ALGORITMA DIVIDE AND CONQUER BAGIAN 1

Disusun guna memenuhi tugas mata kuliah

Analisis Algoritma I

Dosen Pengampu

Dian Rusvinasari, S.Kom., M,Kom.

Oleh:

Kelompok 1

Rainaldi Putra Setiawan	210202582
Amos Cristo Aginta	210202556
Farhan Hidayat	210202613
Radityo Priyambudi	210202581

KELAS IKRA

PROGRAM STUDI ILMU KOMPUTER

FAKULTAS SAINS DAN TEKNOLOGI

UNIVERSITAS PUTRA BANGSA

TAHUN 2022

KATA PENGANTAR

Puji syukur kami panjatkan kepada Tuhan Yang Maha Esa, karena atas limpahan rahmatnya penyusun dapat menyelesaikan makalah ini tepat waktu tanpa ada halangan yang berarti dan sesuai dengan harapan.

Ucapan terima kasih kami sampaikan kepada ibu Dian Rusvinasari S.kom M.kom sebagai dosen pengampu mata kuliah Analisis Algoritma I yang telah membantu memberikan arahan dan pemahaman dalam penyusunan makalah ini.

Kami menyadari bahwa dalam penyusunan makalah ini masih banyak kekurangan karena keterbatasan kami. Maka dari itu penyusun sangat mengharapkan kritik dan saran untuk menyempurnakan makalah ini. Semoga apa yang ditulis dapat bermanfaat bagi semua pihak yang membutuhkan.

Kebumen, 26 Desember 2022

Kelompok 1

DAFTAR ISI

DIVIDE AND CONQUER .....

1. Pengertian Algoritma Divide And Conquer .....

1. 1 Skema Umum Algoritma Divide And Conquer .....

1. 2 Penerapan Alogritma Divide and Conquer .....

2. Contoh soal dengan menggunakan Algoritma Divide and Conquer .....

2.1 Contoh soal mencari nilai minimum dan maksimum (MinMax).....

2.2 Perbandingan Algoritma MinMax.....

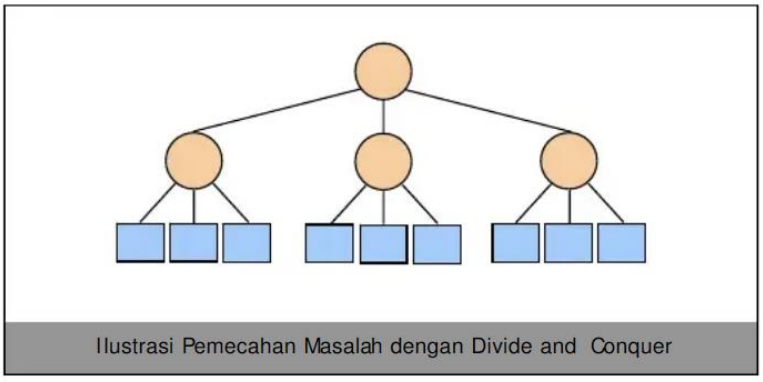
2.3 Persoalan Perpangkatan a^n.....

DIVIDE AND CONQUER

1. Pengertian Algoritma Divide And Conquer

Algoritma divide and conquer sudah lama diperkenalkan sebagai sumber dari pengendalian proses paralel, karena masalah-masalah yang terjadi dapat diatasi secara independen. Banyak arsitektur dan bahasa pemrograman paralel mendesain implementasinya (aplikasi) dengan struktur dasar dari algoritma divide and conquer. Untuk menyelesaikan masalah-masalah yang besar, dan dibagi (dipecah) menjadi bagian yang lebih kecil dan menggunakan sebuah solusi untuk menyelesaikan problem awal adalah prinsip dasar dari pemrograman/strategi divide and conquer, Divide and Conquer dulunya adalah strategi militer yang dikenal dengan nama divide ut imperes. Sekarang strategi tersebut menjadi strategi fundamental di dalam ilmu komputer dengan nama Divide and Conquer. Berikut pengertian dari algoritma Divide Anda Conquer

1. Devide adalah membagi masalah menjadi beberapa sub-masalah yang memiliki kemiripan dengan masalah semula namun berukuran lebih kecil (idealnya berukuran hampir sama),
2. Conquer: memecahkan (menyelesaikan) masing-masing sub-masalah (secara rekursif), dan
3. Combine: menggabungkan solusi masing-masing sub-masalah sehingga membentuk solusi masalah semula.



Gambar 1. Ilustrasi Pemecahan Maslah dengan Divide And Conquer

Combine: menggabungkan solusi masing-masing sub-masalah sehingga membenDivide and conquer adalah varian dari beberapa strategi pemrograman top-down, tetapi keistimewaannya adalah membuat sub-sub problem dari problem yangbesar, oleh karena itu strategi ini ditunjukkan secara berulang-ulang (recursively),didalam menerapkan algoritma yang sama dalam sub-sub problem seperti yangditerapkan pada masalah aslinya (original problem). Sebagaimana prinsip dasaralgoritma perulangan dibutuhkan sebuah kondisi untuk mengakhiri perulangantersebut. Biasanya untuk mengecek apakah problem sudah cukup kecil untukdiselesaikan dengan metode secara langsung. Mungkin dari segi ilustrasi kita, bahwaproses-proses pada komputer paralel tentunya memiliki proses/problem/job yangcukup kompleks sehingga harus dipecah-pecah menjadi sub-sub problem.Selain dibutuhkan sebuah “kondisi”, juga diperlukan “fase divide” untukmembagi/memecah problem menjadi sub-sub problem yang lebih kecil, dan “fasecombine“ untuk menggabun kembali solusi dari sub-sub problem kedalam solusi dari problem awalnya. Berikut pseudocode dari strategi divide and conquer :tuk solusi masalah semula

1. 1 Skema Umum Algoritma Divide And Conquer

```
prosedure DIVIDE_n_CONQUER(input n : integer)
{ Masukan: masukan yang berukuran n
 Keluaran: solusi dari masalah semula
}
Deklarasi
        r, k : integer
Algoritma
    if  n ≤ n0 then { masalah sudah cukup kecil }
        SOLVE sub-masalah yang berukuran n ini
    else
        Bagi menjadi r sub-masalah, masing-masing
            Berukuran n/k
        for masing-masing dari r upa-masalah do
            DIVIDE_n_CONQUER(n/k)
        endfor
        COMBINE solusi dari r sub-masalah menjadi
            solusi masalah semula
    endif
```

Kompleksitas algoritma *divide and conquer*: $T(n) = \begin{cases} g(n) & , n \leq n_0 \\ T(n_1) + T(n_2) \ldots + T(n_r) + f(n) & , n > n_0 \end{cases}$

Gambar 2. Skema Umum Algoritma Devide And Conquer

Dari gambar diatas dapat dijelaskan seperti dibawah ini :

$$T(n) = \begin{cases} g(n) & , \ n \leq n_0 \\ T(n1) + T(n2) \ldots + T(nr) + f(n), & n > n_0 \end{cases}$$

1. T(n) : Kompleksitas waktu penyelesaian persoalan P yang berukuran n
2. g(n) : Kompleksitas waktu untuk SOLVE jika n sudah berukuran kecil
3. $T(n_1) + T(n_2) \ldots + T(n_r)$: Kompleksitas waktu untuk memproses setiap upa-persoalan
4. F(n) : Kompleksitas waktu untuk COMBINE solusi dari masing-masing upa-persoalan

Jika pembagian selalu menghasilkan dua sub-masalah yang berukuran sama:

```
procedure DIVIDEandCONQUER(input P : problem, n : integer)
{ Menyelesaikan persoalan dengan algoritma divide and conquer
  Masukan: masukan yang berukuran n
  Luaran: solusi dari persoalan semula
}
Deklarasi
   r : integer

Algoritma
 if n ≤ n0 then {ukuran persoalan sudah cukup kecil }
     SOLVE persoalan P yang berukuran n ini
 else
    DIVIDE menjadi 2 upa-persoalan, P1 dan P2, masing-masing berukuran n/2
    DIVIDEandCONQUER(P1, n/2)
    DIVIDEandCONQUER(P2, n/2)
    COMBINE solusi dari P1 dan P2
 endif
```

Kompleksitas algoritma *divide and conquer*: $T(n) = \begin{cases} g(n) & ,n \leq n_0 \\ 2T(n/2) + f(n) & ,n > n_0 \end{cases}$

1. 2 Penerapan Alogritma Divide and Conquer

1. Pada penyelasaian masalah pencarian Convex Hull dengan menggunakan algoritma Divide and Conquer, hal ini dapat dipandang
2. Sebagai generalisasi dari algoritma pengurutan merge sort. Berikut ini merupakan garis besar gambaran dari algoritmanya:
3. Pertama-tama lakukan pengurutan terhadap titik-titik dari himpunan S yang diberika berdasarkan koordinat absis-X, dengan kompleksitas waktu O(n log n).
4. Jika |S| ≤ 3, maka lakukan pencarian convex hull secara brute-force dengan kompleksitas waktu O(1). (Basis).
5. Jika tidak, partisi himpunan titik-titik pada S menjadi 2 buah himpunan A dan B, dimana A terdiri dari setengah jumlah dari |S| dan titik dengan koordinat absix-X yang terendah dan B terdiri dari setengah dari jumlah |S| dan titik dengan koordinat absis-X terbesar.

6. Secara rekursif lakukan penghitungan terhadap HA = conv(A) dan HB = conv(B).
7. Lakukan penggabungan (merge) terhadap kedua hull tersebut menjadi convex hull, H, dengan menghitung da mencari upper dan lower tangents untuk HA dan HB dengan mengabaikan semua titik yang berada diantara dua buah tangen ini.

Permasalahan convex hull adalah sebuah permasalahan yang memiliki aplikasi terapan yang cukup banyak, seperti pada permasalahan grafika komputer, otomasi desain, pengenalan pola (pattern recognition), dan penelitian operasi. Divide and Conquer adalah metode pemecahan masalah yang bekerja dengan membagi masalah menjadi beberapa upa-masalah yang lebih kecil, kemudian menyelesaikan masing-masing upa-masalah tersebut secara independent, dan akhirnya menggabungkan solusi masing-masing upa-masalah sehingga menjadi solusi dari masalah semula.

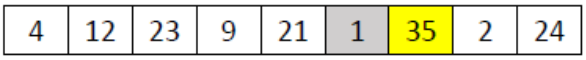
Algoritma Divide and Conquer merupakan salah satu solusi dalam penyelesaian masalah convex hull. Algoritma ini ternyata memiliki kompleksitas waktu yang cukup kecil dan efektif dalam menyelesaikan permasalahan ini (jika dibandingkan algoritma lain). Selain itu juga, algoritma ini dapat digeneralisasi untuk permasalahan convex hull yang berdimensi lebih dari 3.

2. Contoh soal dengan menggunakan Algoritma Divide and Conquer

2.1 Contoh soal mencari nilai minimum dan maksimum (MinMax)

Misalkan diberikan tabel A yang berukuran n elemen dan sudah berisi nilai integer. Carilah nilai minimum dan nilai maksimum sekaligus di dalam tabel tersebut.

Contoh:



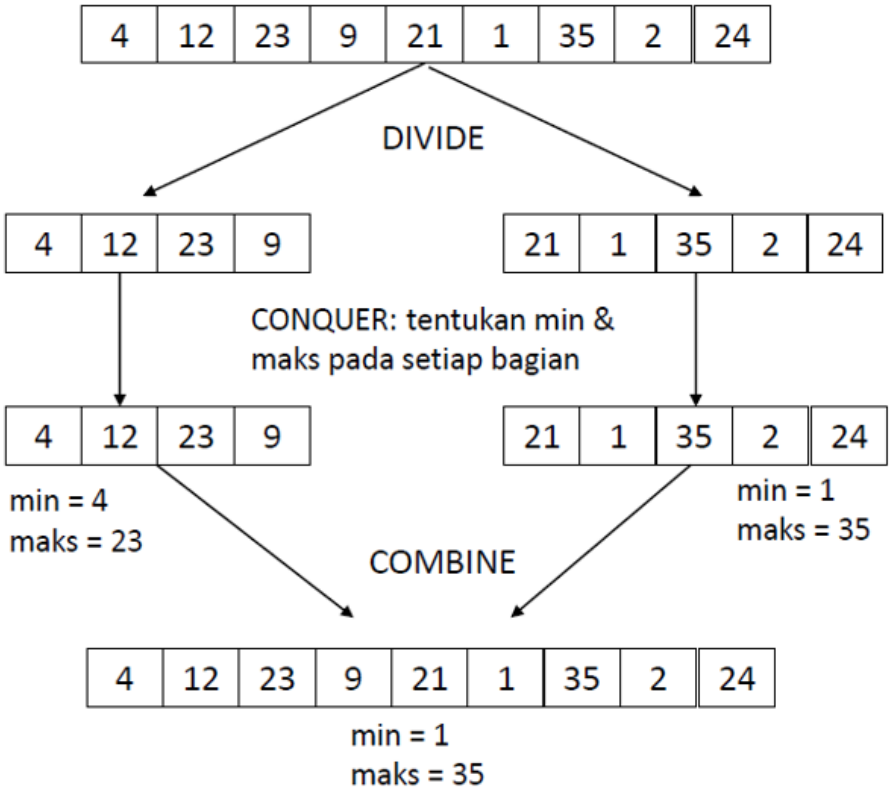
min = 1
max = 35

Jika psudeocode penyelesaian dengan menggunakan algoritma brute force adalah sebagai berikut :

```
procedure MinMaks1(input A : TabelInt, n : integer,
                   output min, maks : integer)
{ Mencari nilai minimum dan maksimum di dalam tabel A yang berukuran n
elemen, secara brute force.
Masukan: tabel A yang sudah terdefinisi elemen-elemennya
Keluaran: nilai maksimum dan nilai minimum tabel
}
Deklarasi
    i : integer

Algoritma:
    min← A1 { inisialisasi nilai minimum}
    maks←A1 { inisialisasi nilai maksimum }
    for i←2 to n do

      if Ai < min then
        min←Ai
      endif

      if Ai > maks then
        maks←Ai
      endif

   endfor
```

$$T(n) = (n-1) + (n-1) = 2n - 2 = O(n)$$

Jika dilihat dalam bentuk gambar dengan menggunakan algoritma Divide and Conquer adalah sebagai berikut :

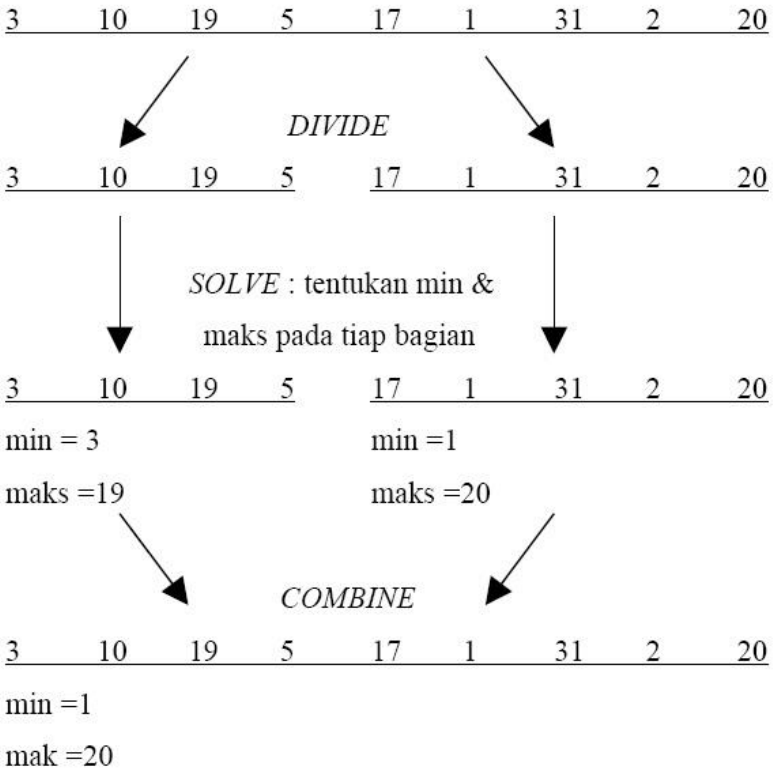


Seperti persoalan yang ada diatas, maka 9 angka tersebut dipecah menjadi 2 maka menghasilkan 2 blok, antara blok kiri dan blok kanan yang berisikan masing-masing 4 dan 5 anggota. Setelah dibagi menjadi 2, kemudian masuk kedalam proses CONQUER dimana menentukan nilai minimal dan maksimal pada setiap bagiannya, kasus diatas nilai minimal pada blok bagian kiri adalah 4 dan nilai maksimalnya adalah 23, sementara itu nilai minimal pada blok bagian kanan adalah 1, dan nilai maksimalnya adalah 35, jika sudah menemukan nilai MinMax pada masing-masing

blok, maka terjadi proses COMBINE dimana proses ini menyatukan kembali tiap bloknya sehingga dapat ditemukan nilai minimalnya adalah 1 dan nilai maksimalnya adalah 35.

Contoh soal MinMax lain adalah sebagai berikut :

Terdapat n=9 dengan nilai yaitu 3,10,19,5,17,1,31,2,20. Untuk dapat menentukan nilai MinMaxnya dapat dilihat pada gambar dibawah ini :



Sama seperti kasus sebelumnya, dikasus ini dibagi menjadi 2 blok, antara blok kiri dan blok kanan sehingga menghasilkan 4 nilai untuk blok kiri dan 5 nilai untuk blok kanan. Setelah itu lakukan SOLVE yaitu mencari nilai minimal dan maksimal, sehingga pada masing-masing blok sudah menemukan nilai MinMaxnya, dan COMBINE kembali nilai yang tadi dipisah, setelah digabungkan kembali akan menghasilkan nilai minimal yaitu 1 dan maksimalnya yaitu 20.

Berikut ini adalah contoh kompleksitas waktu MinMax2:

$$T(n) = \begin{cases} 0 & , n = 1 \\ 1 & , n = 2 \\ 2T(n/2) + 2 & , n > 2 \end{cases}$$

Penyelesaian:

Asumsi: $n = 2^k$, dengan k bilangan bulat positif, maka

$$\begin{aligned} T(n) &= 2\mathrm{T}(n/2) + 2 \\ &= 2(2T(n/4) + 2) + 2 = 4T(\mathrm{n}/4) + 4 + 2 \\ &= 4T(2\mathrm{T}(n/8) + 2) + 4 + 2 = 8T(n/8) + 8 + 4 + 2 \\ &= \ldots \\ &= 2^{k-1}\, T(2) + \sum_{i=1}^{k-1} 2^i \\ \\ &= 2^{k-1} \cdot 1 + 2^k - 2 \\ &= n/2 + n - 2 \\ &= 3n/2 \ - 2 \\ &= O(n) \end{aligned}$$

2.2 Perbandingan Algoritma MinMax

1. MinMaks1 secara *brute force* :

   $T(n) = 2n - 2$

2. MinMaks2 secara *divide and conquer*:

   $T(n) = 3n/2 \ - 2$

3. Perhatikan: $3n/2 - 2 < 2n - 2$, $n \geq 2$.

4. Kesimpulan: algoritma MinMaks lebih mangkus/efisien dengan metode *Divide and Conquer*.

2.3 Persoalan Perpangkatan a^n

1. Misalkan a Є R dan n adalah bilangan bulat tidak negatif, maka perpangkatan an didefinisikan sebagai berikut:

$$a^n = \begin{cases} 1, & n = 0 \\ a \times a \times \cdots \times a, & n > 0 \end{cases}$$

Bagaimana algoritma menghitung perpangkatan an secara brute force dan secara divide and conquer?

(a) Penyelesaian dengan menggunakan algoritma Brute Force

```
function Exp1(a : real, n : integer)→ real
{ Menghitung a^n, a > 0 dan n bilangan bulat tak-negatif }

Deklarasi
   k : integer
   hasil : real

Algoritma:
  hasil ← 1
  for k ← 1 to n do
      hasil ← hasil * a
 endfor

 return hasil
```

Kompleksitas algoritma, dihitung dari jumlah operasi perkalian : T(n) = n = O(n)

(b) Penyelesaian dengan menggunakan algoritma Divide and Conquer

1. Ide dasar : bagi dua pangkat n menjadi n = n/2 + n/2

$$a^n = a^{(n/2 + n/2)} = a^{n/2} \cdot a^{n/2}$$

2. Algoritma divide and conquer untuk menghitung an:
3. 1. Untuk kasus n = 0, maka an = 1.
4. 2. Untuk kasus n > 0, bedakan menjadi dua kasus lagi:
5. (i) jika n genap , maka an = an/2. an/2
6. (ii) jika n ganjil , maka an = an/2. an/2. A
7.

2. Contoh soal perpangkatan a^n

1. $2^4 = 2^{2}.2^{2}$

$= (2^{2})^2$

$= ((2^{1})^2)^2$

$= ((2^{0})2 . ^{2})^2)^2$

$= (((1)^2 . 2)^2)^2$

$= (22)^2$

$= (4)^2$

$= 16$

Dari soal diatas terdapat 2^4 sehingga awalnya adalah $2^2.2^2$ diturunkan menjadi $(2^{2})^2$ diturunkan lagi menjadi $((2^{1})^2)^2$ sehingga menghasilkan $((2^{0})2 . ^{2})^2)^2$ kemudian diturunkan lagi menjadi $(((1)^2 . 2)^2)^2$ diturunkan lagi menjadi $(22)^2$ karna masih ada pangkatnya maka diturunkan lagi menjadi $(4)^2$ sehingga menghasilkan 16, jadi 2 pangkat 4 adalah 16.

2.1 Pseudocode menghitung a^n dengan divide and conquer

```
function Exp2(a : real, n : integer) → real
{ mengembalikan nilai a^n, dihitung dengan metode Divide and Conquer }

Algoritma:
  if n = 0 then
    return 1
  else
    if odd(n) then   { kasus n ganjil }
        return Exp2(a, n div 2) * Exp2(a, n div 2) * a      {a^n = a^(n/2) · a^(n/2) · a}
    else             { kasus n genap }
        return Exp2(a, n div 2) * Exp2(a, n div 2)          {a^n = a^(n/2) · a^(n/2) }
    endif
  endif
```

Fungsi *Exp2* tidak mangkus, sebab terdapat dua kali pemanggilan rekursif untuk nilai parameter yang sama → *Exp2*(*a*, *n* **div** 2) * *Exp2*(*a*, *n* **div** 2).

Penyesuaian simpan hasil *Exp2*(*a*, *n* **div** 2) di dalam sebuah peubah (misalkan x), lalu gunakan x untuk menghitung a^n pada kasus *n* genap dan *n* ganjil.

```
function Exp3(a : real, n : integer) → real
{ mengembalikan nilai a^n, dihitung dengan metode Divide and Conquer }

Deklarasi
   x : real
Algoritma:
  if n = 0 then
    return 1
  else
    x ← Exp3(a, n div 2)
    if odd(n) then   { kasus n ganjil }
        return x * x * a
    else             { kasus n genap }
        return x * x
    endif
  endif
```

Kompleksitas algoritma Exp3 dihitung dari jumlah operasi perkalian:

$$T(n) = \begin{cases} 0, & n = 0 \\ T\left(\frac{n}{2}\right) + 2, & n > 0 \text{ dan } n \text{ ganjil} \quad x * x * a \\ T\left(\frac{n}{2}\right) + 1, & n > 0 \text{ dan } n \text{ genap} \quad x * x \end{cases}$$

Dalam menghitung T(n) ini ada sedikit kesulitan, yaitu nilai n mungkin ganjil atau genap, sehingga penyelesain relasi rekurens menjadi lebih rumit. Namun, perbedaan ini dianggap kecil sehingga dapat kita abaikan. Sebagai implikasinya, kita membuat asumsi penghampiran bahwa untuk n genap atau ganjil, jumlah operasi perkalian relatif sama.

Sehingga, kompleksitas algoritma Exp3 menjadi:

$$T(n) = \begin{cases} 0, & n = n_0 \\ T\left(\frac{n}{2}\right) + 1, & n > n_0 \end{cases}$$

Asumsikan n adalah perpangkatan dari 2, atau $n = 2^k$, maka

$T(n) = 1 + T(n/2)$

$= 1 + (1 + T(n/4) = 2 + T(n/4)$

$= 2 + (1 + T(n.8) = 3 + T(n/8)$

$= \ldots$

$= k + T(n/2^k)$

Kerna $n = 2^k$ maka $k = {}^2\log n$, sehingga

$= k + T(n/2^k) = {}^2\log n + T(1)$

$= {}^2\log n + (1 + T(0)) = {}^2\log n + 1 + 0$

$= {}^2\log n + 1 = O({}^2\log n)$ -> lebih baik daripada algoritma brute force