Analisis Algoritma Greedy dan Brute-Force

Fadhil Hidayat, NIM. 23509313
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

Abstrak

Algoritma Greedy adalah algoritma yang berusaha memecahkan masalah dengan cara mengambil pilihan terbaik atau solusi optimum yang diperoleh saat itu tanpa mempertimbangkan konsekwensi yang diterimanya kemudian. Sedangkan algoritma Brute-Force adalah algoritma yang yang lempang (*straightforward*) untuk memecahkan suatu masalah, biasanya langsung pada pernyataan masalah (*problem statement*), dan definisi konsep yang dilibatkan. Algoritma Greedy dan algoritma Brute-Force dapat diterapkan dalam penyelesaian persoalan transportasi seimbang, persoalan pewarnaan graf, dan pencarian kombinasi 5 kartu pada permainan poker. Eksperimen ini akan membahas langkah-langkah penyelesaian persoalan tersebut menggunakan algoritma Greedy dan Brute-Force. Pada eksperimen ini juga akan dicari kompleksitas algoritma Greedy dan Brute-Force dalam menyelesaikan 3 persoalan di atas.

Kata kunci: *Algoritma Greedy, algoritma Brute-Force, transportasi seimbang, pewarnaan graf, permainan poker*.

1. Pendahuluan

Algoritma Greedy adalah algoritma yang berusaha memecahkan masalah dengan cara mengambil pilihan terbaik atau solusi optimum yang diperoleh saat itu tanpa mempertimbangkan konsekwensi yang diterimanya kemudian. Dengan kata lain algoritma Greedy berusaha mencari solusi optimum lokal dan berharap dari optimum-optimum lokal ini ditemukan optimum globalnya.

Algoritma Brute-Force adalah sebuah pendekatan yang lempang (*straightforward*) untuk memecahkan suatu masalah, biasanya langsung pada pernyataan masalah (*problem statement*), dan definisi konsep yang dilibatkan.

Algoritma Greedy dan algoritma Brute-Force dapat diterapkan pada persoalan berikut:

1) Persoalan transportasi seimbang
2) Persoalan pewarnaan graf
3) Kombinasi 5 kartu pada permainan poker

Pada pembahasan berikutnya akan dipaparkan penyelesaian 3 persoalan di atas dengan menggunakan algoritma Greedy dan algoritma Brute-Force. Hasil akhirnya akan diperoleh perbandingan hasil akhir yang didapat dari algoritma Greedy dan algoritma Brute-Force. Selain membandingkan hasil akhir penyelesaian persoalan di atas, akan dibahas juga hasil analisis kompleksitas algoritma Greedy dan algoritma Brute-Force yang digunakan.

2. Persoalan Transportasi Seimbang

Pada dasarnya, persoalan transportasi muncul karena beberapa faktor berikut:

1) Terdapat sejumlah produsen (*supplier*) pada suatu daerah tertentu.
2) Terdapat sejumlah permintaan (*demmand*) oleh konsumen pada suatu daerah tertentu.
3) Sejumlah produsen memiliki kemampuan memproduksi suatu barang dengan jumlah tertentu.
4) Sejumlah konsumen memiliki sejumlah permintaan tertentu atas suatu barang yang diproduksi oleh produsen.
5) Terdapat variasi biaya (*cost*) yang harus dibayarkan untuk mendistribusikan sejumlah barang tertentu dari produsen (*supplier*) kepada konsumen sesuai dengan permintaannya (*demmand*).

Secara umum, persoalan transportasi yang muncul dari faktor-faktor di atas adalah bagaimana meminimalkan biaya (*cost*) untuk mendistribusikan barang dari produsen (*supplier*) yang terletak pada suatu daerah tententu kepada konsumen pada daerah tertentu sehingga seluruh permintaan (*demmand*) akan barang dapat terpenuhi, dengan tujuan untuk memaksimalkan keuntungan. Sedangkan persoalan transportasi seimbang adalah persoalan transportasi dimana total jumlah barang yang dihasilkan oleh *supplier* sama dengan jumlah barang yang diminta (*demmand*) oleh konsumen.

Persoalan transportasi seimbang dapat digambarkan dalam suatu matriks persoalan transportasi seperti pada gambar berikut ini:

Supplier (Sn) Demmand (Dn)	S_1	S_2	S_3	$S_{...}$	S_m	$\sum (D_{i,j})$
D_1	$X_{(1,1)}$ $C_{(1,1)}$	$X_{(1,2)}$ $C_{(1,2)}$	$X_{(1,3)}$ $C_{(1,3)}$	$X_{(1,...)}$ $C_{(1,...)}$	$X_{(1,m)}$ $C_{(1,m)}$	$D_{1,j}$
D_2	$X_{(2,1)}$ $C_{(2,1)}$	$X_{(2,2)}$ $C_{(2,2)}$	$X_{(2,3)}$ $C_{(2,3)}$	$X_{(2,...)}$ $C_{(2,...)}$	$X_{(2,m)}$ $C_{(2,m)}$	$D_{2,j}$
D_3	$X_{(3,1)}$ $C_{(3,1)}$	$X_{(3,2)}$ $C_{(3,2)}$	$X_{(3,3)}$ $C_{(3,3)}$	$X_{(3,...)}$ $C_{(3,...)}$	$X_{(3,m)}$ $C_{(3,m)}$	$D_{3,j}$
$D_{...}$	$X_{(...,1)}$ $C_{(...,1)}$	$X_{(...,2)}$ $C_{(...,2)}$	$X_{(...,3)}$ $C_{(...,3)}$	$X_{(...,...)}$ $C_{(...,...)}$	$X_{(...,m)}$ $C_{(...,m)}$	$D_{...,j}$
D_n	$X_{(n,1)}$ $C_{(n,1)}$	$X_{(n,2)}$ $C_{(n,2)}$	$X_{(n,3)}$ $C_{(n,3)}$	$X_{(n,...)}$ $C_{(n,...)}$	$X_{(n,m)}$ $C_{(n,m)}$	$D_{n,j}$
$\sum (S_{i,j})$	$S_{i,1}$	$S_{i,2}$	$S_{i,3}$	$S_{i,..}$	$S_{i,m}$	$\sum (S_{i,j}) = \sum (D_{i,j})$

Gambar 2. 1 Matriks persoalan transportasi seimbang.

Penjelasan matriks di atas, misalkan terdapat sejumlah *n Supplier* (*S*) dan sejumlah *m Demmand* (*D*). Masing masing *Supplier* ($S_{i,j}$) akan menghasilkan sejumlah produk ($S_{i,j}$), dan sejumlah

demmand ($D_{i,j}$) membutuhkan permintaan *supply* barang dengan jumlah tertentu ($D_{i,j}$). Setiap barang yang didistribusikan dari *supplier* ke *demmand* dikenakan biaya atau *cost* tertentu ($C_{(i,j)}$). Persoalan transportasi seimbang berarti jumlah barang yang dihasilkan oleh supplier sama dengan jumlah demmand yang dibutuhkan konsumen ($\sum (S_{i,j}) = \sum (D_{i,j})$). Persoalan transportasi dari matriks di atas adalah bagaimana agar seluruh *supply* (S_i) dapat memenuhi seluruh *demmand* (D_j) serta memememinumkan *cost* yang dinyatakan dalam persamaan *cost* (z) berikut:

$$z = \sum_{i=1}^{n}\sum_{j=1}^{n} X_{(i,j)}.C_{(i,j)} = X_{(1,1)}.C_{(1,1)} + X_{(1,2)}.C_{(1,2)} + \ \dots \ + X_{(n,m)}.C_{(n,m)}$$

2.1 Algoritma Pemecahan Persoalan Transportasi

Algoritma Greedy dan Brute-Force dapat digunakan untuk memecahkan persoalan transportasi seimbang. Algoritma Brute-Force untuk menyelesaikan persoalan transportasi seimbang tersebut adalah, sebagai berikut:

1) Pilih kolom $X_{(i,j)}$, dimana jika langkah ini adalah langkah awal maka $i = j = 1$.
2) Bandingkan jumlah *supply* (S_i) dan jumlah *demmand* (D_j) pada kolom $X_{(i,j)}$.
3) Isi kolom $X_{(i,j)}$ dengan jumlah *supply* jika $S_i < D_j$ kemudian kurangi jumlah D_j dengan jumlah S_i (D_j - S_i) dan perbarui nilai S_i menjadi 0 serta naikkan indeks *j* sebesar 1 poin
4) Isi kolom $X_{(i,j)}$ dengan jumlah *demmand* jika $S_i > D_j$ kemudian kurangi jumlah S_i dengan jumlah D_j (S_i - D_j) dan perbarui nilai D_j dengan 0 serta naikkan indeks *i* sebesar 1 poin
5) Isi kolom $X_{(i,j)}$ dengan jumlah *demmand* (D_i) atau jumlah *supply* (S_i) jika $S_i = D_j$ kemudian perbarui nilai S_i dan D_j menjadi 0 dan naikkan indeks *i* dan indeks *j* sebanyak 1 poin.
6) Lakukan kembali langkah pertama hingga seluruh *supply* (S_i) dan *demmand* (D_j) terpenuhi atau $S_i = D_j = 0$.

Algoritma Greedy untuk menyelesaikan persoalan transportasi seimbang adalah, sebagai berikut:

1) Pilih kolom $X_{(i,j)}$ pada matriks persoalan transportasi yang memiliki biaya (*cost*) paling rendah dan $X_{(i,j)}$ belum diisi nilai apapun serta nilai S_i dan D_j tidak 0. Jika terdapat 2 atau lebih kolom yang memiliki *cost* terendah, maka kolom yang dipilih adalah kolom yang ditemukan terlebih dahulu (indeks (*i,j*) yang terendah).
2) Bandingkan jumlah *supply* (S_i) dan jumlah *demmand* (D_j) pada kolom $X_{(i,j)}$.
3) Isi kolom $X_{(i,j)}$ dengan jumlah *supply* jika $S_i < D_j$ kemudian kurangi nilai D_j dengan nilai S_i (D_j - S_i) dan ubah nilai S_i dengan 0.
4) Isi kolom $X_{(i,j)}$ dengan jumlah *demmand* jika $S_i > D_j$ kemudian kurangi nilai S_i dengan nilai D_j (S_i - D_j) dan ubah nilai D_j dengan 0.
5) Jika jumlah *supply* (S_i) sama dengan jumlah *demmand* (D_i) atau $S_i = D_j$, maka kolom $X_{(i,j)} = S_i = D_j$, kemudian ubah nilai S_i dan D_j menjadi 0.
6) Lakukan kembali langkah pertama hingga semua *supply* (S_i) dan *demmand* (D_j) terpenuhi atau $S_i = D_j = 0$.

Dalam eksperimen ini diasumsikan bahwa pada persoalan transportasi seimbang, setiap *supplier* (S_i) dapat mendistribusikan produk ke setiap *demmand* (D_i).

2.2 *Pseudocode* Algoritma Brute-Force

Pseudocode algoritma Brute-Force untuk menyelesaikan persoalan transportasi seimbang adalah sebagai berikut:

```
function transportasiBruteForce(var m:Integer, var n:Integer,
      var supply:Array, var demmand:Array)

      var i:Integer;
      var j:Integer;
      var x:Array;

      i = 0; j = 0;

      while (supply[(n-1)] != 0) and (demmand[(m-1)] != 0) do

            if supply[j] > demmand[i] then

                  x[i][j] = demmand[i];
                  supply[j] = supply[j] - demmand[i];
                  demmand[i] = 0;
                  inc i;

            else if supply[j] < demmand[i] then

                  x[i][j] = supply[j];
                  demmand[i] = demmand[i] - supply[j];
                  supply[j] = 0;
                  inc j;

            else

                  x[i][j] = supply[j];
                  supply[j] = 0;
                  supply[i] = 0;
                  inc i; inc j;

            end if

      end while

end function
```

2.3 *Pseudocode* Algoritma Greedy

Sebelum menjalankan algoritma Greedy untuk menyelesaikan persoalan transportasi seimbang, pertama array *cost* yang menjadi masukkan algoritma Greedy ini harus diurutkan terlebih dahulu dari *cost* terkecil ke *cost* terbesar. Salah satu metode pengurutan array yang dapat digunakan dalam pengurutan cost adalah Selection Sort. Pada dasarnya Selection Sort adalah algoritma Greedy untuk pengurutan array dengan kompleksitas $O(n^2)$. Pseudocode algoritma Selection Sort untuk pengurutan array *cost* adalah, sebagai berikut:

```
procedure selectionSort(var n:Interger, var cost:Array,
      var A:Array)

      var iPos:Integer; var iMin:Integer;
      for iPos = 0 to (n - 1) do
            iMin = iPos;
            for i = iPos+1 to (n - 1) do
                  if cost[i] < cost[iMin] then
                        iMin = i;
                  end if
            end for
            if iMin != iPos then
                  swap(A[iPos], A[iMin]);
            end if
      end for

end procedure
```

Variabel A dengan tipe data array berisi indeks *[i,j]* dari *cost*. Contoh isi dari array A adalah *[[1,1], [1,2], [1,3], …, [m,n]]*, dimana *m* adalah jumlah *demmand* dan *n* adalah jumlah *supply*.

Dalam pengurutan array di atas, pada dasarnya yang diurutkan bukanlah nilai dari array *cost*, melainkan indeks array *cost* pada matriks persoalan transportasi. Jadi hasil akhir array A nilai awal A adalah pada *pseudocode* di atas adalah indeks *[i,j]* yang menunjukkan kolom matriks. Setelah indeks array *cost* diurutkan dari nilai *cost* yang terkecil ke *cost* yang terbesar, selanjutnya adalah melakukan penyelesaian persoalan transportasi seimbang dengan menggunakan algoritma Greedy.

Pseudocode algoritma Greedy untuk menyelesaikan persoalan transportasi seimbang adalah sebagai berikut:

```
function transportasiGreedy(var m:Integer, var supply:Array,
      var demmand:Array, var A:Array)

      var i:Integer; var j:Integer; var k:Integer; var x:Array;
      for k = 0 to (m - 1) do
            i = A[k][1];
            j = A[k][2];
            if (supply[j] != 0) and (demmand[i] != 0) do
                  if supply[j] > demmand[i] do
                        x[(A[k])] = demmand[i];
                        supply[j] = supply[j] - demmand[i];
                        demmand[i] = 0;
                  else if supply[j] < demmand[i] do
                        x[(A[k])] = supply[j];
                        demmand[i] = demmand[i] - supply[j];
                        supply[j] = 0;
                  else
                        x[(A[k])] = supply[j];
                        demmand[i] = 0;
                        supply[j] = 0;
                  end if
            end if
      end for

end function
```

2.4 Langkah Penyelesaian dengan Algoritma Brute-Force

Contoh persoalan transportasi seimbang yang akan dipecahkan adalah, sebagai berikut:

Diberikan matriks persoalan transportasi sebagai berikut:

Supplier (Sn) Demmand (Dn)	S_1	S_2	S_3	$\sum (D_{i,j})$
D_1	$X_{(1,1)}$ 10	$X_{(1,2)}$ 2	$X_{(1,3)}$ 20	15
D_2	$X_{(2,1)}$ 12	$X_{(2,2)}$ 7	$X_{(2,3)}$ 9	25
D_3	$X_{(3,1)}$ 2	$X_{(3,2)}$ 14	$X_{(3,3)}$ 16	5
$\sum (S_{i,j})$	5	20	20	$\sum (S_{i,j}) = \sum (D_{i,j})$ 45 = 45

Gambar 2. 2 Contoh matrik persoalan transportasi seimbang

Isilah nilai dari masing-masing kolom matriks sehingga seluruh *supply* dapat memenuhi seluruh *demmand* dan cost yang dikeluarkan seminimal mungkin.

Pemecahan persoalan transportasi seimbang tersebut menggunakan algoritma Brute-Force adalah, sebagi berikut:

1) Langkah 1, isi nilai $X_{(1,1)}$ dengan nilai S_1 yaitu 5, kurangi D_1 dengan nilai S_1 sehingga *15 - 5 = 10*, dan ubah nilai S_1 dengan 0. Matriks persoalan transportasi pada langkah 1 ini akan menjadi seperti pada gambar di bawah.

Supplier (Sn) Demmand (Dn)	S_1	S_2	S_3	$\sum (D_{i,j})$
D_1	5 10	$X_{(1,2)}$ 2	$X_{(1,3)}$ 20	10
D_2	$X_{(2,1)}$ 12	$X_{(2,2)}$ 7	$X_{(2,3)}$ 9	25
D_3	$X_{(3,1)}$ 2	$X_{(3,2)}$ 14	$X_{(3,3)}$ 16	5
$\sum (S_{i,j})$	0	20	20	$\sum (S_{i,j}) = \sum (D_{i,j})$ 45 = 45

Gambar 2. 3 Matriks persoalan transportasi langkah 1 menggunakan algoritma Brute-Force

2) Langkah 2, isi nilai $X_{(1,2)}$ dengan nilai D_1 yaitu 10, kurangi S_2 dengan nilai D_1 sehingga *20 - 10 = 10*, dan ubah nilai D_1 dengan 0. Matriks persoalan transportasi pada langkah 2 ini akan menjadi seperti pada gambar di bawah.

Supplier (Sn) Demmand (Dn)	S_1	S_2	S_3	$\sum (D_{i,j})$
D_1	5 | 10	10 | 2	$X_{(1,3)}$ | 20	0
D_2	$X_{(2,1)}$ | 12	$X_{(2,2)}$ | 7	$X_{(2,3)}$ | 9	25
D_3	$X_{(3,1)}$ | 2	$X_{(3,2)}$ | 14	$X_{(3,3)}$ | 16	5
$\sum (S_{i,j})$	0	10	20	$\sum (S_{i,j}) = \sum (D_{i,j})$ 45 = 45

Gambar 2. 4 Matriks persoalan transportasi langkah 2 menggunakan algoritma Brute-Force

3) Langkah 3, isi nilai $X_{(2,2)}$ dengan nilai S_2 yaitu 10, kurangi D_2 dengan nilai S_2 sehingga *25 - 10 = 15*, dan ubah nilai S_2 dengan 0. Matriks persoalan transportasi pada langkah 3 ini akan menjadi seperti pada gambar di bawah.

Supplier (Sn) Demmand (Dn)	S_1	S_2	S_3	$\sum (D_{i,j})$
D_1	5 | 10	10 | 2	$X_{(1,3)}$ | 20	0
D_2	$X_{(2,1)}$ | 12	10 | 7	$X_{(2,3)}$ | 9	15
D_3	$X_{(3,1)}$ | 2	$X_{(3,2)}$ | 14	$X_{(3,3)}$ | 16	5
$\sum (S_{i,j})$	0	0	20	$\sum (S_{i,j}) = \sum (D_{i,j})$ 45 = 45

Gambar 2. 5 Matriks persoalan transportasi langkah 3 menggunakan algoritma Brute-Force

4) Langkah 4, isi nilai $X_{(2,3)}$ dengan nilai D_2 yaitu 15, kurangi S_3 dengan nilai D_2 sehingga *20 – 15 = 5*, dan ubah nilai D_2 dengan 0. Matriks persoalan transportasi pada langkah 4 ini akan menjadi seperti pada gambar di bawah.

Supplier (Sn) Demmand (Dn)	S_1	S_2	S_3	$\sum (D_{i,j})$
D_1	5 10	10 2	$X_{(1,3)}$ 20	0
D_2	$X_{(2,1)}$ 12	10 7	15 9	0
D_3	$X_{(3,1)}$ 2	$X_{(3,2)}$ 14	$X_{(3,3)}$ 16	5
$\sum (S_{i,j})$	0	0	5	$\sum (S_{i,j}) = \sum (D_{i,j})$ 45 = 45

Gambar 2. 6 Matriks persoalan transportasi langkah 4 menggunakan algoritma Brute-Force

5) Langkah 5, isi nilai $X_{(3,3)}$ dengan nilai S_3 atau D_3 yaitu 5, ubah nilai S_3 dan D_3 dengan 0. Matriks persoalan transportasi pada langkah 5 ini akan menjadi seperti pada gambar di bawah.

Supplier (Sn) Demmand (Dn)	S_1	S_2	S_3	$\sum (D_{i,j})$
D_1	5 10	10 2	$X_{(1,3)}$ 20	0
D_2	$X_{(2,1)}$ 12	10 7	15 9	0
D_3	$X_{(3,1)}$ 2	$X_{(3,2)}$ 14	5 16	0
$\sum (S_{i,j})$	0	0	0	$\sum (S_{i,j}) = \sum (D_{i,j})$ 45 = 45

Gambar 2. 7 Matriks persoalan transportasi langkah 5 menggunakan algoritma Brute-Force

Cost yang dikeluarkan pada hasil akhir penyelesaian persoalan transportasi menggunakan algoritma Brute-Force adalah, sebagai berikut:

$$z = \sum_{i=1}^{3}\sum_{j=1}^{3} X_{(i,j)}.C_{(i,j)}$$

$$z = (5 * 10) + (10 * 2) + (0 * 20) + (0 * 12) + (10 * 7) + (15 * 9) + (0 * 2) + (0 * 14) + (5 * 16)$$

$$z = 50 + 20 + 70 + 135 + 80$$

$$z = 355$$

2.5 Langkah penyelesaian dengan Algoritma Greedy

Dengan menggunakan contoh matriks persoalan transportasi seimbang yang digunakan pada algoritma Brute-Force, maka pemecahan persoalan tersebut dengan menggunakan algoritma Greedy adalah, sebagai berikut:

1) Langkah 1, isi nilai $X_{(1,2)}$ yaitu kolom yang memiliki *cost* terkecil dengan nilai D_1 yaitu 15, kurangi S_2 dengan nilai D_1 sehingga *20 - 5 = 5*, dan ubah nilai D_1 dengan 0. Matriks persoalan transportasi pada langkah 1 ini akan menjadi seperti pada gambar di bawah.

Supplier (Sn) Demmand (Dn)	S_1	S_2	S_3	$\sum (D_{i,j})$
D_1	$X_{(1,1)}$ 10	15 2	$X_{(1,3)}$ 20	0
D_2	$X_{(2,1)}$ 12	$X_{(2,2)}$ 7	$X_{(2,3)}$ 9	25
D_3	$X_{(3,1)}$ 2	$X_{(3,2)}$ 14	$X_{(3,3)}$ 16	5
$\sum (S_{i,j})$	5	5	20	$\sum (S_{i,j}) = \sum (D_{i,j})$

Gambar 2. 8 Matriks persoalan transportasi langkah 1 menggunakan algoritma Greedy

2) Langkah 2, isi nilai $X_{(3,1)}$ yaitu kolom yang memiliki *cost* terkecil dengan nilai S_1 yaitu 5, ubah nilai D_3 dan S_1 dengan 0. Matriks persoalan transportasi pada langkah 2 ini akan menjadi seperti pada gambar di bawah.

Supplier (Sn) Demmand (Dn)	S_1	S_2	S_3	$\sum (D_{i,j})$
D_1	$X_{(1,1)}$ 10	15 2	$X_{(1,3)}$ 20	0
D_2	$X_{(2,1)}$ 12	$X_{(2,2)}$ 7	$X_{(2,3)}$ 9	25
D_3	5 2	$X_{(3,2)}$ 14	$X_{(3,3)}$ 16	0
$\sum (S_{i,j})$	0	5	20	$\sum (S_{i,j}) = \sum (D_{i,j})$

Gambar 2. 9 Matriks persoalan transportasi langkah 2 menggunakan algoritma Greedy

3) Langkah 3, isi nilai $X_{(2,2)}$ yaitu kolom yang memiliki *cost* terkecil dengan nilai S_2 yaitu 5, ubah nilai D_2 dan S_2 dengan 0. Matriks persoalan transportasi pada langkah 3 ini akan menjadi seperti pada gambar di bawah.

Supplier (Sn) Demmand (Dn)	S_1	S_2	S_3	$\sum (D_{i,j})$
D_1	$X_{(1,1)}$ 10	15 2	$X_{(1,3)}$ 20	0
D_2	$X_{(2,1)}$ 12	5 7	$X_{(2,3)}$ 9	20
D_3	5 2	$X_{(3,2)}$ 14	$X_{(3,3)}$ 16	0
$\sum (S_{i,j})$	0	0	20	$\sum (S_{i,j}) = \sum (D_{i,j})$

Gambar 2. 10 Matriks persoalan transportasi langkah 3 menggunakan algoritma Greedy

4) Langkah 4, isi nilai $X_{(2,3)}$ yaitu kolom yang memiliki *cost* terkecil dengan nilai S_3 yaitu 20, ubah nilai D_2 dan S_3 dengan 0. Matriks persoalan transportasi pada langkah 4 ini akan menjadi seperti pada gambar di bawah.

Supplier (Sn) Demmand (Dn)	S_1	S_2	S_3	$\sum (D_{i,j})$
D_1	$X_{(1,1)}$ 10	15 2	$X_{(1,3)}$ 20	0
D_2	$X_{(2,1)}$ 12	5 7	20 9	0
D_3	5 2	$X_{(3,2)}$ 14	$X_{(3,3)}$ 16	0
$\sum (S_{i,j})$	0	0	0	$\sum (S_{i,j}) = \sum (D_{i,j})$

Gambar 2. 11 Matriks persoalan transportasi langkah 4 menggunakan algoritma Greedy

Cost yang dikeluarkan pada hasil akhir penyelesaian persoalan transportasi menggunakan algoritma Brute-Force adalah, sebagai berikut:

$$z = \sum_{i=1}^{3}\sum_{j=1}^{3} X_{(i,j)}.C_{(i,j)}$$

$$z = (0 * 10) + (15 * 2) + (0 * 20) + (0 * 12) + (5 * 7) + (20 * 9) + (5 * 2) + (0 * 14) + (0 * 16)$$

$$z = 30 + 35 + 180 + 10$$

$$z = 235$$

2.6 Kompleksitas Algoritma Brute-Force

Dengan menghitung jumlah pengulangan dari masing-masing operasi pada *preudocode* algoritma Brute-Force untuk penyelesaian persoalan transportasi seimbang dibawah ini, maka akan didapat kompleksitas waktu dari algoritma tersebut.

```
function transportasiBruteForce(var m:Integer, var n:Integer,
      var supply:Array, var demmand:Array)
      var i:Integer;
      var j:Integer;
      var x:Array;

      i = 0; j = 0;

      while (supply[(n-1)] != 0) and (demmand[(m-1)] != 0) do

            if supply[j] > demmand[i] then

                  x[i][j] = demmand[i];
                  supply[j] = supply[j] - demmand[i];
                  demmand[i] = 0;
                  inc i;

            else if supply[j] < demmand[i] then

                  x[i][j] = supply[j];
                  demmand[i] = demmand[i] - supply[j];
                  supply[j] = 0;
                  inc j;

            else

                  x[i][j] = supply[j];
                  supply[j] = 0;
                  supply[i] = 0;
                  inc i; inc j;

            end if
      end while
end function
```

Dari *pseudocode* algoritma Brute-Force di atas, didapat beberapa operasi pada tabel berikut:

Baris Instruksi	Operasi	Pengulangan
5	`i = 0;`	*1* kali
5	`j = 0;`	*1* kali
8	`x[i][j] = demmand[i];`	*(n/2)+1* kali
9	`supply[j] = supply[j] - demmand[i];`	*(n/2)+1* kali
10	`demmand[i] = 0;`	*(n/2)+1*kali
11	`inc i;`	*(n/2)+1* kali
13	`x[i][j] = supply[j];`	*(n/2)+1* kali
14	`demmand[i] = demmand[i] - supply[j];`	*(n/2)+1* kali
15	`supply[j] = 0;`	*(n/2)+1* kali
16	`inc j;`	*(n/2)+1* kali
18	`x[i][j] = supply[j];`	*(n/2)+1*kali
19	`supply[j] = 0;`	*(n/2)+1* kali
20	`supply[i] = 0;`	*(n/2)+1* kali
21	`inc i;`	*(n/2)+1*kali
21	`inc j;`	*(n/2)+1* kali

Dari tabel di atas dapat dihitung kompleksitas waktu algoritma Brute-Force untuk menyelesaikan persoalan transportasi seimbang, sebagai berikut:

$$T(n) = \sum t_i$$

$$T(n) = 1 + 1 + 13(\frac{n}{2} + 1)$$

$$T(n) = 2 + \frac{13n}{2} + 13)$$

$$T(n) = \frac{13n}{2} + 15$$

Untuk alasan penyederhanaan, pengukuran kompleksitas algoritma dapat dilakukan dengan menghitung pengulangan operasi-operasi dasar dari algoritma. Masing-masing baris kode yang berada di dalam lingkup operasi dasar dinyatakan dengan *1* dan tidak dilibatkan dalam perhitungan kompleksitas waktu. Berikut adalah operasi dasar dari algoritma Brute-Force untuk menyelesaikan persoalan transportasi seimbang:

Baris Instruksi	Operasi	Pengulangan
6	`while (supply[(n-1)] != 0) and (demmand[(m-1)] != 0) do`	*(n/2)+1* kali
7	`if supply[j] > demmand[i] then`	*(n/2)+1* kali
12	`else if supply[j] < demmand[i] then`	*(n/2)+1* kali
17	`else`	*(n/2)+1* kali

Dari tabel pengulangan operasi-operasi dasar pada algoritma Brute-Force, didapat kompleksitas waktu sebagai berikut:

$$T(n) = \sum t_i$$

$$T(n) = 4(\frac{n}{2} + 1)$$

$$T(n) = 2n + 2$$

Kasus terbaik untuk algoritma Brute-Force dalam menyelesaikan persoalan transportasi seimbang adalah jika {$S_1 = D_1, S_2 = D_2, S_3 = D_3, \ldots, S_n = D_n$}. Contoh kasus terbaik untuk algoritma Brute-Force dalam menyelesaikan persoalan transportasi adalah, sebagai berikut:

Supplier (Sn) Demmand (Dn)	S_1	S_2	S_3	$\sum (D_{i,j})$
D_1	5 | 10	$X_{(1,2)}$ | 2	$X_{(1,3)}$ | 20	5
D_2	$X_{(2,1)}$ | 12	10 | 7	$X_{(2,3)}$ | 9	10
D_3	$X_{(3,1)}$ | 2	$X_{(3,2)}$ | 14	15 | 16	15
$\sum (S_{i,j})$	5	10	15	$\sum (S_{i,j}) = \sum (D_{i,j})$ 45 = 45

Gambar 2. 12 Matriks kasus terbaik persoalan transportasi menggunakan algoritma Brute-Force

Kompleksitas waktu dengan menghitung jumlah pengulangan operasi pada algoritma Brute-Force untuk kasus terbaik ini ($T_{min}(n)$) adalah, sebagai berikut:

$$T_{min}(n) = \sum t_i$$

$$T_{min}(n) = 1 + 1 + n + n + n + n + n$$

$$T_{min}(n) = 5n + 2$$

Sedangkan kasus terburuk untuk algoritma Brute-Force dalam menyelesaikan persoalan transportasi seimbang adalah jika {$S_1 < D_1 < S_2 < D_2 < S_3 < D_3 < \ldots < S_n > D_n$} atau sebaliknya {$S_1 > D_1 > S_2 > D_2 > S_3 > D_3 > \ldots > S_n < D_n$}. Contoh kasus terburuk untuk algoritma Brute-Force dalam menyelesaikan persoalan transportasi adalah, sebagai berikut:

Supplier (Sn) Demmand (Dn)	S_1	S_2	S_3	$\sum (D_{i,j})$
D_1	19 10	1 2	$X_{(1,3)}$ 20	20
D_2	$X_{(2,1)}$ 12	13 7	2 9	15
D_3	$X_{(3,1)}$ 2	$X_{(3,2)}$ 14	10 16	10
$\sum (S_{i,j})$	19	14	12	$\sum (S_{i,j}) = \sum (D_{i,j})$ 45 = 45

Gambar 2. 13 Matriks kasus terburuk persoalan transportasi menggunakan algoritma Brute-Force

Kompleksitas waktu dengan menghitung jumlah pengulangan operasi pada algoritma Brute-Force untuk kasus terburuk ini ($T_{max}(n)$) adalah, sebagai berikut:

$$T_{max}(n) = \sum t_i$$

$$T_{max}(n) = 1 + 1 + n + n + n + n + n + n + n + n + n + n + n + n + n + n + n + n + n + n + n + n$$

$$T_{max}(n) = 21n + 2$$

Kompleksitas waktu suatu algoritma dapat juga ditentukan dengan membandingkan kompleksitas waktu algoritma tersebut dengan kompleksitas waktu yang presisi untuk suatu algoritma. Kompleksitas seperti ini disebut juga dengan kompleksitas waktu asimptotik yang dinyatakan dengan notasi O-besar (*O*). Untuk algoritma Brute-Force pada kasus pemecahan persoalan transportasi seimbang ini, kompleksitas waktu asimptotiknya adalah, sebagai berikut:

$$T(n) = \sum t_i$$

$$T(n) = 1 + 1 + 13(\frac{n}{2} + 1)$$

$$T(n) = 2 + \frac{13n}{2} + 13)$$

$$T(n) = \frac{13n}{2} + 15$$

Berdasarkan teorema: $T(n) \leq C.f(n)$,

Sehingga $\frac{13n}{2} + 15 \leq \frac{13n}{2} + 15n$,

maka: $\frac{13n}{2} + 15 = O(n)$

Jadi, kompleksitas waktu asimptotiknya adalah $O(n)$

2.7 Kompleksitas Algoritma Greedy

Dengan menghitung jumlah pengulangan dari masing-masing operasi pada *preudocode* algoritma Greedy untuk penyelesaian persoalan transportasi seimbang dibawah ini, maka akan didapat kompleksitas waktu dari algoritma tersebut.

```
function transportasiGreedy(var m:Integer, var supply:Array,
      var demmand:Array, var A:Array)
      var i:Integer; var j:Integer; var k:Integer; var x:Array;
      for k = 0 to (m - 1) do
            i = A[k][1]; j = A[k][2];
            if (supply[j] != 0) and (demmand[i] != 0) do
                  if supply[j] > demmand[i] do
                        x[(A[k])] = demmand[i];
                        supply[j] = supply[j] - demmand[i];
                        demmand[i] = 0;
                  else if supply[j] < demmand[i] do
                        x[(A[k])] = supply[j];
                        demmand[i] = demmand[i] - supply[j];
                        supply[j] = 0;
                  else
                        x[(A[k])] = supply[j];
                        demmand[i] = 0; supply[j] = 0;
                  end if
            end if
      end for
end function
```

Dari *pseudocode* algoritma Greedy di atas, didapat beberapa operasi pada tabel berikut:

Baris Instruksi	Operasi	Pengulangan
4	`i = A[k][1];`	*n* kali
4	`j = A[k][2];`	*n* kali
7	`x[(A[k])] = demmand[i];`	*n* kali
8	`supply[j] = supply[j] - demmand[i];`	*n* kali
9	`demmand[i] = 0;`	*n* kali
11	`x[(A[k])] = supply[j];`	*n* kali
12	`demmand[i] = demmand[i] - supply[j];`	*n* kali
13	`supply[j] = 0;`	*n* kali
15	`x[(A[k])] = supply[j];`	*n* kali
16	`demmand[i] = 0;`	*n* kali
27	`supply[j] = 0;`	*n* kali

Dari tabel di atas dapat dihitung kompleksitas waktu algoritma Greedy untuk menyelesaikan persoalan transportasi seimbang, sebagai berikut:

$$T(n) = \sum t_i$$

$$T(n) = n + n + n + n + n + n + n + n + n + n + n$$

$$T(n) = 11n$$

Untuk alasan penyederhanaan, pengukuran kompleksitas algoritma dapat dilakukan dengan menghitung pengulangan operasi-operasi dasar dari algoritma. Masing-masing baris kode yang berada di dalam lingkup operasi dasar dinyatakan dengan *1* dan tidak dilibatkan dalam perhitungan kompleksitas waktu. Berikut adalah operasi dasar dari algoritma Greedy untuk menyelesaikan persoalan transportasi seimbang:

Baris Instruksi	Operasi	Pengulangan
5	`if (supply[j] != 0) and (demmand[i] != 0) do`	*n* kali
6	`if supply[j] > demmand[i] do`	*n* kali
10	`else if supply[j] < demmand[i] do`	*n* kali
14	`else`	*n* kali

Dari tabel pengulangan operasi-operasi dasar pada algoritma Greedy, didapat kompleksitas waktu sebagai berikut:

$$T(n) = \sum t_i$$

$$T(n) = n + n + n + n$$

$$T(n) = 4n$$

Kasus terbaik untuk algoritma Greedy dalam menyelesaikan persoalan transportasi seimbang adalah jika {$S_1 = D_1$, $S_2 = D_2$, $S_3 = D_3$, ..., $S_n = D_n$} dan {$c_{11} = c_{22} = c_{33} = \ldots = c_{nn} < c_{12}$, c_{13}, c_{21}, ..., $c_{(n-1)n}$, $c_{n(n-1)}$}. Contoh kasus terbaik untuk algoritma Greedy dalam menyelesaikan persoalan transportasi adalah, sebagai berikut:

Supplier (Sn) Demmand (Dn)	S_1	S_2	S_3	$\sum (D_{i,j})$
D_1	5 | 1	$X_{(1,2)}$ | 2	$X_{(1,3)}$ | 20	5
D_2	$X_{(2,1)}$ | 12	10 | 1	$X_{(2,3)}$ | 9	10
D_3	$X_{(3,1)}$ | 2	$X_{(3,2)}$ | 14	15 | 1	15
$\sum (S_{i,j})$	5	10	15	$\sum (S_{i,j}) = \sum (D_{i,j})$ 45 = 45

Gambar 2. 14 Matriks kasus terbaik persoalan transportasi menggunakan algoritma Greedy

Kompleksitas waktu dengan menghitung jumlah pengulangan operasi pada algoritma Greedy untuk kasus terbaik ini ($T_{min}(n)$) adalah, sebagai berikut:

$$T_{min}\,(n) = \sum t_i$$

$$T_{min}\,(n) = n + n + n + n + n$$

$$T_{min}\,(n) = \ 5n$$

Kompleksitas waktu suatu algoritma dapat juga ditentukan dengan membandingkan kompleksitas waktu algoritma tersebut dengan kompleksitas waktu yang presisi untuk suatu algoritma. Kompleksitas seperti ini disebut juga dengan kompleksitas waktu asimptotik yang dinyatakan dengan notasi O-besar (*O*). Untuk algoritma Greedy pada kasus pemecahan persoalan transportasi seimbang ini, kompleksitas waktu asimptotiknya adalah, sebagai berikut:

$$T(n) = \sum t_i$$

$$T(n) = n + n + n + n + n + n + n + n + n + n + n$$

$$T(n) = \ 11n$$

Berdasarkan teorema: $T(n) \leq C.f(n)$,

sehingga $11n \ \leq 11n$, maka: $11n = O(n)$

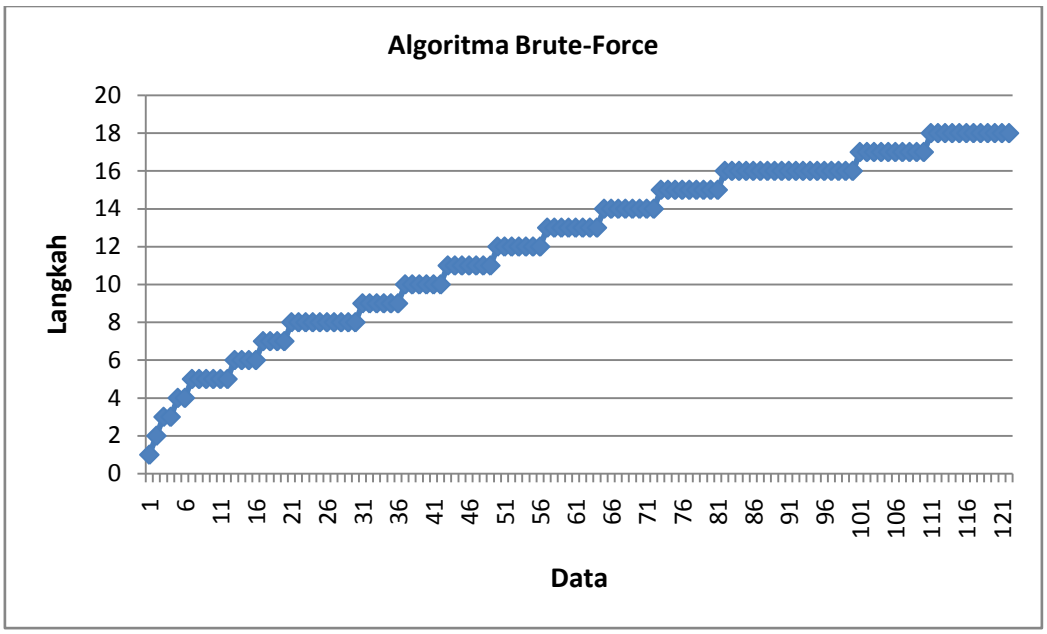
Jadi, kompleksitas waktu asimptotiknya adalah $O(n)$

Karena algoritma Greedy ini diawali dengan melakukan pengurutan pada Array, maka kompleksitas waktu algoritma Greedy ini dibandingkan dengan kompleksitas waktu asimptotik algoritma sorting yang digunakan yaitu selection sort dengan kompleksitas $O(n^2)$.

$$max(O(n), O(n^2)) = \ O(n^2)$$

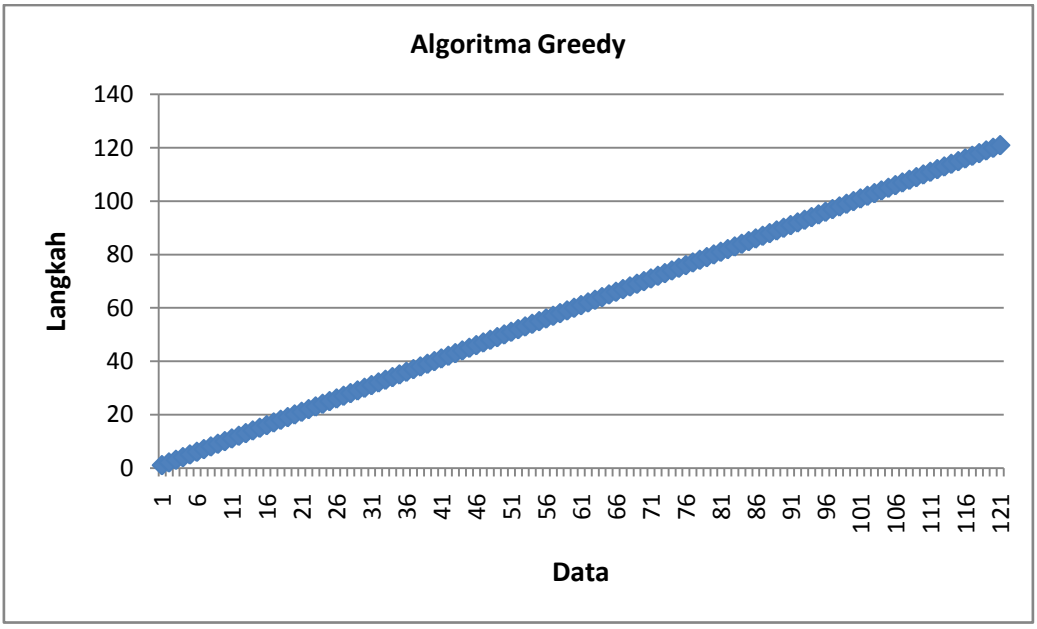
2.8 Grafik Percobaan

Hasil pengujian menggunakan algoritma Brute-Force dengan sejumlah n data menghasilkan grafik berikut ini:



Gambar 2. 15 Grafik hasil percobaan menggunakan algoritma Brute-Force

Hasil pengujian menggunakan algoritma Greedy dengan sejumlah n data menghasilkan grafik berikut ini:



Gambar 2. 16 Grafik hasil percobaan menggunakan algoritma Greedy

3. Persoalan Pewarnaan Graf

Persoalan pewarnaan graf yang akan dibahas adalah pewarnaan simpul (*vertex*) graf. Dasar permasalahan dari pewarnaan simpul graf adalah bagaimana memberi warna pada simpul-simpul dalam graf sedemikian sehingga setiap 2 simpul yang bertetangga tidak boleh memiliki warna yang sama (setiap simpul yang bertetangga memiliki warna yang berbeda). Penyelesaian persoalan pewarnaan graf bukan hanya mewarnai semua simpul yang bertetangga dengan warna yang berbeda, tapi juga menghasilkan jumlah variasi warna yang lebih sedikit. Jumlah warna minimum yang dapat digunakan untuk mewarnai simpul pada graf disebut dengan bilangan kromatik. Algoritma yang baik adalah algoritma yang menemukan bilangan kromatik dari graf tersebut.

3.1 Algoritma Pemecahan Persoalan Pewarnaan Graf

Algoritma Greedy dan Brute-Force dapat digunakan untuk memecahkan persoalan pewarnaan graf. Algoritma Brute-Force adalah algoritma yang paling lempang untuk menyelesaikan persoalan pewarnaan graf, algoritma ini menyelesaikan masalah dengan cara sederhana sesuai dengan pernyataan masalah. Langkah-langkah algoritma Brute-Force untuk persoalan pewarnaan graf adalah, sebagai berikut:

1) Pilih simpul 1 yaitu simpul yang dianggap sebagai titik awal dari graf.
2) Warnai simpul 1 dengan satu warna.
3) Ambil simpul berikutnya, cek apakah simpul tersebut bertetangga dengan simpul yang sudah memiliki warna.
4) Jika simpul tersebut bertetangga dengan simpul yang sudah diberi warna, maka warnai simpul tersebut dengan warna yang berbeda dengan simpul tetangganya.
5) Jika simpul tersebut tidak bertetangga dengan simpul yang sudah diberi warna, maka warnai simpul tersebut dengan satu warna yang sama dengan warna pada simpul 1.
6) Lakukan langkah 3 hingga simpul-simpul pada graf sudah diberi warna seluruhnya.

Algoritma Greedy untuk menyelesaikan persoalan pewarnaan graf salah satunya adalah algoritma Welch-Powell. Langkah-langkah algoritma Welch-Powell adalah, sebagai berikut:

1) Urutkan simpul-simpul pada graf dari simpul yang berderajad tinggi (simpul yang memiliki tetangga atau cabang yang lebih banyak) hingga simpul yang berderajad rendah (simpul yang memiliki tetangga atau cabang yang lebih sedikit).
2) Ambil simpul yang memilik derajad paling tinggi kemudian gunakan satu warna untuk mewarnai simpul tersebut.
3) Ambil simpul berikutnya, cek apakah simpul tersebut bertetangga dengan simpul yang sudah memiliki warna.
4) Jika simpul tersebut bertetangga dengan simpul yang sudah diberi warna, maka warnai simpul tersebut dengan warna yang berbeda dengan simpul tetangganya.
5) Jika simpul tersebut tidak bertetangga dengan simpul yang sudah diberi warna, maka warnai simpul tersebut dengan satu warna yang sama dengan warna pada simpul pertama.
6) Lakukan langkah 3 hingga simpul-simpul pada graf sudah diberi warna seluruhnya.

3.2 *Pseudocode* Algoritma Brute-Force

Pseudocode algoritma Brute-Force untuk menyelesaikan persoalan pewarnaan graf adalah, sebagai berikut:

```
function pewarnaanGrafBruteForce(var n:Integer, var Graf:Array)

      var i:Integer; var j:Integer; var k:Integer;
      var indexWarna:Integer; var indexGraf: Integer;
      var warnaGraf:Array; var warna:Array
      indexWarna = 1;
      warna.push(indexWarna);
      for i = 0 to (n-1) do
            for j = 0 to (Graf[i].length - 1) do
                  for k = 0 to (warna.lenght - 1) do
                        indexGraf = Graf[i][j];
                        if warnaGraf[indexGraf] = warna[k] then
                              inc indexWarna;
                        end if
                  end for
                  if indexWarna > warna[(warna.length - 1)] then
                        warna.Push(indexWarna);
                  end if
            end for
            warnaGraf[i] = warna[(warna.length - 1)];
      end for

end function
```

3.3 *Pseudocode* Algoritma Greedy

Sebelum menjalankan algoritma Greedy dalam menyelesaikan persoalan pewarnaan graf, data array Graf terlebih dahulu melalui proses pengurutan. Proses pengurutan yang dapat digunakan adalah Selection Sort. Selection Sort merupakan teknik pengurutan array yang tergolong algoritma Greedy. Kompleksitas algoritma selection sort adalah $O(n^2)$. *Pseudocode* algoritma Selection Sort adalah, sebagai berikut:

```
procedure selectionSort(var n:Interger, var derajadGraf:Array,
      var Graf:Array)

      var iPos:Integer;
      var iMax:Integer;

      for iPos = 0 to (n - 1) do
            iMax = iPos;
            for i = iPos+1 to n do
                  if derajadGraf[i] > derajadGraf [iMax] then
                        iMax = i;
                  end if
            end for
            if iMax != iPos then
                  swap(Graf[iPos], Graf[iMax]);
            end if
      end for

end procedure
```

Dalam pengurutan array di atas, yang diurutkan adalah nilai dari array Graf dengan cara membandingkan derajad dari graf tersebut. Setelah array graf diurutkan dari nilai graf dengan

derajad yang terbesar ke graf dengan derajad yang terkecil, selanjutnya adalah melakukan penyelesaian persoalan pewarnaan graf dengan menggunakan algoritma Greedy.

Pseudocode algoritma Greedy untuk menyelesaikan persoalan pewarnaan graf adalah, sebagai berikut:

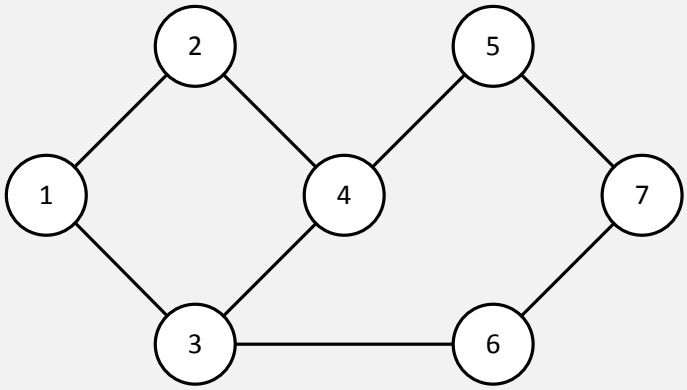
```
function pewarnaanGrafGreedy(var n:Integer, var Graf:Array)

      var i:Integer;
      var j:Integer;
      var k:Integer;
      var indexWarna:Integer;
      var indexGraf: Integer;
      var warnaGraf:Array;
      var warna:Array

      indexWarna = 1;
      warna.push(indexWarna);

      for i = 0 to (n-1) do

            for j = 0 to (Graf[i].length - 1) do

                  for k = 0 to (warna.lenght - 1) do

                        indexGraf = Graf[i][j];
                        if warnaGraf[indexGraf] = warna[k] then

                              inc indexWarna;

                        end if

                  end for

                  if indexWarna > warna[(warna.length - 1)] then

                        warna.Push(indexWarna);

                  end if

            end for

            warnaGraf[i] = warna[(warna.length - 1)];

      end for

end function
```

Algoritma Greedy untuk menyelesaikan persoalan pewarnaan graf sama dengan algoritma Brute-Force, yang membedakannya adalah pada algoritma Greedy sebelumnya dilakukan pengurutan graf dari yang berderajad tinggi ke graf yang berderajad rendah menggunakan Selection Sort.

3.4 Langkah Penyelesaian dengan Algoritma Brute-Force

Contoh persoalan pewarnaan graf yang akan dipecahkan adalah, sebagai berikut:

Diberikan graf sebagai berikut:

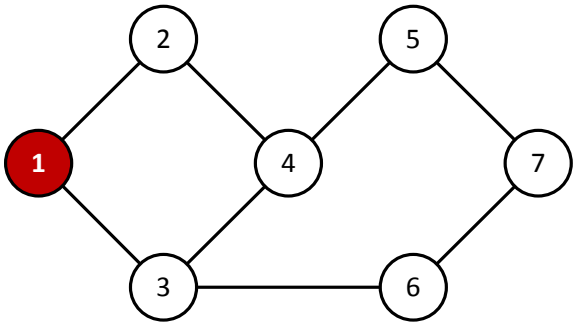


Gambar 3. 1 Contoh graf planar yang terdiri dari 7 simpul (vertex)

Warnai setiap simpul dari graf sehingga masing-masing simpul memiliki warna yang berbeda dengan tetangganya.

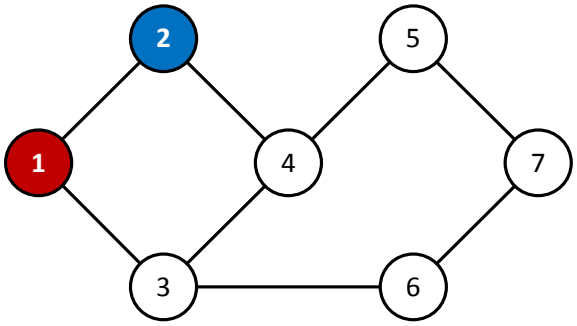
Pemecahan persoalan pewarnaan graf tersebut menggunakan algoritma Brute-Force adalah, sebagai berikut:

1) Langkah 1, warnai simpul 1 dengan satu warna (misalnya warna merah). Graf pada langkah 1 ini akan menjadi sebagai berikut:



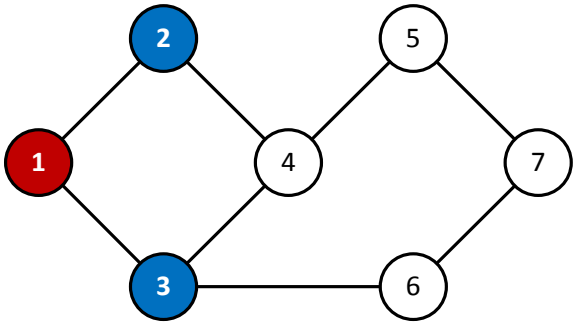
Gambar 3. 2 Warna simpul graf langkah 1 menggunakan algoritma Brute-Force

2) Langkah 2, ambil simpul 2 pada graf. Karena simpul 2 bertetangga dengan graf berwarna merah, maka gunakan warna lain untuk mewarnai simpul 2 (misalnya warna biru). Graf pada langkah 2 ini akan menjadi sebagai berikut:



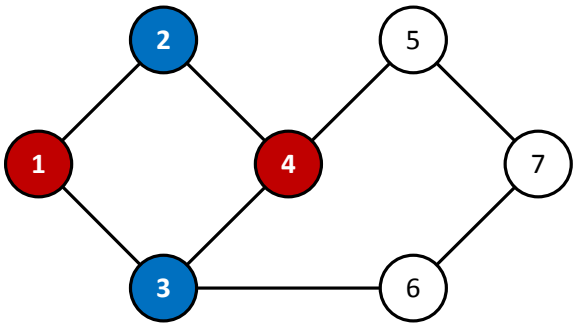
Gambar 3. 3 Warna simpul graf langkah 2 menggunakan algoritma Brute-Force

3) Langkah 3, ambil simpul 3 pada graf. Karena simpul 3 bertetangga dengan graf berwarna merah (warna pertama yang diambil) namun tidak bertetangga dengan graf dengan warna biru (warna yang kudua), maka gunakan warna biru ke untuk mewarnai simpul 3. Graf pada langkah 3 ini akan menjadi sebagai berikut:



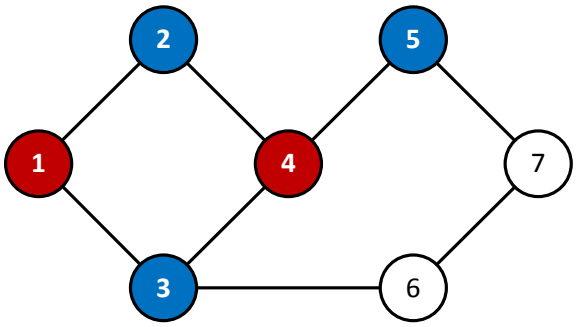
Gambar 3. 4 Warna simpul graf langkah 3 menggunakan algoritma Brute-Force

4) Langkah 4, ambil simpul 4 pada graf. Karena simpul 4 tidak bertetangga dengan graf berwarna merah yaitu warna yang diambil pertama kali, maka simpul 4 dapat diwarnai kembali dengan warna merah. Graf pada langkah 4 ini akan menjadi sebagai berikut:



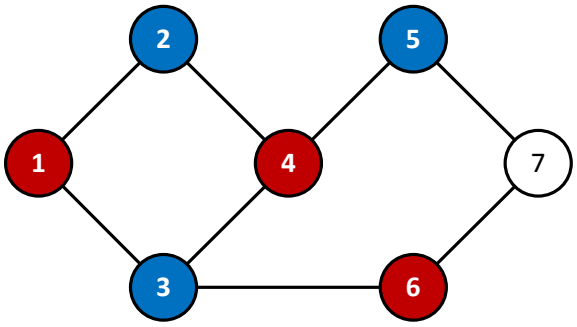
Gambar 3. 5 Warna simpul graf langkah 4 menggunakan algoritma Brute-Force

5) Langkah 5, ambil simpul 5 pada graf. Karena simpul 5 bertetangga dengan graf berwarna merah yaitu warna yang diambil pertama kali namun tidak bertetangga dengan graf berwarna biru yang merupakan warna kedua yang diambil, maka simpul 5 dapat diwarnai dengan warna biru. Graf pada langkah 5 ini akan menjadi sebagai berikut:



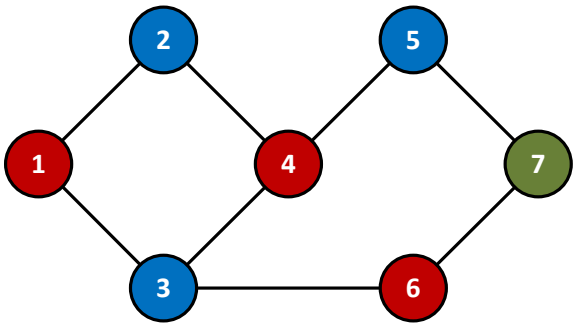
Gambar 3. 6 Warna simpul graf langkah 5 menggunakan algoritma Brute-Force

6) Langkah 6, ambil simpul 6 pada graf. Karena simpul 6 tidak bertetangga dengan graf berwarna merah yaitu warna yang diambil pertama kali, maka simpul 6 dapat diwarnai kembali dengan warna merah. Graf pada langkah 6 ini akan menjadi sebagai berikut:



Gambar 3. 7 Warna simpul graf langkah 6 menggunakan algoritma Brute-Force

7) Langkah 7, ambil simpul 7 pada graf. Karena simpul 7 bertetangga dengan graf berwarna merah yaitu warna yang diambil pertama kali dan bertangga juga dengan graf berwarna biru yaitu warna yang kedua diambil, maka simpul 7 harus diwarnai dengan warna yang berbeda dengan warna merah dan biru (misal warna hijau). Graf pada langkah 7 ini akan menjadi sebagai berikut:



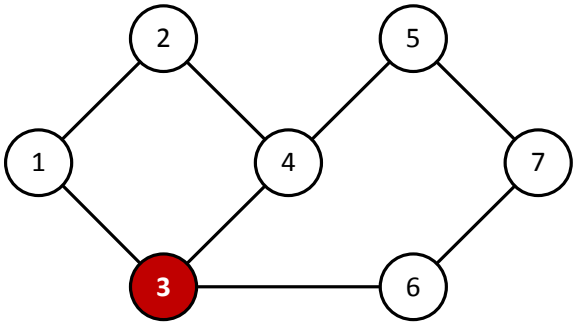
Gambar 3. 8 Warna simpul graf langkah 7 menggunakan algoritma Brute-Force

Warna yang dibutuhkan dalam penyelesaian persoalan pewarnaan graf menggunakan algoritma Brute-Force adalah 3 warna, yaitu warna merah, biru dan hijau.

3.5 Langkah Penyelesaian dengan Algoritma Greedy

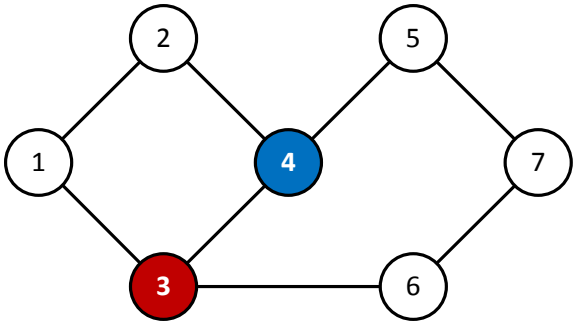
Menggunakan contoh persoalan pewarnaan graf yang sama, maka pemecahan persoalan pewarnaan graf tersebut menggunakan algoritma Greedy adalah, sebagai berikut:

1) Langkah 1, urutkan graf berdasarkan derajad tertinggi hingga derajad terendah. Urutan pewarnaan simpul graf adalah {3, 4, 1, 2, 5, 6, 7} .
2) Langkah 2, ambil simpul 3 dengan satu warna (misal warna merah). Graf pada langkah 2 akan menjadi sebagai berikut:



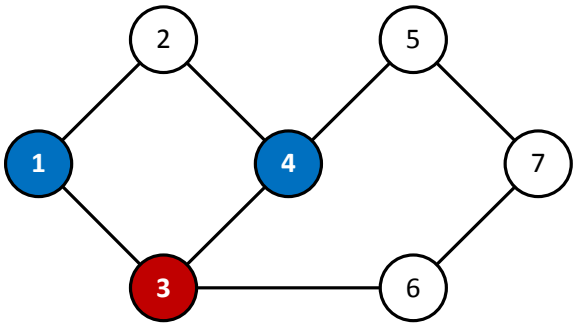
Gambar 3. 9 Warna simpul graf langkah 2 menggunakan algoritma Greedy

3) Langkah 3, ambil simpul 4 pada graf. Karena simpul 4 bertetangga dengan simpul yang berwarna merah (warna yang pertama diambil), maka warnai simpul 4 dengan warna yang berbeda dengan warna merah (misalnya warna biru). Graf pada langkah 3 akan menjadi sebagai berikut:



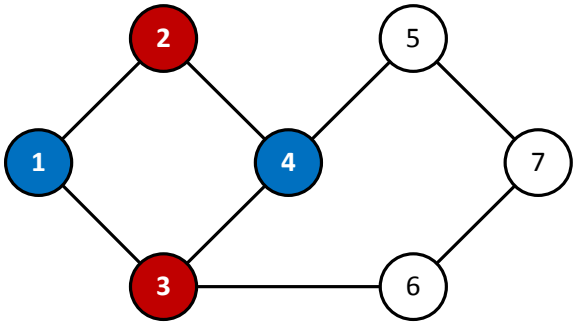
Gambar 3. 10 Warna simpul graf langkah 2 menggunakan algoritma Greedy

4) Langkah 4, ambil simpul 1 pada graf. Karena simpul 1 bertetangga dengan simpul yang memiliki warna merah (warna pertama yang diambil) namun tidak bertetangga dengan simpul berwarna biru (warna kedua yang diambil), maka simpul 1 dapat diwarnai dengan warna biru. Graf pada langkah 4 akan menjadi sebagai berikut:



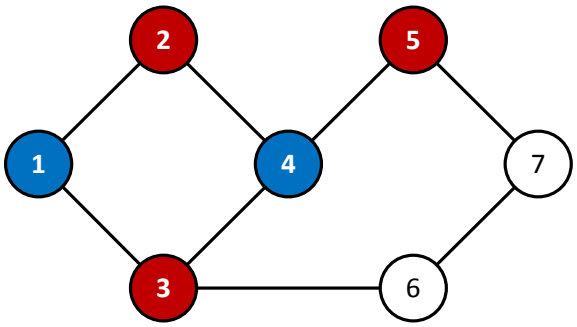
Gambar 3. 11 Warna simpul graf langkah 2 menggunakan algoritma Greedy

5) Langkah 5, ambil simpul 2 pada graf. Karena simpul 2 tidak bertetangga dengan warna merah (warna yang pertama diambil), maka simpul 2 dapat diwarnai dengan warna merah. Graf pada langkah 5 akan menjadi sebagai berikut:



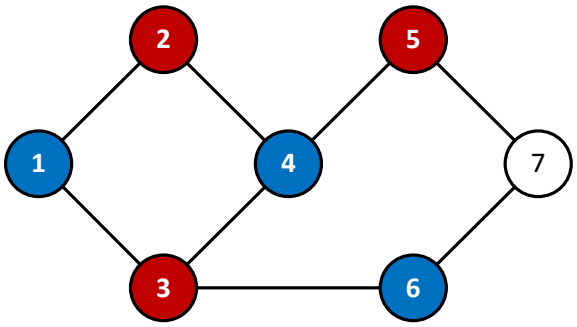
Gambar 3. 12 Warna simpul graf langkah 2 menggunakan algoritma Greedy

6) Langkah 6, ambil simpul 5 pada graf. Karena simpul 5 tidak bertetangga dengan graf yang memiliki warna merah (warna yang pertama diambil), maka simpul 5 dapat diwarnai dengan warna merah. Graf pada langkah 6 akan menjadi sebagai berikut:



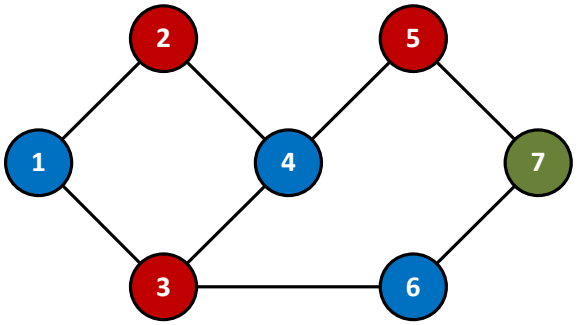
Gambar 3. 13 Warna simpul graf langkah 2 menggunakan algoritma Greedy

7) Langkah 7, ambil simpul 6 pada graf. Karena simpul 6 bertetangga dengan simpul berwarna merah (warna yang pertama diambil) namun tidak bertetangga dengan simpul berwarna biru (warna kedua yang diambil), maka simpul 6 dapat diwarnai dengan warna biru. Maka graf pada langkah 7 akan menjadi sebagai berikut:



Gambar 3. 14 Warna simpul graf langkah 2 menggunakan algoritma Greedy

8) Langkah 8, ambil simpul 7 pada graf. Karena simpul 7 bertetangga dengan graf berwarna merah yaitu warna yang diambil pertama kali dan bertangga juga dengan graf berwarna biru yaitu warna yang kedua diambil, maka simpul 7 harus diwarnai dengan warna yang berbeda dengan warna merah dan biru (misal warna hijau). Graf pada langkah 7 ini akan menjadi sebagai berikut:



Gambar 3. 15 Warna simpul graf langkah 2 menggunakan algoritma Greedy

Warna yang dibutuhkan dalam penyelesaian persoalan pewarnaan graf menggunakan algoritma Greedy adalah 3 warna, yaitu warna merah, biru dan hijau.

3.6 Kompleksitas Algoritma Brute-Force

Dengan menghitung jumlah pengulangan dari masing-masing operasi pada *preudocode* algoritma Brute-Force untuk penyelesaian persoalan pewarnaan graf dibawah ini, maka akan didapat kompleksitas waktu dari algoritma tersebut.

```
function pewarnaanGrafBruteForce(var n:Integer, var Graf:Array)

      var i:Integer; var j:Integer; var k:Integer;
      var indexWarna:Integer; var indexGraf: Integer;
      var warnaGraf:Array; var warna:Array

      indexWarna = 1;  warna.push(indexWarna);
      for i = 0 to (n-1) do
            for j = 0 to (Graf[i].length - 1) do
                  for k = 0 to (warna.lenght - 1) do
                        indexGraf = Graf[i][j];
                        if warnaGraf[indexGraf] = warna[k] then
                              inc indexWarna;
                        end if
                  end for
                  if indexWarna > warna[(warna.length - 1)] then
                        warna.Push(indexWarna);
                  end if
            end for
            warnaGraf[i] = warna[(warna.length - 1)];
      end for
end function
```

Dari *pseudocode* algoritma Brute-Force di atas, didapat beberapa operasi pada tabel berikut:

Baris Instruksi	Operasi	Pengulangan
4	`indexWarna = 1;`	*1* kali
8	`warna.push(indexWarna);`	*1* kali
10	`indexGraf = Graf[i][j];`	n^3 kali
14	`inc indexWarna;`	n^3 kali
15	`warna.Push(indexWarna);`	n^2 kali
18	`warnaGraf[i] = warna[(warna.length - 1)];`	*n* kali

Dari tabel di atas dapat dihitung kompleksitas waktu algoritma Brute-Force untuk menyelesaikan persoalan pewarnaan graf, sebagai berikut:

$$T(n) = \sum t_i$$

$$T(n) = 1 + 1 + n^3 + n^3 + n^2 + n$$

$$T(n) = (1 + 1) + (n^3 + n^3) + n^2 + n$$

$$T(n) = 2n^3 + n^2 + n + 2$$

Untuk alasan penyederhanaan, pengukuran kompleksitas algoritma dapat dilakukan dengan menghitung pengulangan operasi-operasi dasar dari algoritma. Masing-masing baris kode yang berada di dalam lingkup operasi dasar dinyatakan dengan *1* dan tidak dilibatkan dalam perhitungan kompleksitas waktu. Berikut adalah operasi dasar dari algoritma Brute-Force untuk menyelesaikan persoalan pewarnaan graf:

Baris Instruksi	Operasi	Pengulangan
10	`if warnaGraf[indexGraf] = warna[k] then`	n^3 kali
14	`if indexWarna > warna[(warna.length - 1)] then`	n^2 kali

Dari tabel pengulangan operasi-operasi dasar pada algoritma Brute-Force, didapat kompleksitas waktu, sebagai berikut:

$$T(n) = \sum t_i$$

$$T(n) = n^3 + n^2$$

Kasus terbaik untuk algoritma Brute-Force dalam menyelesaikan persoalan pewarnaan graf adalah jika *n = 2*, karena syarat pewarnaan graf adalah *n > 1*. Kompleksitas waktu dengan menghitung jumlah pengulangan operasi pada algoritma Brute-Force untuk kasus terbaik ini ($T_{min}(n)$) adalah, sebagai berikut:

$$T_{min}\,(n) = \sum t_i$$

$$T_{min}\,(n) = n^3 + n^2$$

Sedangkan kasus terburuk untuk algoritma Brute-Force dalam menyelesaikan persoalan pewarnaan graf belum dapat diketahui.

Kompleksitas waktu suatu algoritma dapat juga ditentukan dengan membandingkan kompleksitas waktu algoritma tersebut dengan kompleksitas waktu yang presisi untuk suatu algoritma. Kompleksitas seperti ini disebut juga dengan kompleksitas waktu asimptotik yang dinyatakan dengan notasi O-besar (*O*). Untuk algoritma Brute-Force pada kasus pewarnaan graf ini, kompleksitas waktu asimptotiknya adalah, sebagai berikut:

$$T(n) = \sum t_i$$

$$T(n) = 1 + 1 + n^3 + n^3 + n^2 + n$$

$$T(n) = (1 + 1) + (n^3 + n^3) + n^2 + n$$

$$T(n) = \ 2n^3 + n^2 + n + 2$$

Berdasarkan teorema: $T(n) \leq C.f(n)$,

sehingga $2n^3 + n^2 + n + 2 \leq 2n^3 + n^3 + n^3 + n^3$,

maka: $2n^3 + n^2 + n + 2 = O(n^3)$

Jadi, kompleksitas waktu asimptotiknya adalah $O(n^3)$

3.7 Kompleksitas Algoritma Greedy

Dengan menghitung jumlah pengulangan dari masing-masing operasi pada *preudocode* algoritma Greedy untuk penyelesaian persoalan pewarnaan graf dibawah ini, maka akan didapat kompleksitas waktu dari algoritma tersebut.

```
function pewarnaanGrafGreedy(var n:Integer, var Graf:Array)

      var i:Integer; var j:Integer; var k:Integer;
      var indexWarna:Integer; var indexGraf: Integer;
      var warnaGraf:Array; var warna:Array

      indexWarna = 1;  warna.push(indexWarna);
      for i = 0 to (n-1) do
            for j = 0 to (Graf[i].length - 1) do
                  for k = 0 to (warna.lenght - 1) do
                        indexGraf = Graf[i][j];
                        if warnaGraf[indexGraf] = warna[k] then
                              inc indexWarna;
                        end if
                  end for
                  if indexWarna > warna[(warna.length - 1)] then
                        warna.Push(indexWarna);
                  end if
            end for
            warnaGraf[i] = warna[(warna.length - 1)];
      end for
end function
```

Dari *pseudocode* algoritma Brute-Force di atas, didapat beberapa operasi pada tabel berikut:

Baris Instruksi	Operasi	Pengulangan
4	`indexWarna = 1;`	*1* kali
8	`warna.push(indexWarna);`	*1* kali
10	`indexGraf = Graf[i][j];`	n^3 kali
14	`inc indexWarna;`	n^3 kali
15	`warna.Push(indexWarna);`	n^2 kali
18	`warnaGraf[i] = warna[(warna.length - 1)];`	*n* kali

Dari tabel di atas dapat dihitung kompleksitas waktu algoritma Greedy untuk menyelesaikan persoalan pewarnaan graf, sebagai berikut:

$$T(n) = \sum t_i$$

$$T(n) = 1 + 1 + n^3 + n^3 + n^2 + n$$

$$T(n) = (1 + 1) + (n^3 + n^3) + n^2 + n$$

$$T(n) = 2n^3 + n^2 + n + 2$$

Untuk alasan penyederhanaan, pengukuran kompleksitas algoritma dapat dilakukan dengan menghitung pengulangan operasi-operasi dasar dari algoritma. Masing-masing baris kode yang berada di dalam lingkup operasi dasar dinyatakan dengan *1* dan tidak dilibatkan dalam perhitungan kompleksitas waktu. Berikut adalah operasi dasar dari algoritma Greedy untuk menyelesaikan persoalan pewarnaan graf:

Baris Instruksi	Operasi	Pengulangan
10	`if warnaGraf[indexGraf] = warna[k] then`	n^3 kali
14	`if indexWarna > warna[(warna.length - 1)] then`	n^2 kali

Dari tabel pengulangan operasi-operasi dasar pada algoritma Greedy, didapat kompleksitas waktu sebagai berikut:

$$T(n) = \sum t_i$$

$$T(n) = n^3 + n^2$$

Kasus terbaik untuk algoritma Greedy dalam menyelesaikan persoalan pewarnaan graf adalah jika *n = 2*, karena syarat pewarnaan graf adalah *n > 1*. Kompleksitas waktu dengan menghitung jumlah pengulangan operasi pada algoritma Greedy untuk kasus terbaik ini ($T_{min}(n)$) adalah, sebagai berikut:

$$T_{min}\,(n) = \sum t_i$$

$$T_{min}\,(n) = n^3 + n^2$$

Sedangkan kasus terburuk untuk algoritma Greedy dalam menyelesaikan persoalan pewarnaan graf belum dapat diketahui.

Kompleksitas waktu suatu algoritma dapat juga ditentukan dengan membandingkan kompleksitas waktu algoritma tersebut dengan kompleksitas waktu yang presisi untuk suatu algoritma. Kompleksitas seperti ini disebut juga dengan kompleksitas waktu asimptotik yang dinyatakan dengan notasi O-besar (*O*). Untuk algoritma Greedy pada kasus pewarnaan graf ini, kompleksitas waktu asimptotiknya adalah, sebagai berikut:

$T(n) = \sum t_i$

$T(n) = 1 + 1 + n^3 + n^3 + n^2 + n$

$T(n) = (1 + 1) + (n^3 + n^3) + n^2 + n$

$T(n) = 2n^3 + n^2 + n + 2$

Berdasarkan teorema: $T(n) \leq C.f(n)$,

sehingga $2n^3 + n^2 + n + 2 \leq 2n^3 + n^3 + n^3 + n^3$,

maka: $2n^3 + n^2 + n + 2 = O(n^3)$

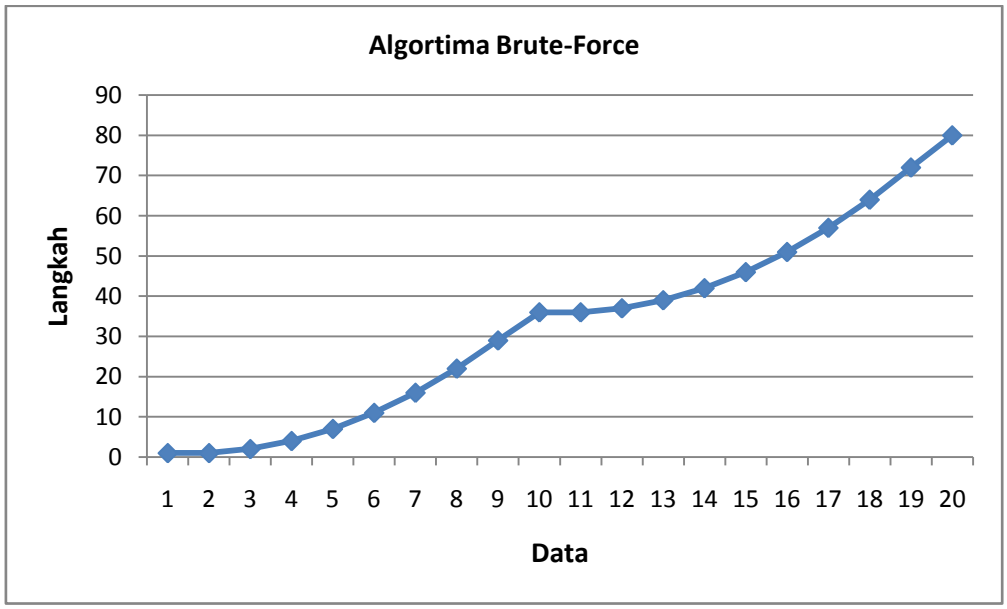
Jadi, kompleksitas waktu asimptotiknya adalah $O(n^3)$

Karena algoritma Greedy ini diawali dengan melakukan pengurutan pada Array, maka kompleksitas waktu algoritma Greedy ini dibandingkan dengan kompleksitas waktu asimptotik algoritma sorting yang digunakan yaitu selection sort dengan kompleksitas $O(n^2)$.

$max(O(n^3), O(n^2)) = O(n^3)$

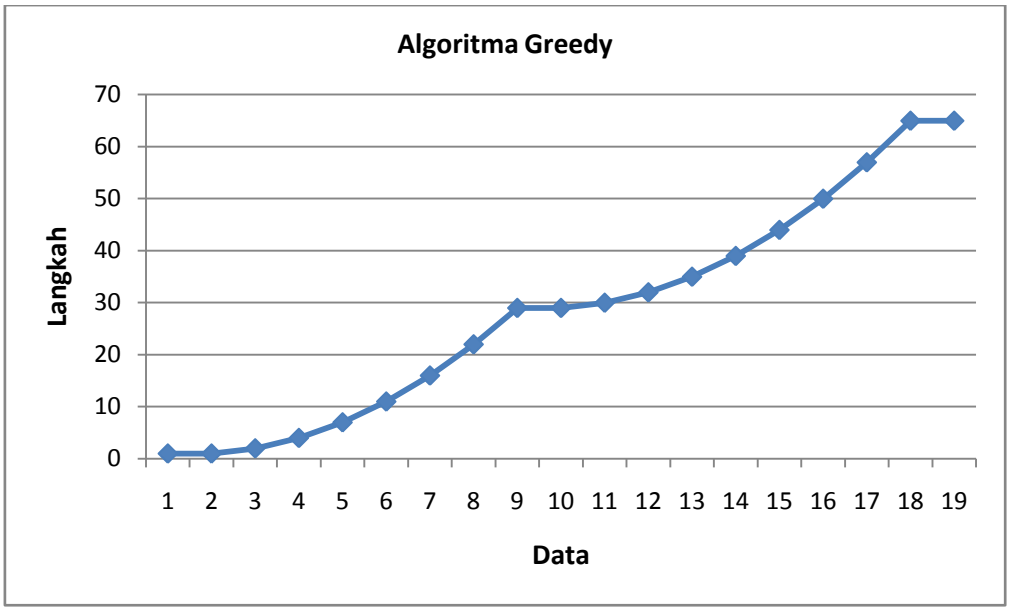
3.8 Grafik Percobaan

Hasil percobaan penyelesaian persoalan pewarnaan graf menggunakan algoritma Brute-Force dapat dilihat pada grafik berikut ini:



Gambar 3. 16 Grafik percobaan menggunakan algoritma Brute-Force

Hasil percobaan penyelesaian persoalan pewarnaan graf menggunakan algoritma Greedy dapat dilihat pada grafik berikut ini:



Gambar 3. 17 Grafik percobaan menggunakan algoritma Greedy

4. Kombinasi 5 Kartu pada Permainan Poker

Permainan poker adalah permainan judi populer yang menggunakan media kartu remi. Permainan poker pada dasarnya merupakan permainan yang membandingkan kombinasi kartu remi berdasarkan corak dan nilainya, yang terbentuk dengan maksimal kombinasi terdiri dari 5 kartu dan minimal adalah tanpa kombinasi (satu kartu). Berikut adalah urutan kombinasi kartu poker dari kombinasi dengan jumlah kartu paling sedikit dan derajad terkecil hingga kombinasi dengan jumlah kartu terbanyak dan derajad terbesar:

1) *Single*, yaitu hanya terdapat satu kartu saja dan tidak membentuk kombinasi.
2) *Pair*, yaitu kombinasi dari 2 kartu yang memiliki nilai sama (misalnya 8 sekop dan 8 wajik).
3) *Two Pair*, yaitu kombinasi dari 4 kartu dimana terdapat 2 pasang kombinasi *pair* (misalnya 2 pasang kartu yg sama misalnya 8 sekop dan 8 wajik serta 6 hati dan 6 keriting).
4) *Triple*, yaitu kombinasi dari 3 kartu yang memiliki nilai sama (misal ada 3 kartu dengan nilai sama 8 sekop, 8 hati dan 8 keriting).
5) *Straight*, yaitu kombinasi dari 5 kartu dengan nilai yang membentuk deret (misalnya kartu berurutan 3 hati, 4 sekop, 5 sekop, 6 wajik dan 7 hati).
6) *Flush*, yaitu kombinasi 5 kartu dengan corak yang sama (misal terdapat 5 kartu hati dengan nilai yang tidak berurutan 4 hati, as hati, queen hati, 7 hati dan 2 hati)
7) *Full house*, yaitu kombinasi dari 5 kartu yanng terdiri dari sebuah kombinasi *triple* dan sebuah kombinasi *one pair* (misalnya kombinasi *triple* 8 hati, 8 sekop dan 8 wajik dengan kombinasi *one pair* 6 wajik dan 6 keriting).
8) *Four of a kind*, yaitu kombinasi 4 kartu yang memiliki nilai sama (misal terdapat 4 kartu 8 sekop, 8 hati, 8 keriting dan 8 wajik)

9) *Straight flush*, yaitu kombinasi *straight* yang memiliki corak yang sama (misalnya kartu 3, 4, 5, 6 dan 7 dengan corak hati)

Derajad kombinasi kartu pada permainan poker juga ditentukan oleh corak kartu yang terdapat pada kombinasi. Urutan corak kartu dari derajad terendah ke derajad tertinggi adalah, sebagai berikut:

1) *Diamond* atau wajik berwarna merah,
2) *Keriting* berwarna hitam,
3) *Love* atau hati berwarna merah, dan
4) *Spade* atau sekop berwarna hitam.

Urutan nilai kartu remi dari nilai terkecil ke nilai terbesar adalah {, A (*As*) dan 2 (disebut juga *Poker*),3, 4, 5, 6, 7, 8, 9, 10, J (Jack), Q (*Queen*), K (*King*) }. Jika terdapat kombinasi kartu yang sama dan kartu dengan nilai terbesar dari kombinasi tersebut pun sama, maka derajad tertinggi ditentukan dari corak kartu dengan nilai terbesar dari masing-masing kombinasi.

Kombinasi kartu yang akan dicari dibatasi 5 kartu, dengan demikian eksperimen ini hanya akan mencari kombinasi *straight flush*, *full house*, *flush* dan *straight*. Pada eksperimen ini akan diberikan sebanyak *n* kartu secara acak dengan urutan yang acak pula, dimana *n* adalah {5, …, 52 (maksimum jumlah kartu dalam 1 set kartu remi)}. Dengan menggunakan algoritma Greedy dan Brute-Force akan dibentuk kombinasi dari sejumlah *n* kartu remi tersebut.

4.1 Algoritma yang Digunakan pada Kombinasi 5 Kartu pada Permainan Poker

Algoritma Greedy dan Brute-Force dapat digunakan untuk mencari kombinasi kartu pada permainan poker. Algoritma Brute-Force adalah algoritma yang paling lempang untuk menyelesaikan persoalan pencarian kombinasi kartu pada permainan poker. Langkah-langkah algoritma Brute-Force untuk pencarian kombinasi 5 kartu pada permainan poker adalah, sebagai berikut:

1) Ambil sejumlah *n* kartu secara *random*.
2) Ambil kartu pertama.
3) Lakukan pemeriksaan kombinasi dengan kartu lainnya sesuai dengan urutan derajad kombinasi 5 kartu poker dari yang terbesar ke derajad kombinasi 5 kartu poker yang terkecil dengan kartu yang diambil sebagai kartu tertinggi pada kombinasi 5 kartu.
4) Jika ditemukan kombinasi yang lebih besar, maka tidak dilakukan pengecekkan kombinasi yang lebih kecil.
5) Jika tidak ditemukan kombinasi, maka kartu tersebut dianggap one pair atau tidak diikut sertakan dalam pencarian berikutnya.
6) Kartu yang sudah dikombinasikan tidak boleh diikutsertakan dalam kombinasi berikutnya.
7) Jika terdapat 5 atau lebih kartu yang belum dikombinasikan, maka ambil kartu berikutnya yang belum dikombinasikan kemudian ulangi langkah 3 hingga semua kartu dikombinasikan.

Algoritma Greedy untuk mencari kombinasi 5 kartu pada permainan poker adalah, sebagai berikut:

1) Ambil sejumlah *n* kartu secara *random*.
2) Urutkan kartu berdasarkan nilai dan coraknya.
3) Ambil kartu pertama.
4) Lakukan pemeriksaan kombinasi dengan kartu lainnya sesuai dengan urutan derajad kombinasi 5 kartu poker dari yang terbesar ke derajad kombinasi 5 kartu poker yang terkecil dengan kartu yang diambil sebagai kartu tertinggi pada kombinasi 5 kartu.
5) Jika ditemukan kombinasi yang lebih besar, maka tidak dilakukan pengecekkan kombinasi yang lebih kecil.
6) Jika tidak ditemukan kombinasi, maka kartu tersebut dianggap one pair atau tidak diikut sertakan dalam pencarian berikutnya.
7) Kartu yang sudah dikombinasikan tidak boleh diikutsertakan dalam kombinasi berikutnya.
8) Jika terdapat 5 atau lebih kartu yang belum dikombinasikan, maka ambil kartu berikutnya yang belum dikombinasikan kemudian ulangi langkah 4 hingga semua kartu dikombinasikan.

4.2 *Pseudocode* Algoritma Brute-Force

Pseudocode algoritma Brute-Force untuk mencari kombinasi 5 kartu pada permainan poker adalah, sebagai berikut:

```
function pokerBruteForce(var n:Integer, var kombinasi:Array,
  var kartu:Array)

  var c:Integer;
  var i:Integer;
  var j:Integer;
  var k:Integer;
  var l:Integer;
  var m:Integer;
  var done:Boolean;
  var teks:String;
  var komposisi:Array;

  if n < 5 then
    break;
  end if

  c = 0
  for i = 0 to (n - 1) then
    done = false;
    if ((n - c) > 4) and (kombinasi[i] = false) then

      // cari straight flush

      for j = (i + 1) to (n - 1) do
        if (kombinasi[j] = false) and (((n - c) - 1) > 3) then
```

```
          if (kartu[j].bobotCorak = kartu[i].bobotCorak) and
          (kartu[j].nilai = (kartu[i].nilai - 1)) then
            for k = (i + 1) to (n - 1) do
              if (kombinasi[k] = false) and (((n - c) - 2) > 2) then
                if (kartu[k].bobotCorak = kartu[j].bobotCorak) and
                (kartu[k].nilai = (kartu[j].nilai - 1)) and
                (kartu[k].nama != kartu[i].nama) and
                (kartu[k].nama != kartu[j].nama) then
                  for l = (i + 1) to (n - 1) do
                    if (kombinasi[l] = false) and
                    (((n - c) - 3) > 1) then
                      if (kartu[l].bobotCorak = kartu[k].bobotCorak)
                      and (kartu[l].nilai = (kartu[k].nilai - 1))
                      and (kartu[l].nama != kartu[i].nama) and
                      (kartu[l].nama != kartu[j].nama) and
                      (kartu[l].nama != kartu[k].nama) then
                        for m = (i + 1) to (n - 1) do
                          if (kombinasi[m] = false) and
                          ((n - c) > 0) then
                            if (kartu[m].bobotCorak =
                            kartu[l].bobotCorak) and
                            (kartu[m].nilai =
                            (kartu[l].nilai - 1)) and
                            (kartu[m].nama != kartu[i].nama) and
                            (kartu[m].nama != kartu[j].nama) and
                            (kartu[m].nama != kartu[k].nama) and
                            (kartu[m].nama != kartu[l].nama) then
                              c = c + 5;
                              kombinasi[i] = true;
                              kombinasi[j] = true;
                              kombinasi[k] = true;
                              kombinasi[l] = true;
                              kombinasi[m] = true;
                              done = true;
                              teks = " Straight flush " + kartu[i] +
                                     ", " + kartu[j] + ", " +
                                     kartu[k] + ", " + kartu[l] +
                                     ", " + kartu[m];
                              komposisi.push(teks);
                              break;
                            end if
                          end if
                          if done = true then
                            break;
                          end if
                        end for
                      end if
                    end if
                    if done = true then
                      break;
                    end if
                  end for
                end if
              end if
              if done = true then
                break;
              end if
            end for
          end if
        end if
        if done = true then
          break;
        end if
```

```
end for

// cari Full House

for j = (i + 1) to (n - 1) do
  if (kombinasi[j] = false) and (((n - c) - 1) > 3) then
    if kartu[j].nilai = kartu[i].nilai then
      for k = (i + 1) to (n - 1) do
        if (kombinasi[k] = false) and (((n - c) - 2) > 2) then
          if (kartu[k].nilai = kartu[j].nilai) and
          (kartu[k].nama != kartu[i].nama) and
          (kartu[k].nama != kartu[j].nama) then
            for l = (i + 1) to (n - 1) do
              if (kombinasi[l] = false) and
              (((n - c) - 3) > 1) then
                if (kartu[l].nilai != kartu[k].nilai)
                and (kartu[l].nama != kartu[i].nama) and
                (kartu[l].nama != kartu[j].nama) and
                (kartu[l].nama != kartu[k].nama) then
                  for m = (i + 1) to (n - 1) do
                    if (kombinasi[m] = false) and
                    ((n - c) > 0) then
                      if (kartu[m].nilai = kartu[l].nilai) and
                      (kartu[m].nama != kartu[i].nama) and
                      (kartu[m].nama != kartu[j].nama) and
                      (kartu[m].nama != kartu[k].nama) and
                      (kartu[m].nama != kartu[l].nama) then
                        c = c + 5;
                        kombinasi[i] = true;
                        kombinasi[j] = true;
                        kombinasi[k] = true;
                        kombinasi[l] = true;
                        kombinasi[m] = true;
                        done = true;
                        teks = "Full house " + kartu[i] +
                               ", " + kartu[j] + ", " +
                               kartu[k] + ", " + kartu[l] +
                               ", " + kartu[m];
                        komposisi.push(teks);
                        break;
                      end if
                    end if
                    if done = true then
                      break;
                    end if
                  end for
                end if
              end if
              if done = true then
                break;
              end if
            end for
          end if
        end if
        if done = true then
          break;
        end if
      end for
    end if
  end if
  if done = true then
    break;
  end if
```

```
      end for

      // cari flush

      if done = false then
       for j = (i + 1) to (n - 1) do
        if (kombinasi[j] = false) and (((n - c) - 1) > 3) then
          if (kartu[j].nilai = (kartu[i].nilai - 1)) then
            for k = (i + 1) to (n - 1) do
              if (kombinasi[k] = false) and (((n - c) - 2) > 2) then
                if (kartu[k].nilai = (kartu[j].nilai - 1)) and
                (kartu[k].bobotCorak = kartu[j].bobotCorak) and
                (kartu[k].nama != kartu[i].nama) and
                (kartu[k].nama != kartu[j].nama) then
                  for l = (i + 1) to (n - 1) do
                    if (kombinasi[l] = false) and
                    (((n - c) - 3) > 1) then
                      if (kartu[l].nilai = (kartu[k].nilai - 1)) and
                      (kartu[l].bobotCorak = kartu[k].bobotCorak)
                      and (kartu[l].nama != kartu[i].nama) and
                      (kartu[l].nama != kartu[j].nama) and
                      (kartu[l].nama != kartu[k].nama) then
                        for m = (i + 1) to (n - 1) do
                          if (kombinasi[m] = false) and
                          ((n - c) > 0) then
                            if (kartu[m].nilai =
                            (kartu[l].nilai - 1)) and
                            (kartu[m].bobotCorak =
                            kartu[l].bobotCorak) and
                            (kartu[m].nama != kartu[i].nama) and
                            (kartu[m].nama != kartu[j].nama) and
                            (kartu[m].nama != kartu[k].nama) and
                            (kartu[m].nama != kartu[l].nama) then
                              c = c + 5;
                              kombinasi[i] = true;
                              kombinasi[j] = true;
                              kombinasi[k] = true;
                              kombinasi[l] = true;
                              kombinasi[m] = true;
                              done = true;
                              teks = "Flush " + kartu[i] +
                                     ", " + kartu[j] + ", " +
                                     kartu[k] + ", " + kartu[l] +
                                     ", " + kartu[m];
                              komposisi.push(teks);
                              break;
                            end if
                          end if
                          if done = true then
                            break;
                          end if
                        end for
                      end if
                    end if
                    if done = true then
                      break;
                    end if
                  end for
                end if
              end if
              if done = true then
                break;
              end if
```

```
        end for
      end if
    end if
    if done = true then
      break;
    end if
   end for
  end if

  // cari straight

  if done = false then
   for j = (i + 1) to (n - 1) do
    if (kombinasi[j] = false) and (((n - c) - 1) > 3) then
      if (kartu[j].nilai = (kartu[i].nilai - 1)) then
        for k = (i + 1) to (n - 1) do
          if (kombinasi[k] = false) and (((n - c) - 2) > 2) then
            if (kartu[k].nilai = (kartu[j].nilai - 1)) and
            (kartu[k].nama != kartu[i].nama) and
            (kartu[k].nama != kartu[j].nama) then
              for l = (i + 1) to (n - 1) do
                if (kombinasi[l] = false) and
                (((n - c) - 3) > 1) then
                  if (kartu[l].nilai = (kartu[k].nilai - 1)) and
                  (kartu[l].nama != kartu[i].nama) and
                  (kartu[l].nama != kartu[j].nama) and
                  (kartu[l].nama != kartu[k].nama) then
                    for m = (i + 1) to (n - 1) do
                      if (kombinasi[m] = false) and
                      ((n - c) > 0) then
                        if (kartu[m].nilai =
                        (kartu[l].nilai - 1)) and
                        (kartu[m].nama != kartu[i].nama) and
                        (kartu[m].nama != kartu[j].nama) and
                        (kartu[m].nama != kartu[k].nama) and
                        (kartu[m].nama != kartu[l].nama) then
                          c = c + 5;
                          kombinasi[i] = true;
                          kombinasi[j] = true;
                          kombinasi[k] = true;
                          kombinasi[l] = true;
                          kombinasi[m] = true;
                          done = true;
                          teks = "Straight " + kartu[i] +
                                 ", " + kartu[j] + ", " +
                                 kartu[k] + ", " + kartu[l] +
                                 ", " + kartu[m];
                          komposisi.push(teks);
                          break;
                        end if
                      end if
                      if done = true then
                        break;
                      end if
                    end for
                  end if
                end if
                if done = true then
                  break;
                end if
              end for
            end if
          end if
```

```
                if done = true then
                  break;
                end if
              end for
            end if
          end if
          if done = true then
            break;
          end if
        end for
       end if

    end if
  end for
end function
```

4.3 *Pseudocode* Algoritma Greedy

Sebelum menjalankan algoritma Greedy dalam mencari kombinasi 5 kartu pada permainan poker, kartu yang diberikan secara *random* terlebih dahulu melalui proses pengurutan. Proses pengurutan yang dapat digunakan adalah Selection Sort. Selection Sort merupakan teknik pengurutan array yang tergolong algoritma Greedy. Kompleksitas algoritma selection sort adalah $O(n^2)$. *Pseudocode* algoritma Selection Sort adalah, sebagai berikut:

```
procedure selectionSort(var kartu:Array)

      var iPos:Integer;
      var iMax:Integer;

      for iPos = 0 to (n - 1) do
            iMax = iPos;
            for i = iPos+1 to n do
                  if (kartu[i].nilai > kartu[iMax].nilai) then
                        iMax = i;
                  else if (kartu[i].nilai = kartu[iMax].nilai) then
                        if (kartu[i].bobotCorak =
                        kartu[iMax].bobotCorak) then
                              iMax = i;
                        end if
                  end if
            end for
            if iMax != iPos then
                  swap(kartu[iPos], kartu[iMax]);
            end if
      end for
end procedure
```

Dalam pengurutan array di atas, yang diurutkan adalah nilai kartu dan bobot coraknya. Bobot dari kartu ditentukan berdasarkan nilai dan coraknya. Kartu 3 wajik memiliki nilai 1 dan bobot corak 1, dan kartu 2 sekop memiliki nilai 13 dan bobot corak 4. Setelah array kartu diurutkan berdasarkan nilai dan bobot coraknya, selanjutnya adalah melakukan pencarian kombinasi kartu dengan menggunakan algoritma Greedy.

Pseudocode algoritma Greedy untuk mencari kombinasi 5 kartu pada permainan poker adalah, sebagai berikut:

```
function pokerGreedy(var n:Integer, var kombinasi:Array,
  var kartu:Array)

  var c:Integer;
  var i:Integer;
  var j:Integer;
  var k:Integer;
  var l:Integer;
  var m:Integer;
  var done:Boolean;
  var teks:String;
  var komposisi:Array;

  if n < 5 then
    break;
  end if

  c = 0
  for i = 0 to (n - 1) then
    done = false;
    if ((n - c) > 4) and (kombinasi[i] = false) then

      // cari straight flush

      for j = (i + 1) to (n - 1) do
        if (kombinasi[j] = false) and (((n - c) - 1) > 3) then
          if (kartu[j].bobotCorak = kartu[i].bobotCorak) and
          (kartu[j].nilai = (kartu[i].nilai - 1)) then
            for k = (i + 1) to (n - 1) do
              if (kombinasi[k] = false) and (((n - c) - 2) > 2) then
                if (kartu[k].bobotCorak = kartu[j].bobotCorak) and
                (kartu[k].nilai = (kartu[j].nilai - 1)) and
                (kartu[k].nama != kartu[i].nama) and
                (kartu[k].nama != kartu[j].nama) then
                  for l = (i + 1) to (n - 1) do
                    if (kombinasi[l] = false) and
                    (((n - c) - 3) > 1) then
                      if (kartu[l].bobotCorak = kartu[k].bobotCorak)
                      and (kartu[l].nilai = (kartu[k].nilai - 1))
                      and (kartu[l].nama != kartu[i].nama) and
                      (kartu[l].nama != kartu[j].nama) and
                      (kartu[l].nama != kartu[k].nama) then
                        for m = (i + 1) to (n - 1) do
                          if (kombinasi[m] = false) and
                          ((n - c) > 0) then
                            if (kartu[m].bobotCorak =
                            kartu[l].bobotCorak) and
                            (kartu[m].nilai =
                            (kartu[l].nilai - 1)) and
                            (kartu[m].nama != kartu[i].nama) and
                            (kartu[m].nama != kartu[j].nama) and
                            (kartu[m].nama != kartu[k].nama) and
                            (kartu[m].nama != kartu[l].nama) then
                              c = c + 5;
                              kombinasi[i] = true;
                              kombinasi[j] = true;
                              kombinasi[k] = true;
                              kombinasi[l] = true;
                              kombinasi[m] = true;
```

```
                            done = true;
                            teks = "Straight Flush " + kartu[i] +
                                   ", " + kartu[j] + ", " +
                                   kartu[k] + ", " + kartu[l] +
                                   ", " + kartu[m];
                            komposisi.push(teks);
                            break;
                          end if
                        end if
                        if done = true then
                          break;
                        end if
                      end for
                    end if
                  end if
                  if done = true then
                    break;
                  end if
                end for
              end if
            end if
            if done = true then
              break;
            end if
          end for
        end if
      end if
      if done = true then
        break;
      end if
    end for

    // cari Full House

    for j = (i + 1) to (n - 1) do
      if (kombinasi[j] = false) and (((n - c) - 1) > 3) then
        if kartu[j].nilai = kartu[i].nilai then
          for k = (i + 1) to (n - 1) do
            if (kombinasi[k] = false) and (((n - c) - 2) > 2) then
              if (kartu[k].nilai = kartu[j].nilai) and
              (kartu[k].nama != kartu[i].nama) and
              (kartu[k].nama != kartu[j].nama) then
                for l = (i + 1) to (n - 1) do
                  if (kombinasi[l] = false) and
                  (((n - c) - 3) > 1) then
                    if (kartu[l].nilai != kartu[k].nilai)
                    and (kartu[l].nama != kartu[i].nama) and
                    (kartu[l].nama != kartu[j].nama) and
                    (kartu[l].nama != kartu[k].nama) then
                      for m = (i + 1) to (n - 1) do
                        if (kombinasi[m] = false) and
                        ((n - c) > 0) then
                          if (kartu[m].nilai = kartu[l].nilai) and
                          (kartu[m].nama != kartu[i].nama) and
                          (kartu[m].nama != kartu[j].nama) and
                          (kartu[m].nama != kartu[k].nama) and
                          (kartu[m].nama != kartu[l].nama) then
                            c = c + 5;
                            kombinasi[i] = true;
                            kombinasi[j] = true;
                            kombinasi[k] = true;
                            kombinasi[l] = true;
                            kombinasi[m] = true;
```

```
                            done = true;
                            teks = "Full house " + kartu[i] +
                                   ", " + kartu[j] + ", " +
                                   kartu[k] + ", " + kartu[l] +
                                   ", " + kartu[m];
                            komposisi.push(teks);
                            break;
                          end if
                        end if
                        if done = true then
                          break;
                        end if
                      end for
                    end if
                  end if
                  if done = true then
                    break;
                  end if
                end for
              end if
            end if
            if done = true then
              break;
            end if
          end for
        end if
      end if
      if done = true then
        break;
      end if
    end for

    // cari flush

    if done = false then
     for j = (i + 1) to (n - 1) do
      if (kombinasi[j] = false) and (((n - c) - 1) > 3) then
        if (kartu[j].nilai = (kartu[i].nilai - 1)) then
          for k = (i + 1) to (n - 1) do
            if (kombinasi[k] = false) and (((n - c) - 2) > 2) then
              if (kartu[k].nilai = (kartu[j].nilai - 1)) and
              (kartu[k].bobotCorak = kartu[j].bobotCorak) and
              (kartu[k].nama != kartu[i].nama) and
              (kartu[k].nama != kartu[j].nama) then
                for l = (i + 1) to (n - 1) do
                  if (kombinasi[l] = false) and
                  (((n - c) - 3) > 1) then
                    if (kartu[l].nilai = (kartu[k].nilai - 1)) and
                    (kartu[l].bobotCorak = kartu[k].bobotCorak)
                    and (kartu[l].nama != kartu[i].nama) and
                    (kartu[l].nama != kartu[j].nama) and
                    (kartu[l].nama != kartu[k].nama) then
                      for m = (i + 1) to (n - 1) do
                        if (kombinasi[m] = false) and
                        ((n - c) > 0) then
                          if (kartu[m].nilai =
                          (kartu[l].nilai - 1)) and
                          (kartu[m].bobotCorak =
                          kartu[l].bobotCorak) and
                          (kartu[m].nama != kartu[i].nama) and
                          (kartu[m].nama != kartu[j].nama) and
                          (kartu[m].nama != kartu[k].nama) and
                          (kartu[m].nama != kartu[l].nama) then
```

```
                            c = c + 5;
                            kombinasi[i] = true;
                            kombinasi[j] = true;
                            kombinasi[k] = true;
                            kombinasi[l] = true;
                            kombinasi[m] = true;
                            done = true;
                            teks = "Flush " + kartu[i] +
                                   ", " + kartu[j] + ", " +
                                   kartu[k] + ", " + kartu[l] +
                                   ", " + kartu[m];
                            komposisi.push(teks);
                            break;
                          end if
                        end if
                        if done = true then
                          break;
                        end if
                      end for
                    end if
                  end if
                  if done = true then
                    break;
                  end if
                end for
              end if
            end if
            if done = true then
              break;
            end if
          end for
        end if
      end if
      if done = true then
        break;
      end if
    end for
  end if

  // cari straight

  if done = false then
   for j = (i + 1) to (n - 1) do
    if (kombinasi[j] = false) and (((n - c) - 1) > 3) then
      if (kartu[j].nilai = (kartu[i].nilai - 1)) then
        for k = (i + 1) to (n - 1) do
          if (kombinasi[k] = false) and (((n - c) - 2) > 2) then
            if (kartu[k].nilai = (kartu[j].nilai - 1)) and
            (kartu[k].nama != kartu[i].nama) and
            (kartu[k].nama != kartu[j].nama) then
              for l = (i + 1) to (n - 1) do
                if (kombinasi[l] = false) and
                (((n - c) - 3) > 1) then
                  if (kartu[l].nilai = (kartu[k].nilai - 1)) and
                  (kartu[l].nama != kartu[i].nama) and
                  (kartu[l].nama != kartu[j].nama) and
                  (kartu[l].nama != kartu[k].nama) then
                    for m = (i + 1) to (n - 1) do
                      if (kombinasi[m] = false) and
                      ((n - c) > 0) then
                        if (kartu[m].nilai =
                        (kartu[l].nilai - 1)) and
                        (kartu[m].nama != kartu[i].nama) and
```

```
                                (kartu[m].nama != kartu[j].nama) and
                                (kartu[m].nama != kartu[k].nama) and
                                (kartu[m].nama != kartu[l].nama) then
                                  c = c + 5;
                                  kombinasi[i] = true;
                                  kombinasi[j] = true;
                                  kombinasi[k] = true;
                                  kombinasi[l] = true;
                                  kombinasi[m] = true;
                                  done = true;
                                  teks = "Straight " + kartu[i] +
                                         ", " + kartu[j] + ", " +
                                         kartu[k] + ", " + kartu[l] +
                                         ", " + kartu[m];
                                  komposisi.push(teks);
                                  break;
                                end if
                              end if
                              if done = true then
                                break;
                              end if
                            end for
                          end if
                        end if
                        if done = true then
                          break;
                        end if
                      end for
                    end if
                  end if
                  if done = true then
                    break;
                  end if
                end for
              end if
            end if
            if done = true then
              break;
            end if
          end for
        end if

    end if
  end for
end function
```

4.4 Langkah Pencarian Menggunakan Algoritma Brute-Force

Contoh persoalan pencarian kombinasi 5 kartu dari permainan poker yang akan diselesaikan adalah, sebagai berikut:

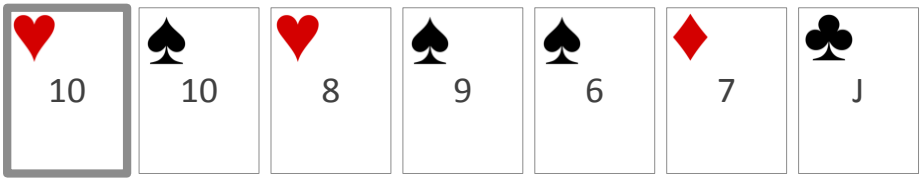
Diberikan sebanyak 7 kartu remi dengan urutan, sebagai berikut:

Gambar 4. 1 Contoh komposisi kartu pada permainan poker

Buatlah kombinasi 5 kartu dari kartu-kartu di atas.

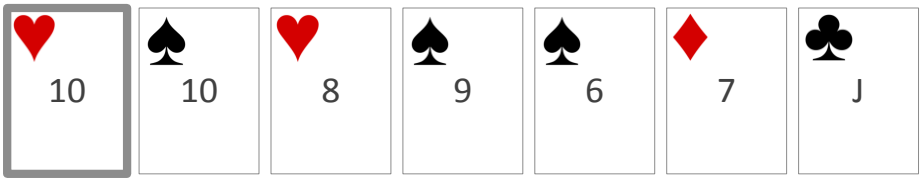
Pemecahan persoalan persoalan pencarian kombinasi 5 kartu dari permainan poker tersebut menggunakan algoritma Brute-Force adalah, sebagi berikut:

1) Ambil kartu pertama, yaitu 10 hati (*nilai = 10* dan *bobot corak = 3*).



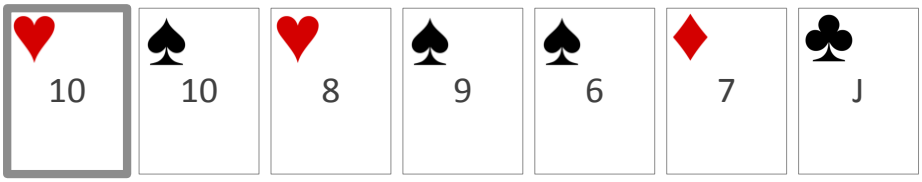
Gambar 4. 2 Perncarian kombinasi kartu menggunakan algoritma Brute-Force langkah 1

2) Karena tidak terdapat kartu dengan nilai 9 dan bobot corak 3 (kartu 9 hati) maka tidak ditemukan kombinasi *straight flush* untuk kartu 10 hati.



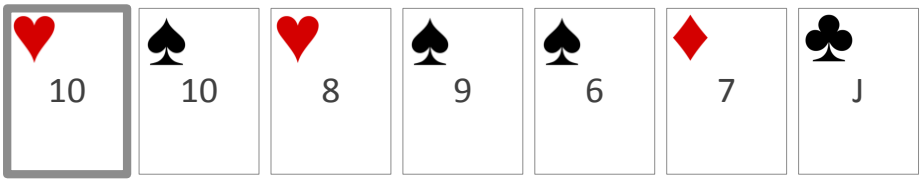
Gambar 4. 3 Perncarian kombinasi kartu menggunakan algoritma Brute-Force langkah 2

3) Karena tidak ditemukan 3 kartu dengan nilai 10 (kartu 10), maka tidak ditemukan kombinasi *full house*.



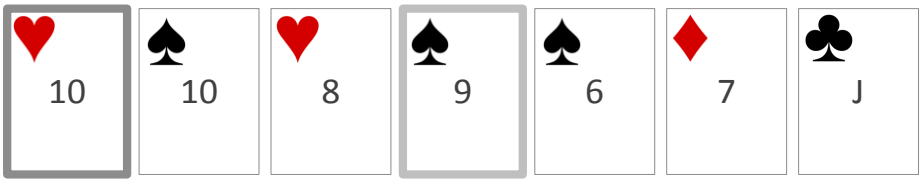
Gambar 4. 4 Perncarian kombinasi kartu menggunakan algoritma Brute-Force langkah 3

4) Karena tidak terdapat 5 kartu yang memiliki bobot corak 3 (kartu corak hati), maka tidak ditemukan kombinasi *flush*.



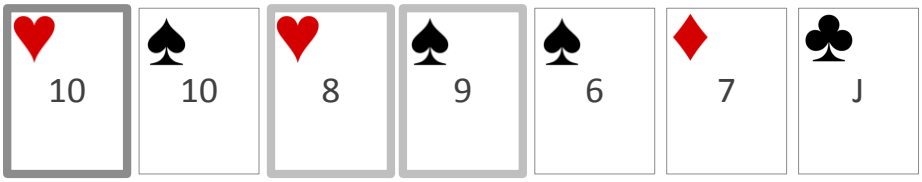
Gambar 4. 5 Perncarian kombinasi kartu menggunakan algoritma Brute-Force langkah 4

5) Ditemukan kartu dengan nilai 9 (kartu 9 sekop), kemudian lakukan pencarian kartu dengan nilai 8 (kartu 8).



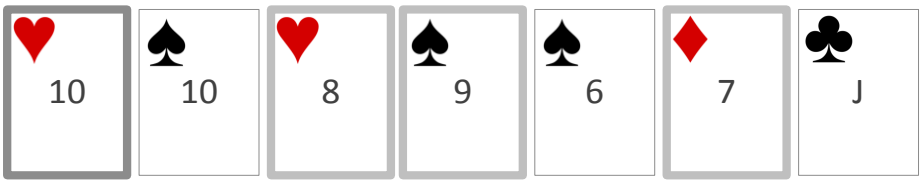
Gambar 4. 6 Perncarian kombinasi kartu menggunakan algoritma Brute-Force langkah 5

6) Ditemukan kartu dengan nilai 8 (kartu 8 hati), kemudian lakukan pencarian kartu dengan nilai 7 (kartu 7).



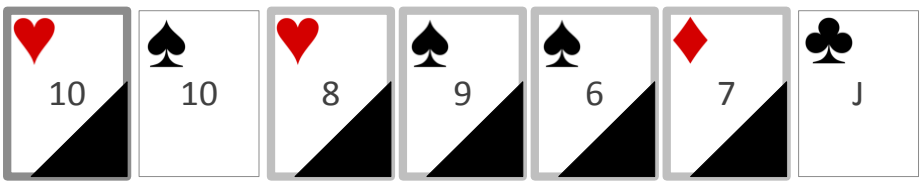
Gambar 4. 7 Perncarian kombinasi kartu menggunakan algoritma Brute-Force langkah 6

7) Ditemukan kartu dengan nilai 7 (kartu 7 wajik), kemudian lakukan pencarian kartu dengan nilai 6 (kartu 6).



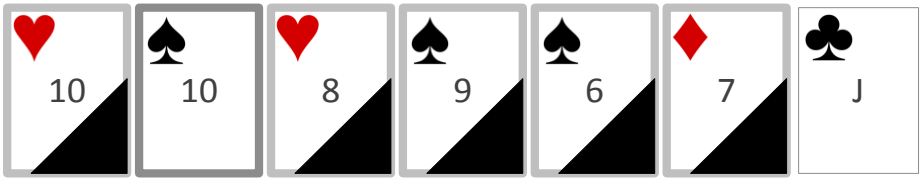
Gambar 4. 8 Perncarian kombinasi kartu menggunakan algoritma Brute-Force langkah 7

8) Ditemukan kartu dengan nilai 6 (kartu 6 sekop) sehingga membentuk kombinasi straight. Beri tanda pada kelima kartu tersebut.



Gambar 4. 9 Perncarian kombinasi kartu menggunakan algoritma Brute-Force langkah 8

9) Ambil kartu berikutnya yang belum diberi tanda, yaitu kartu 10 sekop (*nilai = 10* dan *bobot corak = 4*).

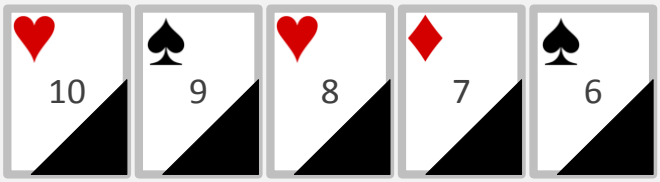


Gambar 4. 10 Perncarian kombinasi kartu menggunakan algoritma Brute-Force langkah 9

10) Karena sisa kartu sudah kurang dari 5, maka pencarian tidak dilanjutkan.

Kombinasi 5 kartu yang dihasilkan menggunakan algoritma Brute-Force adalah, sebagai berikut:

Kombinasi *Straight* dengan kartu tertinggi 10 hati.



Gambar 4. 11 Hasil akhir komposisi kartu menggunakan algoritma Brute-Force

4.5 Langkah Pencarian Menggunakan Algoritma Greedy

Dengan menggunakan contoh persoalan pencarian kombinasi 5 kartu dari permainan poker yang sama, maka pemecahan persoalan persoalan pencarian kombinasi 5 kartu dari permainan poker tersebut menggunakan algoritma Greedy adalah, sebagi berikut:

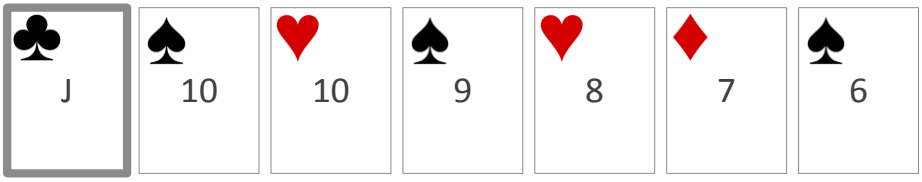
1) Urutkan kartu berdasarkan nilai dan bobotnya.

Gambar 4. 12 Perncarian kombinasi kartu menggunakan algoritma Greedy langkah 1

2) Ambil kartu pertama, yaitu *Jack* keriting (*nilai = 11* dan *bobot corak = 2*).

Gambar 4. 13 Perncarian kombinasi kartu menggunakan algoritma Greedy langkah 2

3) Karena tidak terdapat kartu dengan nilai 10 dan bobot corak 2 (kartu 10 keriting) maka tidak ditemukan kombinasi *straight flush* untuk kartu 10 hati.

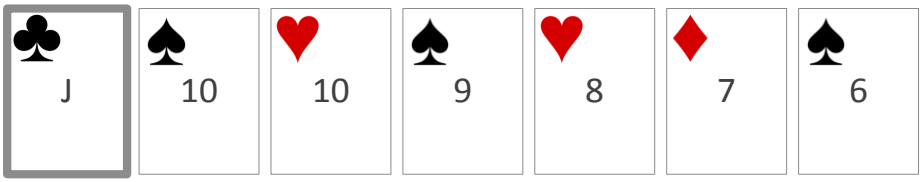


Gambar 4. 14 Perncarian kombinasi kartu menggunakan algoritma Greedy langkah 3

4) Karena tidak ditemukan 3 kartu dengan nilai 11 (kartu *Jack*), maka tidak ditemukan kombinasi *full house*.

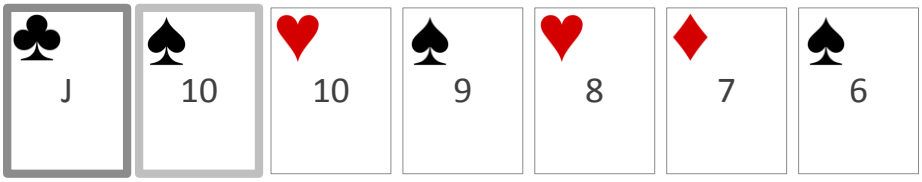
Gambar 4. 15 Perncarian kombinasi kartu menggunakan algoritma Greedy langkah 4

5) Karena tidak terdapat 5 kartu yang memiliki bobot corak 2 (kartu corak keriting), maka tidak ditemukan kombinasi *flush*.



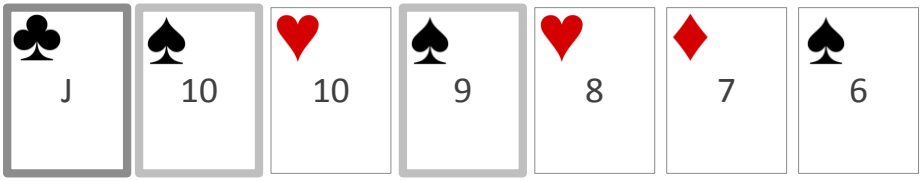
Gambar 4. 16 Perncarian kombinasi kartu menggunakan algoritma Greedy langkah 5

6) Ditemukan kartu dengan nilai 10 (kartu 10 sekop), kemudian lakukan pencarian kartu dengan nilai 9 (kartu 9).



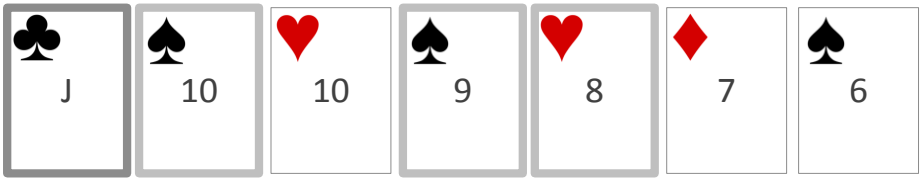
Gambar 4. 17 Perncarian kombinasi kartu menggunakan algoritma Greedy langkah 6

7) Ditemukan kartu dengan nilai 9 (kartu 9 sekop), kemudian lakukan pencarian kartu dengan nilai 8 (kartu 8).



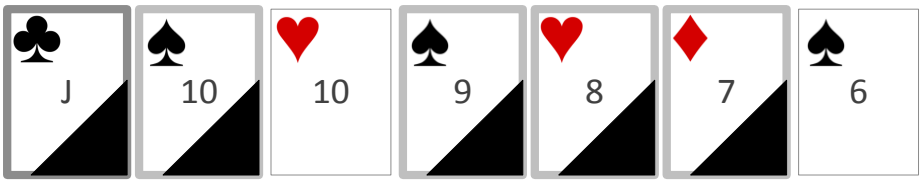
Gambar 4. 18 Perncarian kombinasi kartu menggunakan algoritma Greedy langkah 7

8) Ditemukan kartu dengan nilai 8 (kartu 8 hati), kemudian lakukan pencarian kartu dengan nilai 7 (kartu 7).



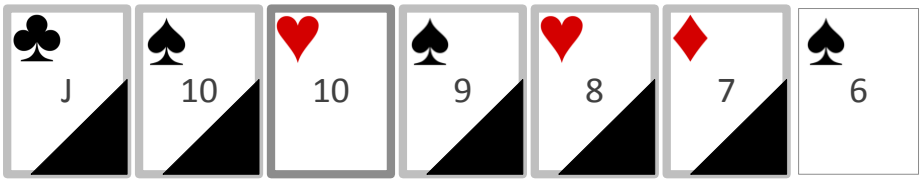
Gambar 4. 19 Perncarian kombinasi kartu menggunakan algoritma Greedy langkah 8

9) Ditemukan kartu dengan nilai 7 (kartu 7 wajik) sehingga membentuk kombinasi straight. Beri tanda pada kelima kartu tersebut.



Gambar 4. 20 Perncarian kombinasi kartu menggunakan algoritma Greedy langkah 9

10) Ambil kartu berikutnya yang belum diberi tanda, yaitu kartu 10 hati (*nilai = 10* dan *bobot corak = 3*).

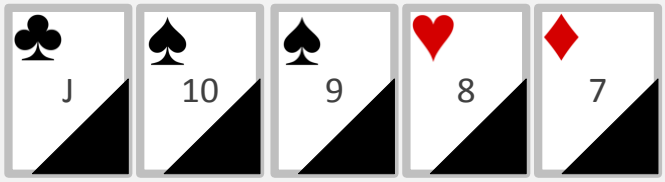


Gambar 4. 21 Perncarian kombinasi kartu menggunakan algoritma Greedy langkah 10

11) Karena sisa kartu sudah kurang dari 5, maka pencarian tidak dilanjutkan.

Kombinasi 5 kartu yang dihasilkan menggunakan algoritma Greedy adalah, sebagai berikut:

Kombinasi *Straight* dengan kartu tertinggi *Jack* keriting.



Gambar 4. 22 Hasil kombinasi 5 kartu menggunakan algoritma greedy

4.6 Kompleksitas Algoritma Brute-Force

Pada dasarnya, kompleksitas waktu suatu algoritma dapat dicari dengan menghitung jumlah pengulangan dari masing-masing operasi. Namun, untuk alasan penyederhanaan, pengukuran kompleksitas algoritma dapat dilakukan dengan menghitung pengulangan operasi-operasi dasar dari algoritma. Masing-masing baris kode yang berada di dalam lingkup operasi dasar dinyatakan dengan *1* dan tidak dilibatkan dalam perhitungan kompleksitas waktu. Dari pengulangan operasi-operasi dasar pada algoritma Brute-Force, didapat kompleksitas waktu, sebagai berikut:

$$T(n) = \sum t_i$$

$$T(n) = 1 + 4(2n^5) + 4(3n^4) + 4(3n^3) + 4(3n^2) + 4(2n^2) + 4(\mathrm{n}) + \mathrm{n}$$

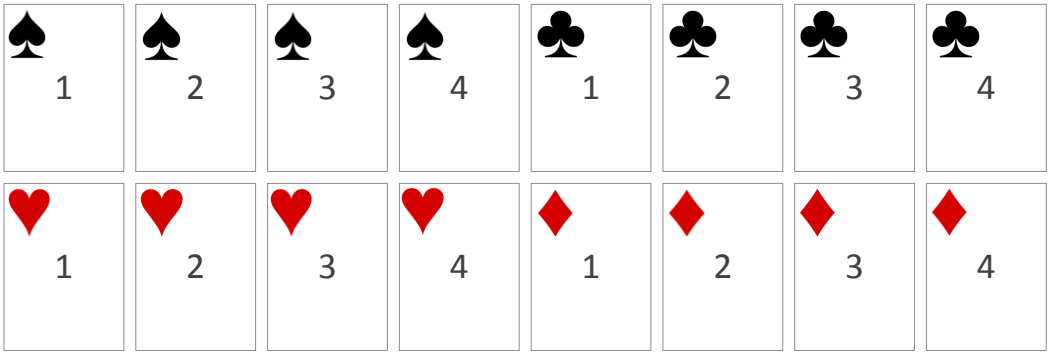
$$T(n) = 8n^5 + 12n^4 + 12n^3 + 12n^2 + 5n + 1$$

Kasus terbaik untuk algoritma Brute-Force untuk mencari kombinasi 5 kartu dalam permainan poker adalah jika *n = 5* dan kartu telah terurut berdasarkan nilai dan bobotnya. Kompleksitas waktu dengan menghitung pengulangan operasi-operasi dasar pada algoritma Brute-Force untuk kasus terbaik ini ($T_{min}(n)$) adalah, sebagai berikut:

$$T(n) = \sum t_i$$

$$T(n) = 2n^5 + 3n^4 + 3n^3 + 3n^2 + 2n^2 + 2\mathrm{n} + 1$$

Sedangkan kasus terburuk untuk algoritma Brute-Force dalam mencari kombinasi 5 kartu dalam permainan poker adalah jika terdapat masing 4 kartu dengan 4 corak yang berbeda (16 kartu), dimana diantara kartu-kartu tersebut tidak ada yang nilainya saling berurutan sebanyak 5 kartu. Contoh kartu berikut ini:



Gambar 4. 23 Kasus terburuk mencari kombinasi 5 kartu pada permainan poker dengan algoritma Brute-Force

Kompleksitas waktu suatu algoritma dapat juga ditentukan dengan membandingkan kompleksitas waktu algoritma tersebut dengan kompleksitas waktu yang presisi untuk suatu algoritma. Kompleksitas seperti ini disebut juga dengan kompleksitas waktu asimptotik yang dinyatakan dengan notasi O-besar (*O*). Untuk algoritma Brute-Force pada kasus mencari kombinasi 5 kartu dalam permainan poker, kompleksitas waktu asimptotiknya adalah, sebagai berikut:

$T(n) = \sum t_i$

$T(n) = 1 + 4(2n^5) + 4(3n^4) + 4(3n^3) + 4(3n^2) + 4(2n^2) + 4(\mathrm{n}) + \mathrm{n}$

$T(n) = 8n^5 + 12n^4 + 12n^3 + 12n^2 + 5n + 1$

Berdasarkan teorema: $T(n) \leq C.f(n)$,

sehingga,

$8n^5 + 12n^4 + 12n^3 + 12n^2 + 5n + 1 \leq 8n^5 + 12n^5 + 12n^5 + 12n^5 + 5n^5 + n^5$,

maka: $8n^5 + 12n^4 + 12n^3 + 12n^2 + 5n + 1 = O(n^5)$

Jadi, kompleksitas waktu asimptotiknya adalah $O(n^5)$

4.7 Kompleksitas Algoritma Greedy

Untuk alasan penyederhanaan, pengukuran kompleksitas algoritma dapat dilakukan dengan menghitung pengulangan operasi-operasi dasar dari algoritma. Masing-masing baris kode yang berada di dalam lingkup operasi dasar dinyatakan dengan *1* dan tidak dilibatkan dalam perhitungan kompleksitas waktu. Dari pengulangan operasi-operasi dasar pada algoritma Greedy, didapat kompleksitas waktu, sebagai berikut:

$T(n) = \sum t_i$

$T(n) = 1 + 4(2n^5) + 4(3n^4) + 4(3n^3) + 4(3n^2) + 4(2n^2) + 4(\mathrm{n}) + \mathrm{n}$

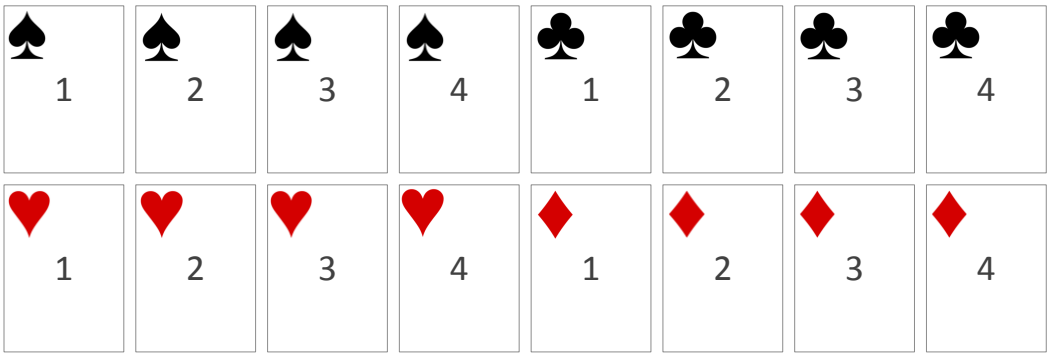
$T(n) = 8n^5 + 12n^4 + 12n^3 + 12n^2 + 5n + 1$

Kasus terbaik untuk algoritma Greedy untuk mencari kombinasi 5 kartu dalam permainan poker adalah jika *n = 5* dan kartu telah terurut berdasarkan nilai dan bobotnya. Kompleksitas waktu dengan menghitung pengulangan operasi-operasi dasar pada algoritma Greedy untuk kasus terbaik ini ($T_{min}(n)$) adalah, sebagai berikut:

$T(n) = \sum t_i$

$T(n) = 2n^5 + 3n^4 + 3n^3 + 3n^2 + 2n^2 + 2\mathrm{n} + 1$

Sedangkan kasus terburuk untuk algoritma Greedy dalam mencari kombinasi 5 kartu dalam permainan poker adalah jika terdapat masing 4 kartu dengan 4 corak yang berbeda (16 kartu), dimana diantara kartu-kartu tersebut tidak ada yang nilainya saling berurutan sebanyak 5 kartu. Contoh kartu berikut ini:



Gambar 4. 24 Kasus terburuk mencari kombinasi 5 kartu pada permainan poker dengan algoritma Greedy

Kompleksitas waktu suatu algoritma dapat juga ditentukan dengan membandingkan kompleksitas waktu algoritma tersebut dengan kompleksitas waktu yang presisi untuk suatu algoritma. Kompleksitas seperti ini disebut juga dengan kompleksitas waktu asimptotik yang dinyatakan dengan notasi O-besar (*O*). Untuk algoritma Greedy pada kasus mencari kombinasi 5 kartu dalam permainan poker, kompleksitas waktu asimptotiknya adalah, sebagai berikut:

$T(n) = \sum t_i$

$T(n) = 1 + 4(2n^5) + 4(3n^4) + 4(3n^3) + 4(3n^2) + 4(2n^2) + 4(\mathrm{n}) + \mathrm{n}$

$T(n) = 8n^5 + 12n^4 + 12n^3 + 12n^2 + 5n + 1$

Berdasarkan teorema: $T(n) \leq C.f(n)$,

sehingga,

$8n^5 + 12n^4 + 12n^3 + 12n^2 + 5n + 1 \leq 8n^5 + 12n^5 + 12n^5 + 12n^5 + 5n^5 + n^5$,

maka: $8n^5 + 12n^4 + 12n^3 + 12n^2 + 5n + 1 = O(n^5)$

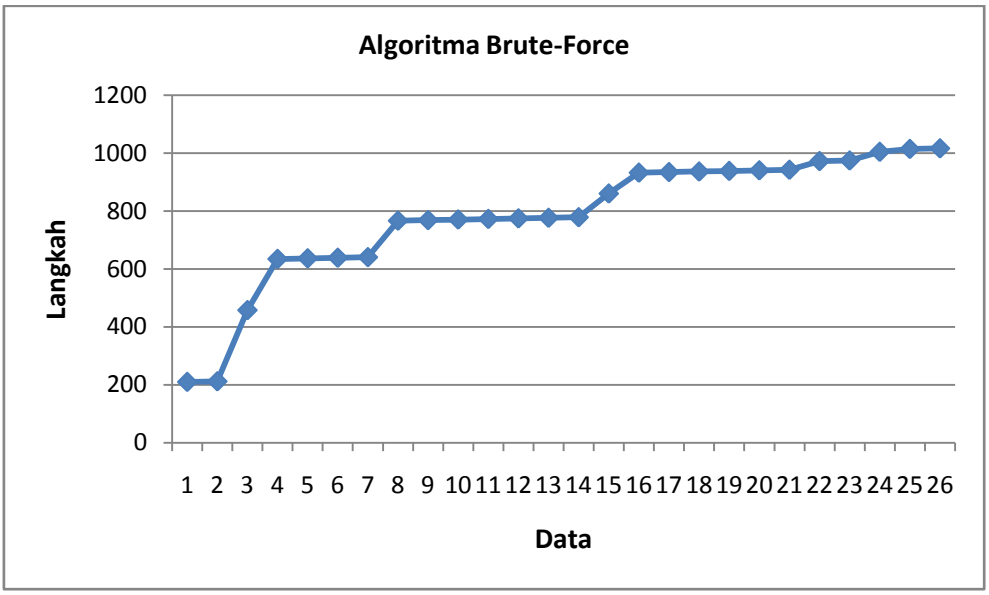
Jadi, kompleksitas waktu asimptotiknya adalah $O(n^5)$

Karena algoritma Greedy ini diawali dengan melakukan pengurutan pada kartu, maka kompleksitas waktu algoritma Greedy ini dibandingkan dengan kompleksitas waktu asimptotik algoritma sorting yang digunakan yaitu selection sort dengan kompleksitas $O(n^2)$.

$max(O(n^5), O(n^2)) = O(n^5)$

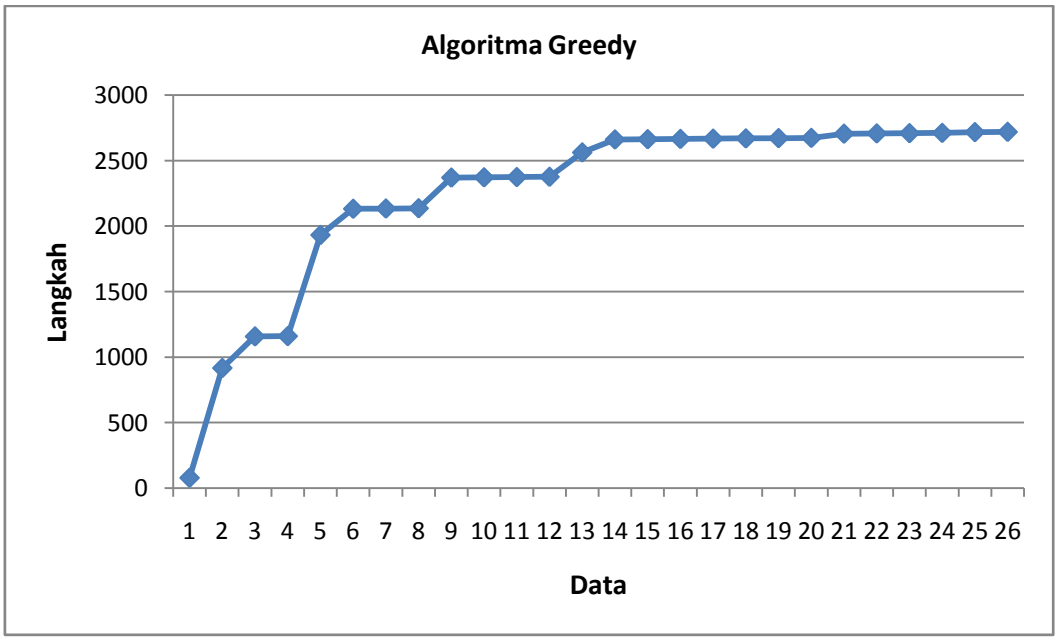
4.8 Grafik Percobaan

Grafik hasil percobaan pencarian kombinasi 5 kartu pada permaian poker menggunakan algoritma Brute-Force adalah, sebagai berikut:



Gambar 4. 25 Grafik hasil percobaan mencari kombinasi 5 kartu pada permainan poker dengan algoritma Brute-Force

Grafik hasil percobaan pencarian kombinasi 5 kartu pada permaian poker menggunakan algoritma Greedy adalah, sebagai berikut:



Gambar 4. 26 Grafik hasil percobaan mencari kombinasi 5 kartu pada permainan poker dengan algoritma Greedy

5. Kesimpulan

Dari hasil akhir yang diperoleh dari eksperimen menggunakan algoritma Brute-Force dan algoritma Greedy ini dapat disimpulkan beberapa poin, sebagai berikut:

1) Persoalan transportasi seimbang.
   - Kompleksitas algoritma Brute-Force = $T(n) = \frac{13n}{2} + 15 = O(n)$.
   - Kompleksitas algoritma Greedy = $T(n) = 11n = O(n)$.
   - Solusi minimum yang dihasilkan menggunakan algoritma Brute-Force= *355*.
   - Solusi minimum yang dihasilkan menggunakan algoritma Greedy = *235*.
   - Pada kasus persoalan transportasi seimbang ini, baik algoritma Greedy maupun algoritma Brute-Force memiliki kompleksitas yang sama yaitu kompleksitas linier *O(n)*. Pada kasus tertentu dimana jumlah supply dan demmand pada masing-masing baris dan kolom sama serta cost terkecil terdapat di bidang diagonal dari kiri atas ke kanan bawah matriks, maka algoritma Brute-Force akan lebih efisien dibandingkan menggunakan algoritma Greedy. Hal ini dikarenakan solusi optimum yang dihasilkan masing-masing algoritma adalah sama. Namun untuk kasus nyata, algoritma Greedy lebih baik digunakan karena dapat menghasilkan solusi yang lebih baik dibandingkan algoritma Brute-Force.

2) Persoalan pewarnaan graf.
   - Kompleksitas algoritma Brute-Force = $2n^3 + n^2 + n + 2 = O(n^3)$.
   - Kompleksitas algoritma Greedy = $2n^3 + n^2 + n + 2 = O(n^3)$.
   - Jumlah warna yang dihasilkan menggunakan algoritma Brute-Force= *3*.
   - Jumlah warna yang dihasilkan menggunakan algoritma Greedy = *3*.
   - Pada kasus pewarnaan graf ini, baik algoritma Greedy maupun algoritma Brute-Force memiliki kompleksitas yang sama yaitu *O(n^3)*. Pada graf yang sudah terurut jumlah cabangnya, algoritma Brute-Force akan memiliki jumlah langkah yang sama dengan algoritma Greedy. Pada dasarnya optimasi jumlah warna pada persoalan pewarnaan graf ini masih menjadi persoalan. Pada kasus ini, algoritma Greedy akan memiliki waktu proses yang lebih cepat dari pada algoritma Greedy mengingat simpul-simpul berikutnya memiliki cabang yang lebih sedikit yang akhirnya proses pencarian warna tetangga pun akan lebih sedikit.

3) Pencarian kombinasi 5 kartu pada permainan poker.
   - Kompleksitas algoritma Brute-Force = $8n^5 + 12n^4 + 12n^3 + 12n^2 + 5n + 1 = O(n^5)$.
   - Kompleksitas algoritma Greedy = $8n^5 + 12n^4 + 12n^3 + 12n^2 + 5n + 1 = O(n^5)$.
   - Jumlah warna yang dihasilkan menggunakan algoritma Brute-Force adalah kombinasi *straight* dengan kartu tertinggi 10 hati..
   - Komposisi 5 kartu yang dihasilkan menggunakan algoritma Greedy adalah kombinasi *straight* dengan kartu tertinggi *Jack* keriting.
   - Pada kasus pencarian kombinasi 5 kartu pada permainan poker ini, baik algoritma Greedy maupun algoritma Brute-Force memiliki kompleksitas yang sama yaitu *O(n^5)*.

Meskipun algoritma Greedy memiliki peluang untuk mendapatkan kombinasi dengan nilai terbesar, namun untuk pencarian selanjutnya algoritma Greedy akan membutuhkan waktu yang lebih lama. Selain itu nilai kombinasi 5 kartu yang dihasilkan berikutnya cenderung lebih kecil. Sedangkan pada algoritma Brute-Force, agar mendapat kombinasi yang lebih tinggi sangat bergantung pada faktor keberungan. Algoritma ini sangat berpeluang menghasilkan kartu tanpa kombinasi atau *single card*.

6. Referensi

[1] Munir, Rinaldi., *Matematika Diskrit.* 3. Bandung : Informatika, 2005. 979-96446-3-1.

[2] Dimyati, Tarliah Tjutju and Dimyati, Ahmad., *Operations Research: Model-model pengambilan keputusan.* 8. bandung : Sinar Baru Algensindo, 2006.