**LAPORAN TUGAS STRUKTUR DATA**

**APLIKASI PENERAPAN BERBAGAI METODE SEARCHING**



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 *Searching*” sebagai tugas struktur data tepat pada waktunya.

Adapun tujuan penulisan laporan ini adalah untuk mengetahui apa saja penerapan metode pencarian dan mengetahui mana penerapan metode pencarian 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 pencarian dan mengetahui mana penerapan metode pencarian yang paling efektif.

Surabaya, 22 April 2024

Penulis

# BAB I

# PENDAHULUAN

## 1.1 Latar Belakang

Pengolahan dan struktur data tidak dipisahkan dari computer pada masa kini dengan seiring berkembangnya jumlah dan kompleksitas data, salah satu aspek penting dalam pengolahan data adalah *searching* (pencarian data). *Searching* adalah proses dimana elemen atau data yang diinginkan, akan dicari oleh komputer dengan proses iterasi, bukan dicari secara manual oleh pengguna.

Bayangkan seorang manusia harus mencari ratusan atau ribuan data secara manual. Pastinya akan sangat membingungkan dan memakan waktu yang lama. Maka dari itu, penggunaan *searching* dalam struktur dara sangat lah penting. Pemilihan algoritma pencarian data 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. Melanjutkan program sebelumnya, Julian berinisiatif untuk membuat fitur dimana dia bisa mencari data kartu yang dimilikinya. Dengan memasukkan kode kartu, *power*, atau *HP*, Julian dapat mencari dan mengetahui apakah dia memiliki kartu yang dicarinya atau tidak.

## 1.2 Rumusan Masalah

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

## 1.3 Tujuan

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

# BAB II

# KAJIAN PUSTAKA

## 2.1 Pengertian dan Contoh dari Algoritma Pencarian Data

Pencarian data adalah proses dimana komputer melakulan penelusuran data atau informasi yang diinginkan dari kumpulan data atau *database.* Proses pencarian melibatkan penggunaan algoritma dengan mencocokkan kriteria yang diinginkan terhadap data yang ada. Tujuannya adalah menemukan data yang relevan dengan cepat dan efisien, diproses ataupun dimanipulasi oleh pengguna.

Tempat pencarian data dapat berupa array dalam memori, bisa juga pada file *internal* atau *external storage*. Algoritma pencarian data terdiri atas berbagai macam metode, seperti *Sequential Search, Binary Search, Interpolation Search*. Setiap algoritma pencarian data tersebut memiliki ciri khas dan kelebihannya 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. Melanjutkan program sebelumnya, Julian berinisiatif untuk membuat fitur dimana dia bisa mencari data kartu yang dimilikinya. Dengan memasukkan kode kartu, *power*, atau *HP*, Julian dapat mencari dan mengetahui apakah dia memiliki kartu yang dicarinya atau tidak.

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 telah dibuat oleh Julian sebelumnya, Julian dapat mencari kriteria kartu yang diinginkannya. Aplikasi tersebut menyediakan 3 pilihan cara untuk mencari kartu yang diinginkan. Pengguna dapat mencari kartu berdasarkan kode kartu, *power*, ataupun *HP*. Dengan menambahkan pilihan menu *3.Find your RoboFight Cards*

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'){

        system("cls");

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

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

        cout << "2. Sort all your RoboFight cards\n";

        cout << "3. Find your RoboFight cards\n";

        cout << "4. Exit\n";

        cout << "> ";

        cin >> Choice;

Setelah mengakses menu *3.Find your RoboFight Cards*. Maka, pengguna dapat mencari kartu yang dicari berdasarkan kode kartu, *power*, ataupun *HP*.

 if(Choice == 3){

     int ChoiceSearch;

    bool Found;

    cout << "Find your RoboFight cards\n";

     cout << "1. Based on Card Code\n";

     cout << "2. Based on Power\n";

     cout << "3. Based on HP\n";

     cout << "> ";

     cin >> ChoiceSearch;

Dalam menu 1. *Based on Card Code*, pengguna bisa menginputkan data kartu yang ingin dicari. Menu 1 menggunakkan fungsi *Sequential Search* untuk melakukan pencarian. Jika fungsi *Sequential Search* mengembalikan nilai *true*, maka data kartu tersebut ada dalam array. Jika fungsi *Sequential Search* mengembalikan nilai *false*, maka data kartu tidak ada dalam array. Fungsi *Sequential Search* tidak memerlukan proses pengurutan.

case 1:

     int SearchCode;

    cout << "Find your card code > ";

      cin >> SearchCode;

      Found = SequentialSearch(CardsCode, SearchCode, CodeCount);

    if(Found == true){

        cout << SearchCode << " is found, you have card with code " << SearchCode << endl << endl;

      cin.ignore();

      cout << "Press Enter to continue\n";

        cin.get();

    }

    else if(Found == false){

       cout << SearchCode << " is not found, you do not have card with code " << SearchCode << endl << endl;

        cin.ignore();

        cout << "Press Enter to continue\n";

        cin.get();

    }

break;

Dalam menu 2. *Based on Power Code*, pengguna bisa menginputkan data kartu yang ingin dicari. Menu 2 menggunakkan fungsi *Binary Search* untuk melakukan pencarian. Jika fungsi *Binary Search* mengembalikan nilai *true*, maka data kartu tersebut ada dalam array. Jika fungsi *Binary Search* mengembalikan nilai *false*, maka data kartu tidak ada dalam array. Fungsi *Binary Search* memerlukan proses pengurutan, sehingga dilakukan pemanggilan fungsi *Bubble Sort Ascending*.

case 2:

  int SearchPower;

      cout << "Find your card power > ";

      cin >> SearchPower;

      BubbleSortAscending(CardsPower, PowerCount);

      Found = BinarySearch(CardsPower, SearchPower, PowerCount);

    if(Found == true){

        cout << SearchPower << " is found, you have card with power " << SearchPower << endl << endl;

         cin.ignore();

         cout << "Press Enter to continue\n";

         cin.get();

      }

      else if(Found == false){

        cout << SearchPower << " is not found, you do not have card with power " << SearchPower << endl << endl;

        cin.ignore();

        cout << "Press Enter to continue\n";

      cin.get();

      }

break;

Dalam menu 3. *Based on HP Code*, pengguna bisa menginputkan data kartu yang ingin dicari. Menu 1 menggunakkan fungsi *Interpolation Search* untuk melakukan pencarian. Jika fungsi *Interpolation Search* mengembalikan nilai *true*, maka data kartu tersebut ada dalam array. Jika fungsi *Interpolation Search* mengembalikan nilai *false*, maka data kartu tidak ada dalam array. Fungsi *Interpolation Search* memerlukan proses pengurutan, sehingga dilakukan pemanggilan fungsi *Bubble Sort Ascending*.

case 3:

    int SearchHP;

    cout << "Find your card HP > ";

    cin >> SearchHP;

    BubbleSortAscending(CardsHP, HPCount);

    Found = InterpolationSearch(CardsHP, SearchHP, HPCount);

    if(Found == true){

        cout << SearchHP << " is found, you have card with HP " << SearchHP << endl << endl;

          cin.ignore();

          cout << "Press Enter to continue\n";

          cin.get();

    }

      else if(Found == false){

          cout << SearchHP << " is not found, you do not have card with HP " << SearchHP << endl << endl;

          cin.ignore();

          cout << "Press Enter to continue\n";

          cin.get();

      }

break;

## 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 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

> 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

bool SequentialSearch(int Numbers[], int Find, int ArraySize){

    int i = 0;

    Numbers[ArraySize - 1] = Find; // Find dijadikan patokan (ditaruh di array paling terakhir)

    while (Numbers[i] != Find){ // selama Find blm ditemukan, dilakukan incre

        i++;

        if (i < ArraySize){ // jika data ditemukan sebelum Find, maka data ditemukan

            return true; // maka mengembalikan nilai true

        }

    } // jika data tidak ditemukan sebelum Find, maka data tidak ditemukan

    return false; // maka mengembalikan nilai true

}

*Sequential Search* adalah algoritma paling sederhana untuk mencari data dan semua elemen *array* dari awal sampai akhir, yang dimana data-data tidak perlu diurutkan terlebih dahulu. Kasus terbaik nya adalah jika data terletak di indeks *array* terdepan, sehingga waktu pencarian lebih cepat. Kasus terburuk nya adalah jika data terletak di indeks *array* terbelakang, sehingga waktu. Berikut adalah hasil output *Sequential Search* jika data berhasil ditemukan:

==== Sorting RoboFight App ====

1. Insert all your RoboFight cards

2. Sort all your RoboFight cards

3. Find your Robofight cards

4. Exit

> 3

Find your RoboFight cards

1. Based on Card Code

2. Based on Power

3. Based on HP

> 1

Find your card code > 147

147 is found, you have card with code 147

Press Enter to continue

Berikut adalah hasil output *Sequential Search* jika data tidak berhasil ditemukan:

==== Sorting RoboFight App ====

1. Insert all your RoboFight cards

2. Sort all your RoboFight cards

3. Find your Robofight cards

4. Exit

> 3

Find your RoboFight cards

1. Based on Card Code

2. Based on Power

3. Based on HP

> 1

Find your card code > 100

100 is not found, you do not have card with code 100

Press Enter to continue

Berikut adalah hasil input *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

bool BinarySearch(int Numbers[], int Find, int ArraySize){

    int Left, Mid, Right;

    Left = 0;

    Right = ArraySize - 1;

    while (Left <= Right){ // Selama awalan, pertengahan, akhiran tidak berada di data yg sama

        Mid = (Left + Right) / 2; // Dibuat patokan mid, dibagi setengah

        if (Numbers[Mid] == Find){ // Jika Mid menemukan angka yg dicari, maka mengembalikan nilai true

            return true;

        }

        else if (Find < Numbers[Mid]){ // Jika data lebih kecil dari Mid

            Right = Mid - 1; // maka ruang pencarian diperkecil ke kiri

        }

        else{ // Jika data lebih besar dari Mid

            Left = Mid + 1; // maka ruang pencarian diperkecil ke kanan

        }

    }

    return false;

}

*Binary Search* adalah cara membagi data menjadi 2 bagian sekali terjadi terjadi proses pencarian. Algoritma ini bekerja dengan membagi *array* menjadi dua bagian secara berulang, dan membandingkan nilai yang dicari dengan nilai tengah *array.* Dengan cara ini, setiap langkah akan mengurangi jumlah data yang perlu diperiksa menjadi setengah dari langkah sebelumnya. *Binary Search* harus mengalami proses pengurutan terlebih dahulu. Berikut adalah hasil output *Binary Search* jika data berhasil ditemukan:

==== Sorting RoboFight App ====

1. Insert all your RoboFight cards

2. Sort all your RoboFight cards

3. Find your Robofight cards

4. Exit

> 3

Find your RoboFight cards

1. Based on Card Code

2. Based on Power

3. Based on HP

> 2

Find your card power > 5000

5000 is found, you have card with power 5000

Press Enter to continue

Berikut adalah hasil output *Binary Search* jika data tidak berhasil ditemukan:

==== Sorting RoboFight App ====

1. Insert all your RoboFight cards

2. Sort all your RoboFight cards

3. Find your Robofight cards

4. Exit

> 3

Find your RoboFight cards

1. Based on Card Code

2. Based on Power

3. Based on HP

> 2

Find your card power > 7750

7750 is not found, you do not have card with power 7750

Press Enter to continue

Berikut adalah hasil input *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

bool InterpolationSearch(int Numbers[], int Find, int ArraySize){

    int Low = 0, High = ArraySize - 1; // Low dimulai dari awal array, High dimulai dari akhir array

    float Pos1;

    int Pos;

    do{

        Pos1 = (float)(Find-Numbers[Low])/(Numbers[High]-Numbers[Low]) \* (High-Low) + Low; // rumus interpolasi

        Pos = Pos1; // maka dimasukkan dalam index

        if(Numbers[Pos] == Find){ // jika data ditemukan sebelum Find, maka data ditemukan

            return true; // maka mengembalikan nilai true

            break;

        }

        if (Numbers[Pos] > Find){ // Jika hasil interpolasi masih lebih besar, maka High dikurangi hasil interpolasi

            High = Pos - 1;

        }

        else if(Numbers[Pos] < Find){ // Jika hasil interpolasi masih lebih kecil, maka Low dijumlahkan hasil interpolasi

            Low = Pos + 1;

        }

    } while (Find >= Numbers[Low] && Find <= Numbers[High]); //selama Find berada di range array

    return false;

}

*Interpolation Search* adalah algoritma yang sudah terurut berdasarkan suatu kunci atau indeks tertentu. *Interpolation Search* dilakukan dengan perkiraan data. Perkiraan data dihitung dengan rumus sebagai berikut :

Posisi = - Data[*Low*] \* (*High – Low*) + *Low*

Jika Data[Posisi] > Data [*Find*], maka *High* = Pos – 1,

Jika Data[Posisi] < Data [*Find*], maka *Low* = Pos + 1.

Namun, fungsi *Interpolation Search* tersebut mengalami *bug* dan tidak bisa berjalan semestinya, sehingga berikut adalah revisinya :

bool InterpolationSearch(int Numbers[], int Find, int ArraySize){

    int Low = 0, High = ArraySize - 1; // Low dimulai dari awal array, High dimulai dari akhir array

    float Pos1;

    int Pos;

    do{

        Pos1 = (float)(Find-Numbers[Low])/(Numbers[High]-Numbers[Low]) \* (High-Low) + Low; // rumus interpolasi

        Pos = Pos1; // maka dimasukkan dalam index

        if(Numbers[Pos] == Find){ // jika data ditemukan sebelum Find, maka data ditemukan

            return true; // maka mengembalikan nilai true

        }

        if (Numbers[Pos] > Find){ // Jika hasil interpolasi masih lebih besar, maka High dikurangi hasil interpolasi

            High = Pos - 1;

        }

        else if(Numbers[Pos] < Find){ // Jika hasil interpolasi masih lebih kecil, maka Low dijumlahkan hasil interpolasi

            Low = Pos + 1;

        }

    } while (Find >= Numbers[Low] && Find <= Numbers[High]); //selama Find berada di range array

    return false;

}

Berikut adalah hasil output *Interpolation Search* jika data berhasil ditemukan:

==== Sorting RoboFight App ====

1. Insert all your RoboFight cards

2. Sort all your RoboFight cards

3. Find your Robofight cards

4. Exit

> 3

Find your RoboFight cards

1. Based on Card Code

2. Based on Power

3. Based on HP

> 3

Find your card HP > 5000

5000 is found, you have card with HP 5000

Press Enter to continue

Berikut adalah hasil output *Interpolation Search* jika data tidak berhasil ditemukan:

==== Sorting RoboFight App ====

1. Insert all your RoboFight cards

2. Sort all your RoboFight cards

3. Find your Robofight cards

4. Exit

> 3

Find your RoboFight cards

1. Based on Card Code

2. Based on Power

3. Based on HP

> 3

Find your card HP > 5000

10000 is found, you do not have card with HP 10000

Press Enter to continue

## 2.4 Algoritma Pencarian Data yang Terbaik untuk Diaplikasikan

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

1. *Sequensial Search*

Algoritmanya sederhana untuk diimplementasikan dan tidak memerlukan data terurut. Namun tidak efisien bila digunakan untuk melakukan pencarian *array* ukuran besar dan memiliki kompleksitas waktu yang buruk bila dibandingkan dengan algoritma yang lain. Algoritma *Selection Search* lebih baik digunakan pada *array* dengan data kecil.

2. *Binary Search*

Algoritmanya efisien dengan kompleksitas waktu yang baik, sangat cocok untuk *array* besar yang digunakan. Namun *Binary Search* memiliki kelemahan dimana *Binary Search* memerlukan *array* yang terurut dan tidak bisa digunakan pada *array* yang dinamis (sering beubah-ubah).

3. *Interpolation Search*

Algoritmanya dan kompleksitas waktunya lebih efisien daripada *Binary Search*. Namun *Interpolation Search* tidak cocok untuk data yang tidak terdistribusi secara merata, sehingga data harus diurutkan terlebih dahulu.

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 *Interpolation Search* adalah pilihan yang terbaik untuk alasan algoritma dan kompleksitas waktunya.