**LAPORAN TUGAS STRUKTUR DATA**

**APLIKASI PENERAPAN BERBAGAI METODE *SORTING***



Dosen Pembimbing:

Dr. Ricky Eka Putra, S.Kom., M.Kom.

Disusun oleh:

Cornelius Louis Nathan 23051204085

**PROGRAM STUDI S1 TEKNIK INFORMATIKA**

**UNIVERSITAS NEGERI SURABAYA**

**2024**

# KATA PENGANTAR

Puji syukur saya haturkan kepada Tuhan Yang Maha Esa sehingga atas rahmat-Nya. Saya dapat menyelesaikan laporan tugas struktur data yang berjudul “Aplikasi Penerapan Berbagai Metode *Sorting*” sebagai tugas struktur data tepat pada waktunya.

Adapun tujuan penulisan laporan ini adalah untuk mengetahui apa saja penerapan metode pengurutan dan mengetahui mana penerapan metode pengurutan yang paling efektif. Pembuatan laporan ini sekaligus menjadi tugas yang mengisi nilai dari Mata Kuliah Struktur Data.

Penyusunan laporan ini tidak lepas dari bantuan dan dukungan. Sehingga, kami menyampaikan terima kasih kepada pihak yang telah mendukung pembuatan laporan ini. Terutama kepada Bapak Ricky Eka Putra, S.Kom., M.Kom., selaku Dosen Pembimbing Mata Kuliah Struktur Data, atas petunjuk, didikan, dan arahan yang sangat membantu dalam penyusunan laporan ini.

Kami menyadari bahwasanya penulisan laporan ini jauh dari kata sempurna, sehingga kami menerima saran dan kritik yang membangun dari pembaca guna untuk perbaikan dan perkembangan di masa mendatang. Semoga penyusunan laporan ini juga berguna bagi para pembaca dalam memahami apa saja penerapan metode pengurutan dan mengetahui mana penerapan metode pengurutan yang paling efektif.

Surabaya, 22 April 2024

Penulis

# DAFTAR ISI

[KATA PENGANTAR ii](#_Toc165108876)

[DAFTAR ISI 3](#_Toc165108877)

[BAB I 4](#_Toc165108878)

[PENDAHULUAN 4](#_Toc165108879)

[1.1 Latar Belakang 4](#_Toc165108880)

[1.2 Rumusan Masalah 4](#_Toc165108881)

[1.3 Tujuan 4](#_Toc165108882)

[BAB II 5](#_Toc165108883)

[KAJIAN PUSTAKA 5](#_Toc165108884)

[2.1 Pengertian dan Contoh dari Algoritma Pengurutan Data 5](#_Toc165108885)

[2.2 Pengimplementasian Algoritma Pengurutan Data dalam Kasus Dunia Nyata 5](#_Toc165108886)

[2.3 Cara Kerja Algoritma Pengurutan Data 11](#_Toc165108887)

[2.4 Algoritma Pengurutan Data yang Terbaik untuk Diaplikasikan 22](#_Toc165108888)

# BAB I

# PENDAHULUAN

## 1.1 Latar Belakang

Dalam penggunaan komputer, pengolahan dan struktur data sangatlah penting seiring dengan berkembangnya jumlah dan kompleksitas data yang terus berkembang, salah satu aspek penting dalam pengolahan data adalah *sorting* (pengurutan data). Dimana elemen atau data diatur sedemikian rupa sehingga memungkinkan akses yang lebih efisien.

Algoritma pengurutan data merupakan aspek penting yang tidak boleh diabaikan. Basis data, *searching* (pencarian data), dan analisis data sangat membutuhkan *sorting* untuk meningkatkan efisiensi dan responsifitas program. Pemilihan algoritma pengurutan yang tepat juga akan mendukung angka efisiensi dan responsifitas semakin naik.

Dalam kasus ini, andaikan seorang remaja penggemar *game arcade* bernama Julian mengoleksi banyak kartu dari permainan *game arcade* bernama RoboFight. Setiap kartu dari RoboFight memiliki kode unik, gambar robot, *power* dan *hitpoint (HP)* robot yang masing-masing berbeda satu dengan yang lain. Karena banyaknya kartu yang dimiliki oleh Julian, Julian sempat kewalahan untuk mengurutkan kartu-kartunya. Julian berinisiatif untuk membuat aplikasi yang mampu mengurutkan kartu-kartu tersebut berdasarkan kode unik, *power*, dan *HP* robot tersebut.

## 1.2 Rumusan Masalah

1. Apa pengertian dari algoritma pengurutan data dan contoh-contohnya?
2. Bagaimana pengimplementasian algoritma pengurutan data dalam kasus dunia nyata?
3. Bagaimana cara kerja algoritma pengurutan data tersebut?
4. Manakah algoritma pengurutan data yang terbaik untuk diaplikasikan?

## 1.3 Tujuan

1. Memahami pengertian dari algoritma pengurutan data dan contoh-contohnya.
2. Memahami pengimplementasian algoritma pengurutan data dalam kasus dunia nyata.
3. Memahami cara kerja algoritma pengurutan data tersebut.
4. Memahami algoritma pengurutan data yang terbaik untuk diaplikasikan

# BAB II

# KAJIAN PUSTAKA

## 2.1 Pengertian dan Contoh dari Algoritma Pengurutan Data

Pengurutan data adalah proses pen gaturan kumpulan data menjadi urutan tertentu sesuai dengan kebutuhan pengguna, seperti urutan numerik (dari urutan terkecil sampai terbesar), urutan alfabetis, urutan tanggal dan tahun, atau urutan berdasarkan kriteria lainnya. Tujuannya adalah memudahkan data untuk dicari, diproses ataupun dimanipulasi oleh pengguna.

Pengurutan data dalam komputasi sangat banyak macamnya, sebagai contoh *Bubble Sort, Insertion Sort*, *Selection Sort*, *Merge Sort*, *Heap Sort*, *Quick Sort,* dan *Shell Sort*. Setiap algoritma pengurutan data tersebut memiliki ciri khas dan logikanya masing-masing

## 2.2 Pengimplementasian Algoritma Pengurutan Data dalam Kasus Dunia Nyata

Andaikan seorang remaja penggemar *game arcade* bernama Julian mengoleksi banyak kartu dari permainan *game arcade* bernama RoboFight. Setiap kartu dari RoboFight memiliki kode unik, gambar robot, *power* dan *hitpoint (HP)* robot yang masing-masing berbeda satu dengan yang lain. Karena banyaknya kartu yang dimiliki oleh Julian, Julian sempat kewalahan untuk mengurutkan kartu-kartunya. Julian berinisiatif untuk membuat aplikasi yang mampu mengurutkan kartu-kartu tersebut berdasarkan kode unik, *power*, dan *HP* robot tersebut.

Berikut adalah data kartu tidak terurut yang dimiliki oleh Julian

|  |  |  |  |
| --- | --- | --- | --- |
| No | Kode Kartu | *Power* | *HP* |
| 1 | 195 | 4000 | 3000 |
| 2 | 432 | 5000 | 4500 |
| 3 | 351 | 4250 | 4500 |
| 4 | 140 | 3500 | 5000 |
| 5 | 247 | 5250 | 4250 |
| 6 | 258 | 4750 | 3750 |
| 7 | 179 | 6250 | 4750 |
| 8 | 316 | 5500 | 4500 |
| 9 | 478 | 5750 | 5500 |
| 10 | 510 | 6000 | 6000 |
| 11 | 174 | 5000 | 4750 |
| 12 | 375 | 6000 | 5750 |
| 13 | 427 | 7500 | 6250 |
| 14 | 414 | 6500 | 5250 |
| 15 | 152 | 6250 | 5000 |
| 16 | 469 | 4250 | 3750 |
| 17 | 537 | 7000 | 6000 |
| 18 | 532 | 6750 | 5250 |
| 19 | 473 | 4500 | 4750 |
| 20 | 264 | 4750 | 5250 |
| 21 | 360 | 5500 | 4250 |
| 22 | 172 | 5000 | 5250 |
| 23 | 164 | 4750 | 4500 |

Dengan aplikasi yang dibuat oleh Julian, diharapkan Julian dapat mengurutkan kartu-kartu yang dipilihnya sesuai dengan kriteria yang Julian inginkan. Aplikasi tersebut menyediakan 6 pilihan cara untuk mengurutkan kartu. Pengguna dapat mengurutkan kartu berdasarkan kode urutnya secara *ascending* (dari angka terkecil sampai terbesar) dan *descending* (dari angka terbesar sampai terkecil), hal ini juga berlaku untuk kriteria *power* dan *HP*.

Aplikasi dimulai dengan menginisiasi beberapa variable dan melakukan deklarasi variable untuk membantu proses pengurutan. *int CodeCount, PowerCount, HPCount* digunakan untuk menghitung banyaknya kartu yang masuk,\. *int CardsCode[50], CardsPower[50], Temp[50], CardsHP[50]* digunakan untuk menyimpan *array* data kartu yang masuk. *Int Choice* untuk melakukan pilihan fitur yang akan digunakan. Pengguna diharuskan untuk menginputkan semua data kartu terlebih dahulu lalu baru bisa melakukan pengurutan kartu.

int main(){

    int CodeCount = 0, PowerCount = 0, HPCount = 0;

    int CardsCode[50];

    int CardsPower[50];

    int Temp[50];

    int CardsHP[50];

    int Choice;

    char Login = 'y';

    while(Login == 'y'){

        cout << "==== Sorting RoboFight App ====\n";

        cout << "1. Insert all your Robofight cards \n";

        cout << "2. Sort all your Robofight card\n";

        cout << "3. Exit\n";

        cout << "> ";

        cin >> Choice;

Pada menu 1. *Insert all your Robofight cards*, pengguna bisa menginputkan data-data kartu berdasarkan kode kartu, *power*, ataupun *HP*. Setelah memilih salah satu opsi, pengguna menginputkan data kartu yang sebelumnya telah dipilih. Data kartu yang telah diinput akan dimasukkan dalam *array*. Bila kartu telah habis atau pengguna ingin menghentikan input, pengguna dapat menginputkan 0 untuk menghentikan *looping*.

if(Choice == 1){

            int ChoiceInsert;

            cout << "Select the method for inserting your cards\n";

            cout << "1. Insert cards based on Card code\n";

            cout << "2. Insert cards based on Power\n";

            cout << "3. Insert cards based on HP\n";

            cout << "> ";

            cin >> ChoiceInsert;

            switch (ChoiceInsert){

            case 1:

                while (true){

                int CodeCard;

                cout << "Insert your RoboFight's Card code (insert 0 to end) > ";

                cin >> CodeCard;

                    if (CodeCard == 0){

                        break;

                    }

                CardsCode[CodeCount] = CodeCard;

                CodeCount++;

                }

            break;

            case 2:

                while (true){

                int PowerCard;

                cout << "Insert your RoboFight's Power (insert 0 to end) > ";

                cin >> PowerCard;

                    if (PowerCard == 0){

                        break;

                    }

                CardsPower[PowerCount] = PowerCard;

                PowerCount++;

                }

            break;

            case 3:

                while (true){

                int HPCard;

                cout << "Insert your RoboFight's HP (insert 0 to end) > ";

                cin >> HPCard;

                    if (HPCard == 0){

                        break;

                    }

                CardsHP[HPCount] = HPCard;

                HPCount++;

                }

            break;

            default:

                cout << "Invalid input\n";

            break;

            }

        }

Pada menu 2. *Sort all your RoboFight cards*, pengguna dapat melakukan pengurutan dengan 6 pilihan opsi. Pengguna dapat memilih untuk mengurutkan kode kartu, *power,* atau *HP* baik secara *ascending* maupun *descending*. Untuk pengurutan kode kartu secara *ascending*, menggunakkan teknik *Bubble Sort*. Pengurutan kode kartu secara *descending* menggunakkan teknik *Insertion Sort*. Pengurutan *power* kartu secara *ascending* menggunakkan teknik *Selection Sort*. Pengurutan *power* kartu secara *descending* menggunakkan teknik *Merge Sort*. Pengurutan *HP* kartu secara *ascending* menggunakkan teknik *Heap Sort*. Pengurutan *HP* kartu secara *descending* menggunakkan teknik *Quick Sort*.

 if(Choice == 2){

            int ChoiceSort;

            cout << "Select the method for sorting your cards\n";

            cout << "1. Sorting based on Card code (Ascending)\n";

            cout << "2. Sorting based on Card code (Descending)\n";

            cout << "3. Sorting based on Power (Ascending)\n";

            cout << "4. Sorting based on Power (Descending)\n";

            cout << "5. Sorting based on HP (Ascending)\n";

            cout << "6. Sorting based on HP (Descending)\n";

            cout << "> ";

            cin >> ChoiceSort;

            switch(ChoiceSort){

                case 1:

                    BubbleSortAscending(CardsCode,CodeCount);

                    cout << "Sorted RoboFight's Card codes:\n";

                    for (int j = 0; j < CodeCount; j++){

                        cout << CardsCode[j] << " ";

                    }

                    cout << endl << endl;

                    cin.ignore();

                    cout << "Press Enter to continue";

                    cin.get();

                break;

                case 2:

                    InsertionSortDescending(CardsCode,CodeCount);

                    cout << "Sorted RoboFight's Card codes:\n";

                    for(int j = 0; j < CodeCount; j++){

                        cout << CardsCode[j] << " ";

                    }

                    cout << endl << endl;

                    cin.ignore();

                    cout << "Press Enter to continue";

                    cin.get();

                break;

                case 3:

                    SelectionSortAscending(CardsPower,PowerCount);

                    cout << "Sorted RoboFight's Card Power:\n";

                    for (int j = 0; j < PowerCount; j++){

                        cout << CardsPower[j] << " ";

                    }

                    cout << endl << endl;

                    cin.ignore();

                    cout << "Press Enter to continue";

                    cin.get();

                break;

                case 4:

                    MergeSortDescending(CardsPower,Temp,PowerCount);

                    cout << "Sorted RoboFight's Card Power:\n";

                    for (int j = 0; j < PowerCount; j++){

                        cout << CardsPower[j] << " ";

                    }

                    cout << endl << endl;

                    cin.ignore();

                    cout << "Press Enter to continue";

                    cin.get();

                break;

                case 5:

                    QuickSortAscending(CardsHP,HPCount);

                    cout << "Sorted RoboFight's Card HP:\n";

                    for (int j = 0; j < HPCount; j++){

                        cout << CardsHP[j] << " ";

                    }

                    cout << endl << endl;

                    cin.ignore();

                    cout << "Press Enter to continue";

                    cin.get();

                break;

                case 6:

                    HeapSortDescending(CardsHP,HPCount);

                    cout << "Sorted RoboFight's Card HP:\n";

                    for (int j = 0; j < HPCount; j++){

                        cout << CardsHP[j] << " ";

                    }

                    cout << endl << endl;

                    cin.ignore();

                    cout << "Press Enter to continue";

                    cin.get();

                break;

                default:

                    cout << "Invalid input\n";

                break;

            }

        }

Yang terakhir, pada menu 3. *Exit*. Pengguna dapat memilih untuk keluar dari aplikasi, menghentikan proses *looping.*

if(Choice == 3){

            char Exit;

            cout << "Are you sure to exit the application? (y/n)\n";

            cout << "> ";

            cin >> Exit;

            if(Exit == 'y' || Exit == 'Y'){

                Login = 'n';

            }

            else{

                Login = 'y';

            }

        }

    }

Setiap fungsi algoritma pengurutan data diletakkan di akhir program sehingga, terdapat *prototype fungsi* pada awal program.

void BubbleSortAscending(int Numbers[], int ArraySize);

void InsertionSortDescending(int Numbers[], int ArraySize);

void SelectionSortAscending(int Numbers[], int ArraySize);

void MergeSortDescending(int Numbers[], int Temp[], int ArraySize);

    void MSort(int Numbers[], int Temp[], int Left, int Right);

    void Merge(int Numbers[], int Temp[], int Left, int Mid, int Right);

void HeapSortAscending(int Numbers[], int ArraySize);

    void ShiftDown(int Numbers[], int Root, int Bottom);

void QuickSortDescending(int Numbers[], int ArraySize);

    void QSort(int Numbers[], int Left, int Right);

void ShellSortDescending(int Numbers[], int ArraySize);

## 2.3 Cara Kerja Algoritma Pengurutan Data

Setelah menjalankan dan menginputkan data sesuai dengan tabel sebelumnya, program dapat berjalan dengan baik dan dapat memberikan output yang diharapkan. Berikut adalah hasil input kode kartu berdasarkan table sebelumnya.

==== Sorting RoboFight App ====

1. Insert all your RoboFight cards

2. Sort all your RoboFight card

3. Exit

> 1

Select the method for inserting your cards

1. Insert cards based on Card code

2. Insert cards based on Power

3. Insert cards based on HP

> 1

Insert your RoboFight's Card code (insert 0 to end) > 195

Insert your RoboFight's Card code (insert 0 to end) > 432

Insert your RoboFight's Card code (insert 0 to end) > 351

Insert your RoboFight's Card code (insert 0 to end) > 140

Insert your RoboFight's Card code (insert 0 to end) > 247

Insert your RoboFight's Card code (insert 0 to end) > 258

Insert your RoboFight's Card code (insert 0 to end) > 179

Insert your RoboFight's Card code (insert 0 to end) > 316

Insert your RoboFight's Card code (insert 0 to end) > 478

Insert your RoboFight's Card code (insert 0 to end) > 510

Insert your RoboFight's Card code (insert 0 to end) > 174

Insert your RoboFight's Card code (insert 0 to end) > 375

Insert your RoboFight's Card code (insert 0 to end) > 427

Insert your RoboFight's Card code (insert 0 to end) > 414

Insert your RoboFight's Card code (insert 0 to end) > 152

Insert your RoboFight's Card code (insert 0 to end) > 469

Insert your RoboFight's Card code (insert 0 to end) > 537

Insert your RoboFight's Card code (insert 0 to end) > 532

Insert your RoboFight's Card code (insert 0 to end) > 473

Insert your RoboFight's Card code (insert 0 to end) > 264

Insert your RoboFight's Card code (insert 0 to end) > 360

Insert your RoboFight's Card code (insert 0 to end) > 172

Insert your RoboFight's Card code (insert 0 to end) > 164

Insert your RoboFight's Card code (insert 0 to end) > 164

Insert your RoboFight's Card code (insert 0 to end) > 0

void BubbleSortAscending(int Numbers[], int ArraySize){

    int i, j, temp;

    for(i = ArraySize - 1; i >= 0; i--){ //sistem decrement, melakukan pengecekan dari awal. Semakin kecil i, maka array yg belakang sudah dicek

        for(j = 1; j <= i;j++){ //melakukan pengecekan dari depan ke belakang

            if(Numbers[j-1] > Numbers[j]){ //jika terdapat angka yang lebih besar di depan indeks j

               temp = Numbers[j-1]; // maka terjadi pertukaran nilai

               Numbers[j-1] = Numbers[j];

               Numbers[j] = temp;

            }

      }

    }

}

*Bubble sort* merupakan algoritma pengurutan yang paling tua dan paling sederhana, tetapi algoritma *Bubble sort* adalah algoritma yang paling lambat apabila dibandingkan dengan algoritma yang lain, dikarenakan data ditukar satu per satu. Berikut adalah hasil dari *Bubble Sort* *Ascending*:

==== Sorting RoboFight App ====

1. Insert all your RoboFight cards

2. Sort all your RoboFight card

3. Exit

> 2

Select the method for sorting your cards

1. Sorting based on Card code (Ascending)

2. Sorting based on Card code (Descending)

3. Sorting based on Power (Ascending)

4. Sorting based on Power (Descending)

5. Sorting based on HP (Ascending)

6. Sorting based on HP (Descending)

> 1

Sorted RoboFight's Card codes:

140 152 164 172 174 179 195 247 258 264 316 351 360 375 414 427 432 469 473 478 510 532 537

Press Enter to continue

void InsertionSortDescending(int Numbers[], int ArraySize){

    int i,j,index;

    for (i = 1; i < ArraySize; i++){ //sistem increment

        index = Numbers[i]; //index menyimpan anggota i

        j = i; //j menyesuaikan urutan i

        while ((j > 0) && (Numbers[j-1] < index)){ //jika nomor sebelum j lebih kecil dari angka index

            Numbers[j] = Numbers[j-1]; //maka nomor yang lebih besar akan “diinsertkan” ke urutan j, sehingga data yang lebih kecil digeser ke sebelah kanan

            j--; //decrement untuk melakukan pengecekan ulang, siapa tahu ada data yang lebih kecil lagi

        }

        Numbers[j] = index; //jika memang tidak ada lagi, maka angka indeks dimasukkan ke urutan j

    }

}

*Insertion sort* adalah algoritma pengurutan yang menyisipkan data ke dalam tempat yang sesuai dalam indeks. Algoritma *Insertion sort* dianggap lebih efektif apabila dibandingkan dengan *Bubble sort* dan *Selection sort*, karena algoritma ini hanya membandingkan data dengan elemen yang sudah diurutkan sampai menemukan posisi yang tepat. Berikut adalah hasil dari *Insertion Sort Descending*:

==== Sorting RoboFight App ====

1. Insert all your RoboFight cards

2. Sort all your RoboFight card

3. Exit

> 2

Select the method for sorting your cards

1. Sorting based on Card code (Ascending)

2. Sorting based on Card code (Descending)

3. Sorting based on Power (Ascending)

4. Sorting based on Power (Descending)

5. Sorting based on HP (Ascending)

6. Sorting based on HP (Descending)

> 2

Sorted RoboFight's Card codes:

537 532 510 478 473 469 432 427 414 375 360 351 316 264 258 247 195 179 174 172 164 152 140

Press Enter to continue

Berikut adalah hasil input data *power* berdasarkan tabel sebelumnya

==== Sorting RoboFight App ====

1. Insert all your RoboFight cards

2. Sort all your RoboFight card

3. Exit

> 1

Select the method for inserting your cards

1. Insert cards based on Card code

2. Insert cards based on Power

3. Insert cards based on HP

> 2

Insert your RoboFight's Power (insert 0 to end) > 4000

Insert your RoboFight's Power (insert 0 to end) > 5000

Insert your RoboFight's Power (insert 0 to end) > 4250

Insert your RoboFight's Power (insert 0 to end) > 3500

Insert your RoboFight's Power (insert 0 to end) > 5250

Insert your RoboFight's Power (insert 0 to end) > 4750

Insert your RoboFight's Power (insert 0 to end) > 6250

Insert your RoboFight's Power (insert 0 to end) > 5500

Insert your RoboFight's Power (insert 0 to end) > 5750

Insert your RoboFight's Power (insert 0 to end) > 6000

Insert your RoboFight's Power (insert 0 to end) > 5000

Insert your RoboFight's Power (insert 0 to end) > 6000

Insert your RoboFight's Power (insert 0 to end) > 7500

Insert your RoboFight's Power (insert 0 to end) > 6500

Insert your RoboFight's Power (insert 0 to end) > 6250

Insert your RoboFight's Power (insert 0 to end) > 4250

Insert your RoboFight's Power (insert 0 to end) > 7000

Insert your RoboFight's Power (insert 0 to end) > 6750

Insert your RoboFight's Power (insert 0 to end) > 4500

Insert your RoboFight's Power (insert 0 to end) > 4750

Insert your RoboFight's Power (insert 0 to end) > 5500

Insert your RoboFight's Power (insert 0 to end) > 5000

Insert your RoboFight's Power (insert 0 to end) > 4750

Insert your RoboFight's Power (insert 0 to end) > 0

void SelectionSortAscending(int Numbers[], int ArraySize){

    int i,j, min, temp;

    for(i = 0;i < ArraySize - 1;i++){

         min = i; // angka min diambil paling depan dulu

         for(j = i+1; j < ArraySize; j++){ // j mencari angka terkecil

            if(Numbers[j] < Numbers[min]){ // jika ada angka yg lebih kecil dri min

                 min = j; // maka angka tersebut ditetapkan sbagai min

              }

              temp = Numbers[i]; //jika sudah, maka dilakukan pertukaran angka

              Numbers[i] = Numbers[min]; //angka paling depan dalam array adalah angka terkecil

              Numbers[min] = temp;

        }

    }

}

*Selection sort* adalah algoritma pengurutan yang menyeleksi data terkecil dari data yang belum terurut dan kemudian ditukar ke posisi terdepan. Kelemahan dari selection sort adalah waktu komputasi yang lama dalam menjalankan proses pengurutan terutama apabila *array* yang diurutkan terlalu besar. Berikut adalah hasil dari *Selection Sort Ascending*:

==== Sorting RoboFight App ====

1. Insert all your RoboFight cards

2. Sort all your RoboFight card

3. Exit

> 2

Select the method for sorting your cards

1. Sorting based on Card code (Ascending)

2. Sorting based on Card code (Descending)

3. Sorting based on Power (Ascending)

4. Sorting based on Power (Descending)

5. Sorting based on HP (Ascending)

6. Sorting based on HP (Descending)

> 3

Sorted RoboFight's Card Power:

3500 4000 4250 4250 4500 4750 4750 4750 5000 5000 5000 5250 5500 5500 5750 6000 6000 6250 6250 6500 6750 7000 7500

Press Enter to continue

//parameter array, array sementara, ukuran array

void MergeSortDescending(int Numbers[], int Temp[], int ArraySize){ //fungsi untuk memulai proses pengurutan

        MSort(Numbers, Temp, 0, ArraySize-1);

    }

    void MSort(int Numbers[], int Temp[], int Left, int Right){ //parameter array, array sementara, index left, right untuk array

        int Mid;

        if(Right > Left){ // pembagian array menjadi setengah, dengan melihat Arraysize dibandingkan dengan Left (0)

            Mid = (Right + Left) / 2; //membagi Array menjadi setengah

            MSort(Numbers, Temp, Left, Mid);

            MSort(Numbers, Temp, Mid+1, Right); //dilakukan hingga 1 array tersisa 2 member

            Merge(Numbers, Temp, Left, Mid+1, Right);

        }

    }

    //parameter array, array sementra, index left, mid, right untuk array

    void Merge(int Numbers[], int Temp[], int Left, int Mid, int Right){

        int i, Left\_End, Num\_Elements, Tmp\_Pos;

        Left\_End = Mid - 1; // menandakan bagian akhir dari array kiri, Numbers[3]

        Tmp\_Pos = Left;

        Num\_Elements = Right - Left + 1; // menentukan jumlah elemen yg akan ditulis

        while((Left <= Left\_End) && (Mid <= Right)){ //melakukan pengurutan

            if(Numbers[Left] >= Numbers[Mid]){ // jika angka pada array kiri lebih besar dari angka pada array kanan

                Temp[Tmp\_Pos] = Numbers[Left]; // maka angka yg lebih besar akan dimasukkan pada array sementara (sebagai pengurutan)

                Tmp\_Pos = Tmp\_Pos + 1;

                Left = Left +1;

            }

            else{ // jika angka pada array kanan lebih besar dari angka pada array kanan

                Temp[Tmp\_Pos] = Numbers[Mid];  // maka angka yg lebih besar akan dimasukkan pada array sementara (sebagai pengurutan)

                Tmp\_Pos = Tmp\_Pos + 1;

                Mid = Mid + 1;

            }

        }

        while(Left <= Left\_End){ //menggabungkan 2 bagian array menjadi 1 bagian, selama Numbers[0] blm mendekati Numbers[3]

            Temp[Tmp\_Pos] = Numbers[Left];

            Left = Left + 1;

            Tmp\_Pos = Tmp\_Pos + 1;

            }

        while (Mid <= Right){ //menggabungkan 2 bagian array menjadi 1 bagian

            Temp[Tmp\_Pos] = Numbers[Mid];

            Mid = Mid + 1 ;

            Tmp\_Pos = Tmp\_Pos + 1;

            }

        for (i = 0; i <= Num\_Elements; i++){ // pengcopyan array yang terurut dri Temp[] ke Numbers[]

            Numbers[Right] = Temp[Right];

            Right = Right - 1;

            }

        }

*Merge sort* adalah algoritma pengurutan yang melakukan proses pembagian *array* untuk memudahkan proses pengurutan. *Merge sort* membagi data yang terurut membagi 2 dan ditempatkan ke dalam *array* yang terpisah. Lalu masing-masing array diurutkan secara rekursif dan digabung apabila data dalam *array* sudah diurutkan. *Merge sort* adalah algoritma pengurutan yang rumit dan membutuhkan 3 fungsi sekaligus untuk melakukan pembagian dan melakukan pengurutan. Berikut adalah hasil pengurutan *Merge Sort Descending*:

==== Sorting RoboFight App ====

1. Insert all your RoboFight cards

2. Sort all your RoboFight card

3. Exit

> 2

Select the method for sorting your cards

1. Sorting based on Card code (Ascending)

2. Sorting based on Card code (Descending)

3. Sorting based on Power (Ascending)

4. Sorting based on Power (Descending)

5. Sorting based on HP (Ascending)

6. Sorting based on HP (Descending)

> 4

Sorted RoboFight's Card Power:

7500 7000 6750 6500 6250 6250 6000 6000 5750 5500 5500 5250 5000 5000 5000 4750 4750 4750 4500 4250 4250 4000 3500

Berikut adalah hasil input data *HP* berdasarkan tabel sebelumnya

==== Sorting RoboFight App ====

1. Insert all your RoboFight cards

2. Sort all your RoboFight card

3. Exit

> 1

Select the method for inserting your cards

1. Insert cards based on Card code

2. Insert cards based on Power

3. Insert cards based on HP

> 3

Insert your RoboFight's HP (insert 0 to end) > 3000

Insert your RoboFight's HP (insert 0 to end) > 4500

Insert your RoboFight's HP (insert 0 to end) > 4500

Insert your RoboFight's HP (insert 0 to end) > 5000

Insert your RoboFight's HP (insert 0 to end) > 4250

Insert your RoboFight's HP (insert 0 to end) > 3750

Insert your RoboFight's HP (insert 0 to end) > 4750

Insert your RoboFight's HP (insert 0 to end) > 4500

Insert your RoboFight's HP (insert 0 to end) > 5500

Insert your RoboFight's HP (insert 0 to end) > 6000

Insert your RoboFight's HP (insert 0 to end) > 4750

Insert your RoboFight's HP (insert 0 to end) > 5750

Insert your RoboFight's HP (insert 0 to end) > 6250

Insert your RoboFight's HP (insert 0 to end) > 5250

Insert your RoboFight's HP (insert 0 to end) > 5000

Insert your RoboFight's HP (insert 0 to end) > 3750

Insert your RoboFight's HP (insert 0 to end) > 6000

Insert your RoboFight's HP (insert 0 to end) > 5250

Insert your RoboFight's HP (insert 0 to end) > 4750

Insert your RoboFight's HP (insert 0 to end) > 5250

Insert your RoboFight's HP (insert 0 to end) > 4250

Insert your RoboFight's HP (insert 0 to end) > 5250

Insert your RoboFight's HP (insert 0 to end) > 4500

Insert your RoboFight's HP (insert 0 to end) > 0

void QuickSortAscending(int Numbers[], int ArraySize){

    QSort(Numbers, 0, ArraySize-1);

}

    void QSort(int Numbers[], int Left, int Right){

        int Pivot, LeftHold, RightHold; // LeftHold dan RightHold berfungsi untuk menyimpan posisi kiri dan kanan sebelum partisi

        LeftHold = Left, RightHold = Right, Pivot = Numbers[Left]; // Pivot dimulai angka array paling kiri

        while (Left < Right){ // Proses partisi

            while ((Numbers[Right] >= Pivot) && (Left < Right)){ // melakukan pengecekan apakah ada angka yg lebih kecil lagi

                Right--;

            }

            if(Left != Right){ // jika memang ada

                Numbers[Left] = Numbers[Right]; // maka terjadi pertukaran elemen

                Left++;

            }

            while ((Numbers[Left] >= Pivot) && (Left < Right)){ // melakukan pengecekan apakah ada angka yg lebih kecil lagi

                Left++;

            }

            if (Left != Right){ // jika memang ada

                Numbers[Right] = Numbers[Left]; // maka terjadi pertukaran elemen

                Right--;

            }

        }

        Numbers[Left] = Pivot; // mengembalikan pivot ke posisi semula

        Pivot = Left; // mengembalikan pivot ke posisi semula

        Left = LeftHold; // mengembalikan nilai left ke posisi semula

        Right = RightHold; // mengembalikan nilai right ke posisi semula

        if (Left < Pivot){ // jika msih ada elemen yg lebih kecil

            QSort(Numbers, Left, Pivot-1);

        }

        if (Right > Pivot){ // jika msih ada elemen yg lebih besar

            QSort(Numbers, Pivot+1, Right);

        }

        //base case terjadi apabila Left == Right == Pivot

    }

*Quick Sort* adalah algoritma pengurutan yang menggunakkan pivot sebagai patokan. Data-data di dalam array akan dibandingkan dengan pivot, sehingga didapatkan data yang nilainya kurang dari pivot dan data yang nilainya lebih dari pivot. Lalu terjadi partisi yang dimana data yang nilainya kurang dari pivot terletak di sebelah kiri pivot, dan data yang nilainya lebih dari pivot terletak di sebelah kanan pivot. Tapi, *Quick Sort* tersebut mengalami *bug* dan tidak bisa berjalan semestinya, sehingga berikut adalah hasil revisinya:

void QuickSortAscending(int Numbers[], int ArraySize){

    QSort(Numbers, 0, ArraySize - 1);

}

// Fungsi QSort

void QSort(int Numbers[], int Left, int Right){

    if (Left < Right) {

        int Pivot, LeftHold, RightHold;

        LeftHold = Left;

        RightHold = Right;

        Pivot = Numbers[(Left + Right) / 2]; // Pilih elemen tengah sebagai pivot, karena sebelumnya pivot masih diletakkan di kiri

        while (Left <= Right) {

            while (Numbers[Left] < Pivot)

                Left++;

            while (Numbers[Right] > Pivot)

                Right--;

            if (Left <= Right) {

                // Tukar elemen

                int temp = Numbers[Left];

                Numbers[Left] = Numbers[Right];

                Numbers[Right] = temp;

                Left++;

                Right--;

            }

        }

        // Rekursif untuk subarray kiri dan kanan dari pivot

        if (LeftHold < Right)

            QSort(Numbers, LeftHold, Right);

        if (Left < RightHold)

            QSort(Numbers, Left, RightHold);

    }

}

Berikut adalah hasil output dari *Quick Sort Ascending*:

==== Sorting RoboFight App ====

1. Insert all your RoboFight cards

2. Sort all your RoboFight card

3. Exit

> 2

Select the method for sorting your cards

1. Sorting based on Card code (Ascending)

2. Sorting based on Card code (Descending)

3. Sorting based on Power (Ascending)

4. Sorting based on Power (Descending)

5. Sorting based on HP (Ascending)

6. Sorting based on HP (Descending)

> 5

Sorted RoboFight's Card HP:

3000 3750 3750 4250 4250 4500 4500 4500 4500 4750 4750 4750 5000 5000 5250 5250 5250 5250 5500 5750 6000 6000 6250

Press Enter to continue

void HeapSortDescending(int Numbers[], int ArraySize){

    int i, Temp;

    for(i = (ArraySize/2)-1; i >= 0; i--){

        ShiftDown(Numbers, i, ArraySize); // proses pembuatan heap ke bawah, hingga i = 0

    }

    for(i = ArraySize-1; i >= 1; i--){

        Temp = Numbers[0];  // proses pemindahan anggota heap paling besar ke array paling belakang

        Numbers[0] = Numbers[i];

        Numbers[i] = Temp;

        ShiftDown(Numbers,0,i); // lalu dirapikan lagi dengan fungsi ShiftDown

    }

}

    void ShiftDown(int Numbers[], int Root, int Bottom){ // Root menunjukkan subtree, Bottom menunjukkan akar paling dasar dari subtree

        int Done, MaxChild, Temp;

        Done = 0;

        while ((Root \* 2 <= Bottom) && (!Done)){ // selama semua root belum punya anak 2 akar dan iterasi blm selesai

            if(Root \* 2 == Bottom){ // pengecekkan apakah node root adalah node terakhir yg tdk punya akar atau akarnya hanya 1

                MaxChild = Root \* 2;

            }

            else if(Numbers[Root \* 2] > Numbers[Root \* 2 + 1]){ // pengecekan perbandingan nilai dari 2 anak akar tersebut

                MaxChild = Root \* 2;

            }

            else{

                MaxChild = Root \* 2 + 1; // nilai terbawah dalam array akan disimpan dalam MaxChild

            }

            if(Numbers[Root] < Numbers[MaxChild]){ // jika terdapat nilai anak yg lebih kecil dari nilai parentnya

                Temp = Numbers[Root]; // maka terjadi pertukaran nilai

                Numbers[Root] = Numbers[MaxChild];

                Numbers[MaxChild] = Temp;

                Root = MaxChild; //sehingga, nilai yg terkecil akan berada di bawah

            }

            else{

                Done = 1;

            }

        }

    }

*Heap sort* adalah algoritma pengurutan yang menggunakkan struktur data heap untuk mengurutkan data. Awalnya, algoritma akan menyusun heap, sehingga heap paling puncak adalah data terbesar dalam *array*. Setelah itu, data terbesar akan ditukar dengan anak heap yang memiliki data paling kecil, dilanjutkan kembali dengan anak heap selanjutnya dengan data terbesar keduanya. Sampai akhirnya heap menjadi urut dimulai dari data terkecil, hingga anak heap terbawah adalah data terbesar dalam array. Tapi, dalam kasus ini, dibalik sehingga array dimulai dari data terbesar, hingga anak heap terbawah adalah data terkecil dalam array. Tapi, *Heap Sort* tersebut mengalami *bug* dan tidak bisa berjalan semestinya, sehingga berikut adalah hasil revisinya:

void HeapSortDescending(int Numbers[], int ArraySize) { //revisi

    int i, Temp;

    for (i = (ArraySize / 2) - 1; i >= 0; i--) {

        ShiftDown(Numbers, i, ArraySize);

    }

    for (i = ArraySize - 1; i >= 1; i--) {

        Temp = Numbers[0];

        Numbers[0] = Numbers[i];

        Numbers[i] = Temp;

        ShiftDown(Numbers, 0, i - 1);

    }

}

    void ShiftDown(int Numbers[], int Root, int Bottom) {

        int Done = 0, MaxChild, Temp;

        while ((Root \* 2 <= Bottom) && (!Done)) {

            if (Root \* 2 == Bottom || Numbers[Root \* 2] > Numbers[Root \* 2 + 1]) {

                MaxChild = Root \* 2;

            } else {

                MaxChild = Root \* 2 + 1;

            }

            if (Numbers[Root] < Numbers[MaxChild]) {

                Temp = Numbers[Root];

                Numbers[Root] = Numbers[MaxChild];

                Numbers[MaxChild] = Temp;

                Root = MaxChild;

            } else {

                Done = 1;

            }

        }

    }

Berikut adalah hasil output dari *Heap Sort Descending*:

==== Sorting RoboFight App ====

1. Insert all your RoboFight cards

2. Sort all your RoboFight card

3. Exit

> 2

Select the method for sorting your cards

1. Sorting based on Card code (Ascending)

2. Sorting based on Card code (Descending)

3. Sorting based on Power (Ascending)

4. Sorting based on Power (Descending)

5. Sorting based on HP (Ascending)

6. Sorting based on HP (Descending)

> 6

Sorted RoboFight's Card HP:

6250 6000 6000 5750 5500 5250 5250 5250 5000 5000 4750 4750 4750 4500 4500 4500 4500 4250 4250 3750 3750 3000

Press Enter to continue

void ShellSortAscending(int Numbers[], int ArraySize){

    int i, j, Incre, Temp;

    Incre = 3; // untuk menentukan jarak antar elemen yg dibandingkan

    while (Incre > 0){

        for (i = 0; i < ArraySize; i++){ //iterasi untuk semua anggota array

            j = i; // variabel j untuk menandai posisi saat ini dalam iterasi

            Temp = Numbers[i];

            while ((j >= Incre) && (Numbers[j-Incre] > Temp)){ // loop untuk membandingkan elemen pada jarak incre

                Numbers[j] = Numbers[j - Incre]; // maka terjadi proses pertukaran

                j = j - Incre;

            }

                Numbers[j] = Temp; // nilai sementara disimpan di posisi yg benar

        }

        if (Incre/2 != 0){ // penanda bahwa increment perlu dibagi lagi (karena blm floor (2/2))

            Incre = Incre / 2;

        }

        else if (Incre == 1){ // jika memang incre = 1, maka berhenti

            Incre = 0;

        }

        else{

            Incre = 1; //apabila jumlah incre nya adlah bilangan ganjil

        }

    }

}

*Shell sort* adalah algoritma pengurutan yang bekerja dengan membagi deretan data dengan jarak tertentu. Lalu, data yang sudah terbagi diurut dengan cara *Insertion sort*. Setelah data yang sudah terbagi, jarak deretan data diperkecil sampai tidak dapat diurutkan kembali (karena data sudah terurut semua). Sehingga, *Shell sort* dapat dikategorikan sebagai perkembangan dari *Insertion sort*. *Shell Sort* saya coba aplikasikan kepada pilihan menu *Sorting based on Card Code (Ascending)* dan bisa berjalan dengan baik, berikut adalah output dari *Shell Sort Ascending*:

==== Sorting RoboFight App ====

1. Insert all your RoboFight cards

2. Sort all your RoboFight card

3. Exit

> 2

Select the method for sorting your cards

1. Sorting based on Card code (Ascending)

2. Sorting based on Card code (Descending)

3. Sorting based on Power (Ascending)

4. Sorting based on Power (Descending)

5. Sorting based on HP (Ascending)

6. Sorting based on HP (Descending)

> 1

Sorted RoboFight's Card codes:

140 152 164 172 174 179 195 247 258 264 316 351 360 375 414 427 432 469 473 478 510 532 537

Press Enter to continue

## 2.4 Algoritma Pengurutan Data yang Terbaik untuk Diaplikasikan

## 

Dari semua algortima pengurutan data yang telah dicoba dan diaplikasikan, berikut adalah penilaian saya mengenai algoritma-algoritma tersebut :

* + - 1. *Bubble Sort*

Algoritma sederhana untuk diimplementasikan, namun kinerjanya buruk apabila dibandingkan dengan algoritma yang lain. Tidak efisien untuk digunakan pada data besar dan banyak.

* + - 1. *Insertion Sort*

Algoritmanya sama sederhana dengan *Bubble Sort*, namun lebih efisien untuk digunakan pada data yang lebih sedikit dan lebih terurut.

* + - 1. *Selection Sort*

Algoritmanya mudah diimplementasikan tapi sangat tidak efisien pada data yang besar karena kompleksitas waktunya yang tinggi.

* + - 1. *Merge Sort*

Algoritmanya rumit dan membutuhkan banyak memori serta banyak fungsi. Namun sangat efisien dalam proses pengurutan dan memiliki kompleksitas waktu yang baik apabila dibandingkan dengan algoritma yang lain.

* + - 1. *Heap Sort*

Algoritmanya sama rumitnya dengan *Merge Sort*. Namun sama-sama efisien dalam proses pengurutan dan memiliki kompleksitas waktu yang baik.

* + - 1. *Quick Sort*

Algoritmanya tidak serumit *Merge Sort* dan *Heap Sort* dan memiliki efisien kompleksitas waktu yang baik. Namum memiliki kelemahan terhadap data yang dimasukkan, sehingga menyebabkan *bug* pada program saya, sehingga akan lebih baik bila data yang dimasukkan setidaknya sudah lebih terurut.

* + - 1. *Shell Sort*

Algoritmanya tidak serumit *Merge Sort, Heap Sort,* dan *Quick Sort* dan lebih efisien dari *Bubble Sort, Selection Sort*, dan *Insertion Sort*. Tetapi tidak seefisien *Merge Sort, Heap Sort,* dan *Quick Sort.*

Dari semua algoritma tersebut, semuanya memiliki ciri khas dan kelebihannya masing-masing. Suatu algoritma akan sangat lebih baik apabila dibandingkan dengan algoritma lainnya. Tetapi dalam kasus ini, karena tidak ada kebutuhan khusus seperti memori terbatas dan kinerja tertentu pada suatu jenis data tertentu, maka *Merge Sort* adalah pilihan terbaik karena efisien dan kompleksitas waktunya terbaik.