 0; i < str.size() - 1; i++)      {          bool swapped = false;            for (int j = 0; j < str.size() - i - 1; j++)          {              if (str[j] > str[j + 1])              {                  swap(str[j], str[j + 1]);                  swapped = true;              }          }          if (!swapped) break;      }  }  int partition(string &str, int low, int high)  {      int pivot = str[high];      int i = (low - 1);      for (int j = low; j <= high - 1; j++)      {          if (str[j] <= pivot)          {              i++;              swap(str[i], str[j]);          }      }      swap(str[i + 1], str[high]);      return (i + 1);  }  void quickSort(string &str, int low, int high)  {      if (low < high)      {          int p\_idx = partition(str, low, high);          quickSort(str, low, p\_idx - 1);          quickSort(str, p\_idx + 1, high);      }  }  void selectionSort(string &str)  {      for (int i = 0; i < str.size() - 1; i++)      {          int minIndex = i;            for (int j = i + 1; j < str.size(); j++)          {              if (str[j] < str[minIndex])              {                  minIndex = j;              }          }          swap(str[i], str[minIndex]);      }  }  int main()  {      int ch;      string temp;      do      {          cout << "+===================+" << endl;          cout << "|      Sorting      |" << endl;          cout << "+===================+" << endl;          cout << "| 1. Insertion Sort |" << endl;          cout << "| 2. Merge Sort     |" << endl;          cout << "| 3. Shell Sort     |" << endl;          cout << "| 4. Bubble Sort    |" << endl;          cout << "| 5. Quick Sort     |" << endl;          cout << "| 6. Selection Sort |" << endl;          cout << "| 7. Exit           |" << endl;          cout << "+===================+" << endl;          cout << "Masukkan Pilihan: "; cin >> ch;            system("cls");          switch(ch)          {              case 1:                  cout << "Masukkan Nama: ";                  cin.ignore();                  getline(cin, name);                  system("cls");                  temp = name;                  cout << "Data Sebelum Diurutkan: " << temp << endl;                  timeSort([&]() {insertionSort(temp); }, "Insertion Sort");                  cout << "Data Setelah Diurutkan: " << temp << endl;                  break;              case 2:                  cout << "Masukkan Nama: ";                  cin.ignore();                  getline(cin, name);                  system("cls");                  temp = name;                  cout << "Data Sebelum Diurutkan: " << temp << endl;                  timeSort([&]() {mergeSort(temp, 0, temp.size() - 1); }, "Merge Sort");                  cout << "Data Setelah Diurutkan: " << temp << endl;                  break;              case 3:                  cout << "Masukkan Nama: ";                  cin.ignore();                  getline(cin, name);                  system("cls");                  temp = name;                  cout << "Data Sebelum Diurutkan: " << temp << endl;                  timeSort([&]() {shellSort(temp, temp.size()); }, "Shell Sort");                  cout << "Data Setelah Diurutkan: " << temp << endl;                  break;              case 4:                  cout << "Masukkan ID: "; cin >> id;                  system("cls");                  temp = id;                  cout << "Data Sebelum Diurutkan: " << temp << endl;                  timeSort([&]() {bubbleSort(temp); }, "Bubble Sort");                  cout << "Data Setelah Diurutkan: " << temp << endl;                  break;              case 5:                  cout << "Masukkan ID: "; cin >> id;                  system("cls");                  temp = id;                  cout << "Data Sebelum Diurutkan: " << temp << endl;                  timeSort([&]() {quickSort(temp, 0, temp.size() - 1); }, "Quick Sort");                  cout << "Data Setelah Diurutkan: " << temp << endl;                  break;              case 6:                  cout << "Masukkan ID: "; cin >> id;                  system("cls");                  temp = id;                  cout << "Data Sebelum Diurutkan: " << temp << endl;                  timeSort([&]() {selectionSort(temp); }, "Selection Sort");                  cout << "Data Setelah Diurutkan: " << temp << endl;                  break;              case 7:                  cout << "Terima Kasih" << endl;                  cout << "This Program Was Made by Rizki Adhitiya Maulana (2410817110014)" << endl;                  break;              default:                  cout << "Opsi Tidak Valid. Silahkan Coba Lagi." << endl;          }          cout << "Press any key to continue...";          getch();          system("cls");      }      while (ch != 7);        return 0;  } |

## Output Program

Sebuah gambar berisi teks, cuplikan layar, software, Software multimedia

Konten yang dihasilkan AI mungkin salah.

Gambar 2 Tampilan Menu Sorting



Gambar 3 Pilihan 1 Pada Menu Sorting

Sebuah gambar berisi teks, cuplikan layar, software, Software multimedia

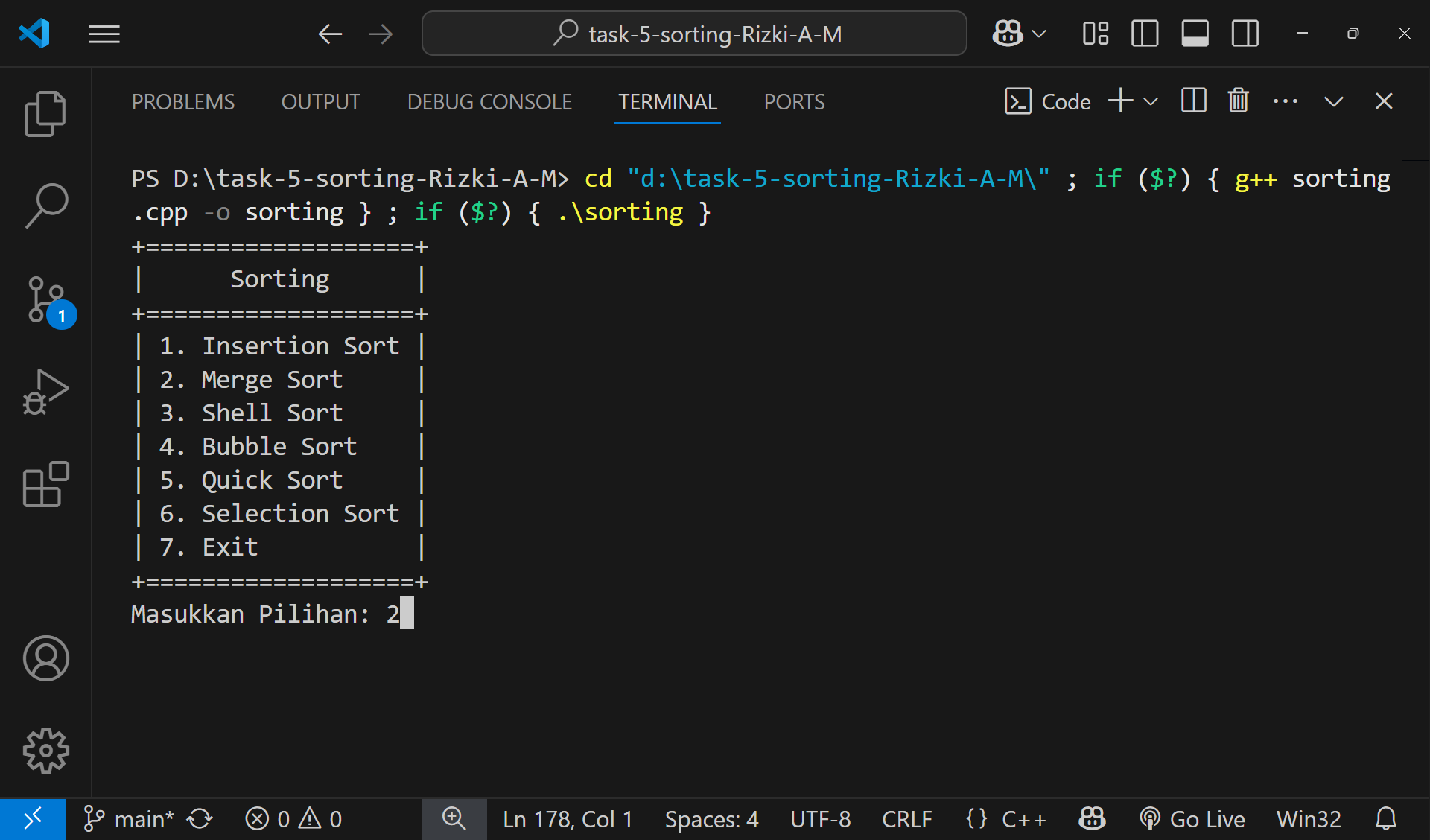
Konten yang dihasilkan AI mungkin salah.

Gambar 4 Masukkan Nama Pada Pilihan 1

Sebuah gambar berisi teks, cuplikan layar, software, Software multimedia

Konten yang dihasilkan AI mungkin salah.

Gambar 5 Tampilan Hasil Dari Insertion Sort



Gambar 6 Pilihan 2 Pada Menu Sorting

Sebuah gambar berisi teks, cuplikan layar, software, Software multimedia

Konten yang dihasilkan AI mungkin salah.

Gambar 7 Masukkan Nama Pada Pilihan 2

Sebuah gambar berisi teks, cuplikan layar, software, Software multimedia

Konten yang dihasilkan AI mungkin salah.

Gambar 8 Tampilan Hasil Dari Merge Sort

Sebuah gambar berisi teks, cuplikan layar, software, Software multimedia

Konten yang dihasilkan AI mungkin salah.

Gambar 9 Pilihan 3 Pada Menu Sorting

Sebuah gambar berisi teks, cuplikan layar, software, Software multimedia

Konten yang dihasilkan AI mungkin salah.

Gambar 10 Masukkan Nama Pada Pilihan 3

Sebuah gambar berisi teks, cuplikan layar, software, Software multimedia

Konten yang dihasilkan AI mungkin salah.

Gambar 11 Tampilan Hasil Dari Shell Sort

Sebuah gambar berisi teks, cuplikan layar, software, Software multimedia

Konten yang dihasilkan AI mungkin salah.

Gambar 12 Pilihan 4 Pada Menu Sorting

Sebuah gambar berisi teks, cuplikan layar, software, Software multimedia

Konten yang dihasilkan AI mungkin salah.

Gambar 13 Masukkan ID Pada Pilihan 4

Sebuah gambar berisi teks, cuplikan layar, software, Software multimedia

Konten yang dihasilkan AI mungkin salah.

Gambar 14 Tampilan Hasil Dari Bubble Sort

Sebuah gambar berisi teks, cuplikan layar, software, Software multimedia

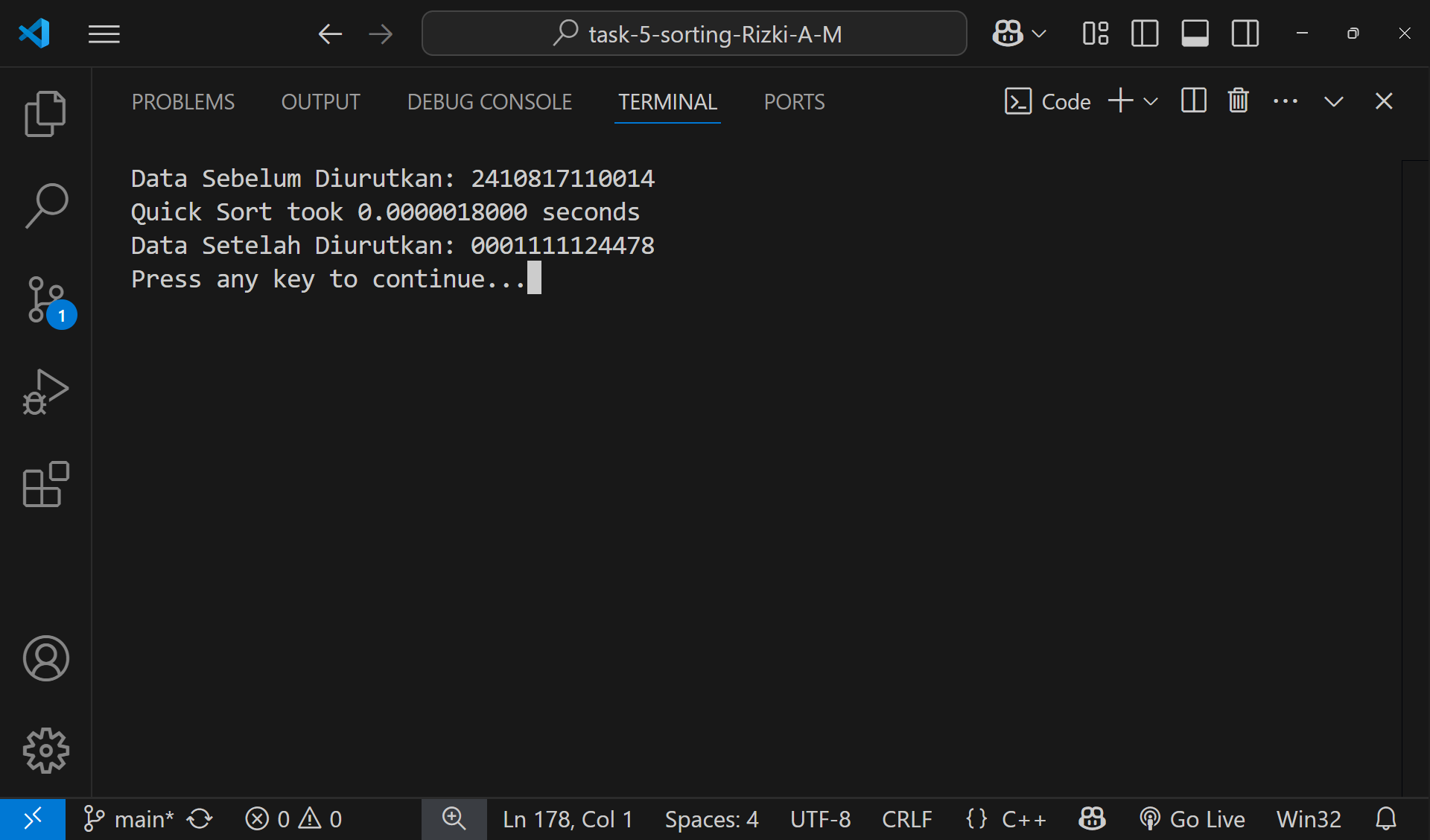
Konten yang dihasilkan AI mungkin salah.

Gambar 15 Pilihan 5 Pada Menu Sorting

Sebuah gambar berisi teks, cuplikan layar, software, Software multimedia

Konten yang dihasilkan AI mungkin salah.

Gambar 16 Masukkan ID Pada Pilihan 5



Gambar 17 Tampilan Hasil Dari Quick Sort

Sebuah gambar berisi teks, cuplikan layar, software, Software multimedia

Konten yang dihasilkan AI mungkin salah.

Gambar 18 Pilihan 6 Pada Menu Sorting

Sebuah gambar berisi teks, cuplikan layar, software, Software multimedia

Konten yang dihasilkan AI mungkin salah.

Gambar 19 Masukkan ID Pada Pilihan 6

Sebuah gambar berisi teks, cuplikan layar, software, Software multimedia

Konten yang dihasilkan AI mungkin salah.

Gambar 20 Tampilan Hasil Dari Selection Sort

Sebuah gambar berisi teks, cuplikan layar, software, Software multimedia

Konten yang dihasilkan AI mungkin salah.

Gambar 21 Pilihan 7 Pada Menu Sorting

Sebuah gambar berisi teks, cuplikan layar, software, Software multimedia

Konten yang dihasilkan AI mungkin salah.

Gambar 22 Tampilan Program Selesai

## Pembahasan

* **Alur Program**

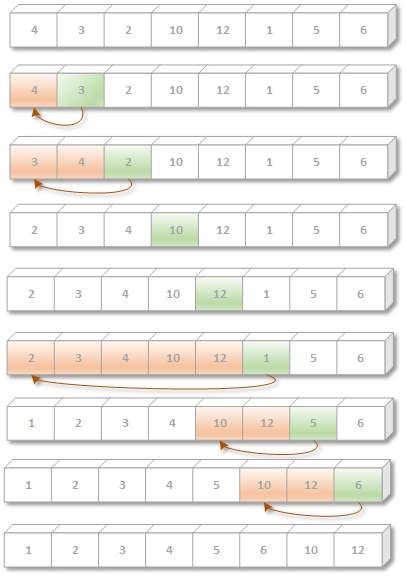
Ketika program dimulai, akan muncul tampilan menu dengan nama “***Sorting***” yang di dalamnya terdapat tujuh opsi, diantaranya ada opsi pertama untuk “***Insetion Sort***”, opsi kedua “***Merge Sort***”, opsi ketiga “***Shell Sort***”, opsi keempat “***Bubble Sort***”, opsi kelima “***Quick Sort***”, opsi keenam “***Selection Sort***” dan opsi ketujuh “***Exit***”. Program yang ada akan selalu berjalan atau melakukan perulangan karena adanya struktur “***do-while***”, yang akan terus menampilkan menu sorting sampai pengguna memilih opsi “***Exit***”. Kemudian, setiap selesai menentukan pilihan atau melakukan input untuk menu apa yang akan diakses, fungsi atau perintah “***CLS***” akan selalu dijalankan untuk membuat tampilan dari program terlihat tidak menumpuk dan rapi.

Struktur “***Switch-case***” digunakan untuk semua pilihan yang ada, mulai dari **case 1 sampai case 3**, pengguna akan diminta untuk memasukkan atau mengimput “***nama***”. Sedangkan, **case 4 sampai case 6** pengguna akan diminta untuk memasukkan atau mengimput “***id***”. Kecuali **case 7** yang akan menampilkan ucapan terima kasih dan nama serta nim dari pembuat program, dan **case default** yang akan memberitahukan bahwa opsi tidak valid karena pengguna memasukan pilihan di luar dari tujuh opsi yang ada. Setelah selesai melakukan input baik itu berupa “***nama***” atau “***nim***”, program akan menampilkan penampakan string sebelum dan sesudah disortir. Pada **case 1**, program akan menjalankan fungsi “***Insertion Sort***” dan menampilkan durasi eksekusinya, **case 2** menjalankan fungsi “***Merge Sort***” beserta durasinya, **case 3** menjalankan fungsi “***Shell Sort***” dengan durasinya, **case 4** menjalankan “***Bubble Sort***” disertai durasi, **case 5** menjalankan “***Quick Sort***” dengan waktu eksekusinya, dan **case 6** menjalankan “***Selection Sort***” serta menampilkan durasinya.

Setelah fungsi dari setiap pilihan dari program menyelesaikan semua tugas dan memprintkan semua hasil prosesnya, program akan menunggu input tombol apa saja untuk melanjutkan, lalu membersihkan layar dan menampilkan tampilan menu “***Sorting***” kembali.

* **Insertion Sort**

Insertion Sort digunakan untuk melakukan sebuah pengurutan data, yang mana dengan cara membandingan elemen yang ada pada array. Setiap elemen dalam array akan dibandingkan dengan elemen sebelumnya secara berurutan hingga semua elemen menemukan posisinya masing-masing. Proses pengurutannya di mulai dari elemen kedua, kemudian elemen kedua akan dibandingkan dengan elemen sebelumnya. Apabila elemen kedua lebih kecil dari elemen sebelumnya, maka elemen kedua akan digeser ke kiri atau disisipkan sebelum elemen yang lebih besar dari elemen kedua. Namun, apabila elemen kedua lebih besar dari elemen sebelumnya, maka tidak akan ada pergeseran posisi. Proses ini akan terus diulangi hingga elemen ketiga, keempat dan yang terakhir terurut.



Gambar 23 Ilustrasi Insertion Sort

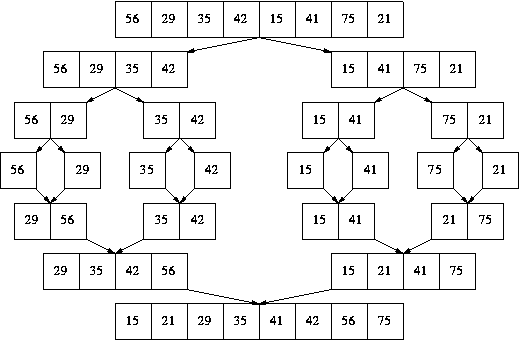
Dalam pembahasan fungsinya, void insertionSort(string &str) merupakan implementasi Insertion Sort yang mengurutkan karakter-karakter dalam sebuah string secara langsung karena menggunakan pass by reference. Proses pengurutan dimulai dengan loop for dari indeks i = 1 (karakter kedua), di mana setiap karakter pada str[i] disimpan sebagai key yang akan disisipkan ke bagian string di sebelah kirinya yang sudah terurut. Variabel j diinisialisasi dengan i - 1 untuk melacak posisi di bagian yang sudah terurut. Selanjutnya, loop while akan menggeser karakter str[j] satu posisi ke kanan (str[j + 1] = str[j]) selama j masih valid (j >= 0) dan str[j] lebih besar dari key, sambil terus mengurangi j (j--) untuk mencari posisi yang tepat. Setelah loop while selesai, key ditempatkan pada posisi str[j + 1], menyelesaikan proses penyisipan satu karakter dan secara bertahap mengurutkan seluruh string.

Kompleksitas waktu dari Insertion Sort tergantung pada kondisi awal data yang akan diurutkan. Pada kasus terbaik (best case) kompleksitas waktu Insertion Sort adalah ***O(n)***, karena data yang ada sudah terurut, algoritma yang ada hanya akan melakukan perbandingan antar elemen atau pengecekan. Pada kasus rata-rata (average case) dan kasus terburuk (worst case) kompleksitas waktunya adalah ***O(n²)***, karena saat data terurut acak atau terbalik, algoritma yang ada harus membandingkan dan menggeser hampir semua elemen. Sementara itu, kompleksitas ruang dari Insertion Sort adalah ***O(1)*** karena algoritma Insertion Sort tidak memerlukan struktur data tambahan.

Jadi kesimpulannya, Insertion Sort ini sangat cocok digunakan untuk mengurutkan data jumlah yang kecil atau hampir terurut. Kemudian juga hemat ruang penyimpanan dalam penggunaannya karena tidak perlu tambahan struktur data dalam prosesnya, hanya beberapa variabel bantu sebagai penyimpanan sementara.

* **Merge Sort**

Merge Sort digunakan untuk melakukan sebuah pengurutan data, yang mana menggunakan pendekatan divide and conquer dalam prosesnya. Cara kerja pengurutannya di mulai dengan membagi kumpulan data dalam satu array menjadi dua bagian secara terus menerus hingga menjadi bagian paling kecil yang berisi satu elemen. Setelah proses pembagian kumpulan data telah selesai, bagian paling kecil yang ada akan digabung kembali. Dalam proses penggabungannya akan selalu melibatkan dua bagian yang telah dibagi sebelumnya, karena ketika digabungkan elemen yang ada di dua bagian tersebut akan dibandingkan dan diurutkan dari yang terkecil disebelah kiri hingga yang terbesar di sebelah kanan. Proses penggabungan akan terus dilakukan secara bertahap dengan memperhatikan urutan elemen agar tetap terurut hingga seluruh elemen yang dibagi sebelumnya kembali menjadi satu kesatuan.



Gambar 24 Ilustrasi Merge Sort

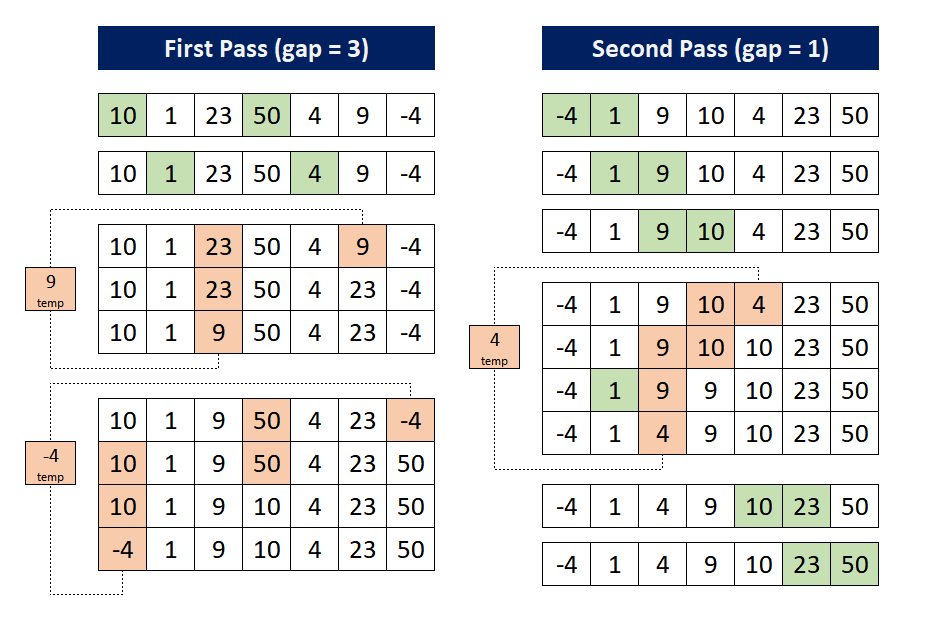
Dalam pembahasan fungsinya, void mergeSort(string &str, int left, int right) adalah bagian yang bekerja secara rekursif untuk mengurutkan string. Prosesnya diawali dengan kondisi dasar if (left < right), yang akan terus membagi string menjadi bagian-bagian yang lebih kecil hingga hanya tersisa satu karakter (yang otomatis terurut), menghentikan rekursi. Kemudian, dua panggilan rekursif dilakukan: satu untuk mengurutkan paruh kiri (mergeSort(str, left, mid)) dan satu untuk mengurutkan paruh kanan (mergeSort(str, mid + 1, right)).Selanjutnya dilajutkan dengan fungsi void merge(string &str, int left, int mid, int right) untuk menggabungkan dua sub array yang sudah terurut (yang direpresentasikan oleh bagian-bagian dari str dari left hingga mid, dan dari mid + 1 hingga right) menjadi satu bagian yang terurut dalam string asli. Fungsi ini pertama menghitung ukuran kedua sub-array (n1 dan n2), lalu mengalokasikan dua array sementara (tempL dan tempR) di heap untuk menyalin elemen-elemen dari sub-array yang bersangkutan. Proses penggabungan dilakukan dengan loop while utama yang membandingkan elemen-elemen dari tempL[i] dan tempR[j], menyalin karakter yang lebih kecil ke str[k] dan menginkrementasi indeks yang relevan (i, j, dan k). Setelah salah satu array sementara habis, dua loop while terpisah akan menyalin sisa elemen dari array sementara yang belum habis ke str. Terakhir, memori yang dialokasikan secara dinamis dibebaskan menggunakan delete[] untuk mencegah memory leak, memastikan operasi penggabungan berjalan efisien dan bersih.

Kompleksitas waktu dari Merge Sort tidak tergantung pada kondisi awal data yang akan diurutkan. Artinya baik dalam kasus baik, kasus rata-rata, ataupun kasus yang buruk kompleksitas waktu Merge Sort adalah ***O(n log n)****,* karena ada setiap tahap penggabungan, seluruh elemen (n elemen) harus diproses. Sementara itu, kompleksitas ruang dari Merge Sort adalah ***O(n)*** karena algoritma Merge Sort memerlukan ruang tambahan sebagai tempat penyimpanan sementara selama proses penggabungan.

Jadi kesimpulannya, Merge Sort ini cocok digunakan untuk mengurutkan data dalam jumlah yang besar karena hasil dari penggurutan datanya yang akurat dan waktu eksekusi data yang konsisten di semua kasus, namun membutuhkan ruang tambahan untuk proses di dalamnya untuk menyimpan array semestara saat prosesnya.

* **Shell Sort**

Shell Sort merupakan pengembangan dari Insertion Sort, yang mana Shell Sort digunakan untuk melakukan sebuah pengurutan data dengan cara membandingkan dan menukar elemen-elemen yang memiliki jarak terlebih dahulu. Jarak antar elemen ini disebut dengan *“gap”*. Cara kerja pengurutannya di mulai dengan membandingkan data yang letaknya jauh terlebih dahulu, lalu jarak yang ada tersebut dikurangi sedikit demi sedikit secara bertahap hingga 1. Apabila elemen gap yang ada berupa 1, maka algoritma yang ada akan melakukan Insertion Sort.



Gambar 25 Ilustrasi Shell Sort

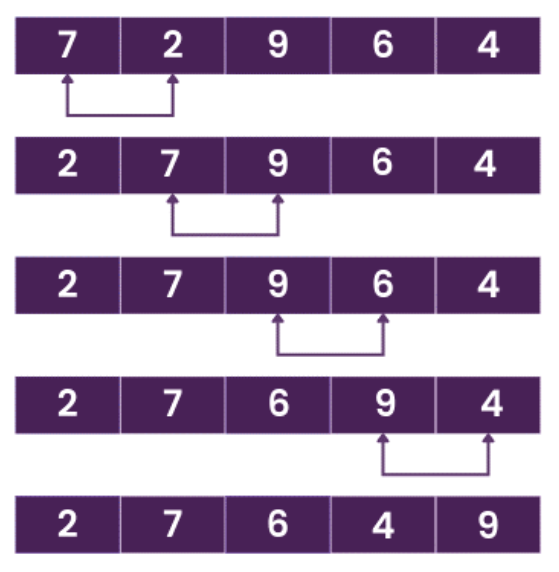
Dalam pembahasan fungsinya, void shellSort(string &str, int n) mengimplementasikan algoritma Shell Sort, sebuah peningkatan dari Insertion Sort yang lebih efisien, terutama untuk data berukuran sedang. Algoritma ini bekerja dengan pendekatan "gap" atau jarak, dimulai dengan loop for terluar yang mengatur ukuran gap awal (biasanya n/2) dan secara bertahap mengurangi gap menjadi setengahnya hingga mencapai 1. Untuk setiap nilai gap, loop for tengah akan mengiterasi string dari posisi gap hingga n, mengambil setiap karakter str[i] sebagai temp. Kemudian, loop for terdalam yang mirip Insertion Sort akan membandingkan temp dengan elemen-elemen yang berada gap posisi di belakangnya (str[j - gap]). Jika str[j - gap] lebih besar dari temp, maka str[j - gap] digeser ke posisi str[j], dan proses ini berlanjut mundur dengan mengurangi j sebesar gap hingga posisi yang tepat ditemukan. Akhirnya, temp disisipkan pada posisi str[j], secara efektif mengurutkan sub-list yang berjarak gap dan secara bertahap mengurutkan seluruh string ketika gap mengecil hingga 1.

Kompleksitas waktu dari Shell Sort bergantung pada kondisi awal data yang akan diurutkan dan gap yang ada. Pada kasus terbaik (best case) kompleksitas waktu Shell Sort adalah ***Ω(n log(n))***, karena data yang ada sudah terurut, algoritma yang ada hanya akan melakukan perbandingan antar elemen atau pengecekan. Pada kasus rata-rata (average case) kompleksitas waktunya adalah ***O(n\*log n)~O(n 1.25)***, karena data yang ada sudah hampi terurut hanya perlu melakukan sedikit pergeseran diakhir dengan Insertion Sort. Kasus terburuk (worst case) kompleksitas waktunya adalah ***O(n²)***, karena saat data terurut acak atau terbalik, algoritma yang ada harus membandingkan dan memindahkan dan menggeser hampir semua elemen. Sementara itu, kompleksitas ruang dari Shell Sort adalah ***O(1)*** karena algoritma Insertion Sort tidak memerlukan struktur data tambahan.

Jadi kesimpulan, Shell Sort ini cocok digunakan untuk mengurutkan data dalam jumlah yang sedang atau hampir terurut. Kemudian juga hemat ruang penyimpanan dalam penggunaannya karena tidak perlu tambahan struktur data dalam prosesnya, hanya beberapa variabel bantu sebagai penyimpanan sementara.

* **Bubble Sort**

Bubble Sort digunakan untuk melakukan sebuah pengurutan data, yang mana dengan mendorong elemen terbesar yang ada di dalam kumpulan data ke kanan secara bertahap. Cara kerja penggurutannya dimulai dengan membandingkan elemen pertama dan elemen kedua. Apabila elemen pertama lebih besar dari elemen kedua maka posisi dari keduanya akan di tukar, elemen yang lebih besar akan berpindah ke kanan. Kemudian, perbandingan berlanjut antara elemen kedua dan ketiga, dan begitu seterusnya hingga elemen terakhir. Setelah satu lintasan selesai, elemen terbesar akan berada di posisi paling kanan. Proses ini diulang kembali untuk elemen-elemen yang tersisa, dengan setiap lintasan mengurangi jumlah elemen yang perlu diperiksa karena elemen-elemen terbesar sudah berada di posisi yang benar. Pengurutan akan berhenti ketika dalam satu lintasan tidak ada lagi pertukaran, yang menandakan bahwa seluruh data sudah dalam keadaan terurut.



Gambar 26 Ilustrasi Bubblu Sort

Dalam pembahasan fungsinya, void bubbleSort(string &str) bekerja melalui dua loop for bersarang, pada loop terluar (i) menentukan berapa banyak elemen terbesar yang sudah dipindahkan ke posisi akhirnya di bagian kanan string, sementara pada loop dalam (j) melakukan perbandingan berpasangan antara elemen yang berdekatan (str[j] dan str[j+1]). Jika str[j] lebih besar dari str[j+1], kedua karakter tersebut akan ditukar posisinya menggunakan swap(), dan sebuah flag swapped akan diatur ke true. Apabila setelah satu iterasi penuh dari loop dalam tidak ada pertukaran yang terjadi (artinya swapped tetap false), maka string dianggap sudah terurut sepenuhnya, dan loop terluar akan dihentikan lebih awal untuk meningkatkan efisiensi.

Kompleksitas waktu dari Bubble Sort tergantung pada kondisi awal data yang akan diurutkan. Pada kasus terbaik (best case) kompleksitas waktu Bubble Sort adalah ***O(n)***, karena data yang ada sudah terurut, algoritma yang ada hanya akan melakukan perbandingan antar elemen atau pengecekan. Pada kasus rata-rata (average case) dan kasus terburuk (worst case) kompleksitas waktunya adalah ***O(n²)***, karena saat data terurut acak atau terbalik, algoritma yang ada harus membandingkan dan menggeser hampir semua elemen. Sementara itu, kompleksitas ruang dari Insertion Sort adalah ***O(1)*** karena algoritma Insertion Sort tidak memerlukan struktur data tambahan.

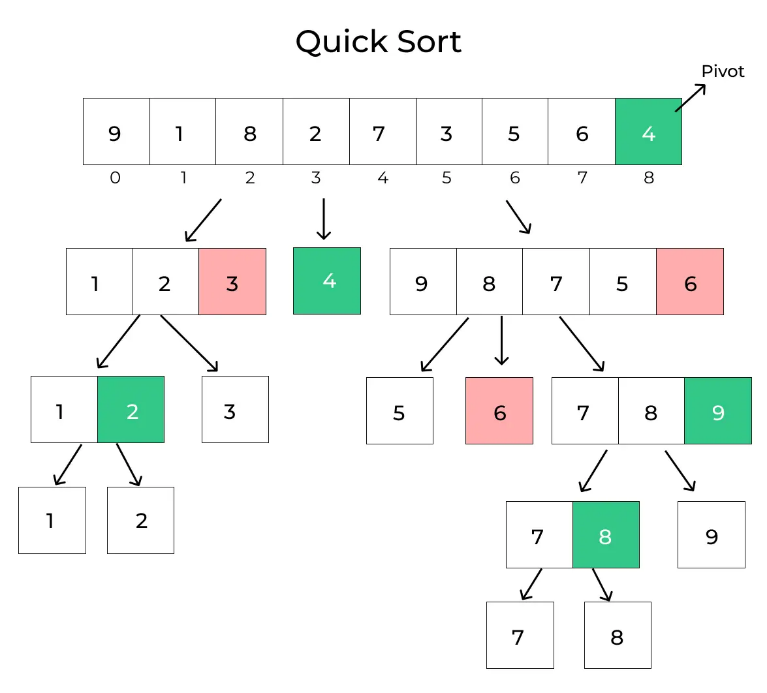
Jadi kesimpulannya, Insertion Sort ini sangat cocok digunakan untuk mengurutkan data jumlah yang kecil atau hampir terurut. Kemudian juga hemat ruang penyimpanan dalam penggunaannya karena tidak perlu tambahan struktur data dalam prosesnya, hanya beberapa variabel bantu sebagai penyimpanan sementara.

* **Quick Sort**

Quick Sort digunakan untuk melakukan sebuah pengurutan data, yang mana menggunakan pendekatan divide and conquer dalam proses. Cara kerja penggurutannya dimulai dengan memilih satu elemen sebagai pivot, biasanya elemen terakhir dalam kumpulan data. Selanjutnya, data dibagi menjadi dua bagian dimana elemen yang lebih kecil atau sama dengan pivot ditempatkan di sebelah kiri, sedangkan elemen yang lebih besar ditempatkan di sebelah kanan. Pivot kemudian diposisikan di tempat yang tepat dalam urutan akhir.

Proses pemisahan ini dilakukan oleh fungsi partition(), yang membandingkan setiap elemen dengan pivot. Bila ditemukan elemen yang lebih kecil atau sama dengan pivot, maka elemen itu akan ditukar ke posisi yang lebih kiri. Setelah semua elemen diperiksa, pivot akan ditukar ke posisi tengah yang sesuai, dan posisi ini akan menjadi batas pembagian data selanjutnya.

Setelah itu, fungsi quickSort() akan memanggil dirinya sendiri secara **rekursif** untuk mengurutkan bagian kiri dan kanan dari pivot. Jika masih ada bagian yang belum terurut, maka proses ini akan diulangi dengan memilih pivot baru dan membagi ulang datanya. Proses ini terus berlangsung hingga seluruh elemen berada di posisi yang tepat dan data menjadi terurut sempurna.



Gambar 27 Ilustrasi Quick Sort

Kompleksitas waktu dari Quick Sort tergantung pada pemilihan pivot dan kondisi awal data yang akan diurutkan. Pada kasus terbaik (best case) kompleksitas waktunya adalah ***Ω(n log n)***, karena pembagian data menjadi dua bagian yang relatif seimbang. Pada kasus rata-rata (average case) kompleksitas waktu adalah ***θ(n log n)***, karena pembagian data biasanya cukup merata dalam banyak kasus. Pada kasus terburuk (worst case) kompleksitas waktunya menjadi ***O(n²)***, yaitu ketika pivot yang dipilih selalu menjadi elemen terkecil atau terbesar (misalnya saat data sudah terurut atau terbalik). Sementara itu, kompleksitas ruang dari Quick Sort pada kasus terbaik dan rata-rata adalah ***O(log n)*** karena membagi dua bagian dengan seimbang atau hampir seimbang. Sedangkan, untuk kompleksitas ruang dari Quick Sort pada kasus terburuk adalah ***O(n)***karena pembagiannya yang tidak seimbang.

Jadi kesimpulannya, Quick Sort ini sangat cocok digunakan untuk mengurutkan data jumlah yang besar karena hasil dari penggurutan datanya yang akurat dan waktu eksekusi rata-ratanya yang efisien. Kemudian, penggunaan ruang penyimpanannya tergolong kecil karena tidak memerlukan struktur data tambahan.

* **Selection Sort**

Selection Sort digunakan untuk melakukan sebuah pengurutan data, yang mana akan memilih elemen terkecil dari bagian data yang belum terurut, lalu menukarnya dengan elemen di posisi paling awal dari bagian tersebut. Proses ini dilakukan berulang-ulang hingga semua elemen yang ada berada diposisi yang sudah tepat atau terurut. Cara kerja penggurutannya dimulai dengan mencari elemen terkecil dari seluruh data, lalu menempatkannya di posisi pertama. Selanjutnya, algoritma melanjutkan pencarian elemen terkecil berikutnya dari data yang tersisa dan menukarnya ke posisi kedua, dan seterusnya. Dengan pendekatan ini, data secara bertahap akan terurut dari posisi awal hingga akhir melalui proses seleksi dan pertukaran.

Sebuah gambar berisi glockenspiel

Konten yang dihasilkan AI mungkin salah.

Gambar 28 Ilustrasi Selection Sort

Dalam pembahasan fungsinya, void selectionSort(string &str) mengimplementasikan algoritma Selection Sort, yang bekerja dengan cara berulang kali mencari elemen terkecil dari bagian string yang belum terurut dan menempatkannya pada posisi yang benar. Ini dilakukan melalui loop for terluar yang mengiterasi dari awal string hingga satu posisi sebelum akhir. Dalam setiap iterasi, sebuah variabel minIndex diinisialisasi dengan indeks i saat ini, lalu loop for dalam akan mencari di sisa bagian string (dari j = i + 1 hingga akhir) untuk menemukan indeks (j) dari karakter yang paling kecil. Setelah loop dalam selesai, elemen pada str[i] dan elemen pada str[minIndex] (yang merupakan elemen terkecil yang ditemukan) akan ditukar posisinya menggunakan swap(), sehingga elemen terkecil tersebut "dipilih" dan ditempatkan pada posisi yang benar di awal bagian yang belum terurut. Proses ini terus berlanjut hingga seluruh string terurut.

Kompleksitas waktu dari Selection Sort tidak bergantung pada kondisi awal data, baik data tersebut sudah terurut, hampir terurut, maupun acak sepenuhnya. Pada kasus terbaik (best case), kasus rata-rata (average case), dan kasus terburuk (worst case), kompleksitas waktunya tetap ***O(n²)***, karena algoritma harus mencari elemen terkecil dalam bagian yang belum terurut di setiap langkahnya, tanpa memperhatikan apakah data sudah dalam urutan tertentu. Sementara itu, kompleksitas ruang dari Selection Sort adalah ***O(1)***, karena tidak perlu tambahan struktur data dalam prosesnya

# TAUTAN GITHUB

https://github.com/Rizki-A-M/Rizki-A-M-PRAKTIKUM\_ALGORITMA\_DAN\_STRUKTUR\_DATA.git