BAB VII

ALGORITMA DIVIDE AND CONQUER

Pemrogram bertanggung jawab atas implementasi solusi. Pembuatan program akan menjadi lebih sederhana jika masalah dapat dipecah menjadi sub masalah - sub masalah yang dapat dikelola.
Penyelesaian masalah dengan komputer berhadapan dengan 4 hal, yaitu :

1. Pemahaman keterhubungan elemen-elemen data yang relevan terhadap solusi secara menyeluruh.
2. Pengambilan keputusan mengenai operasi-operasi yang dilakukan terhadap elemen-elemen data.
3. Perancangan representasi elemen-elemen data di memori sehingga memenuhi kriteria berikut:
   a. Memenuhi keterhubungan logik antara elemen-elemen data.
   b. Operasi-operasi terhadap elemen-elemen data dapat dilakukan secara mudah dan efisien.
4. Pengambilan keputusan mengenai mengenai bahasa pemrograman terbaik untuk menerjemahkan solusi persoalan menjadi program.

7.1. STRATEGI DIVIDE AND CONQUER

Algoritma Divide and Conquer (DANDC) memecah masalah menjadi submasalah-submasalah independen yang lebih kecil sehingga solusi submasalah–submasalah dapat diperoleh secara mudah. Solusi dari submasalah–submasalah kemudian digabung menjadi solusi dari seluruh masalah.

Prinsip dasar dari algoritma ini adalah dengan membagi n input menjadi k subset input yang berbeda ($1 < k \leq n$). Dari k subset input yang berbeda akan terdapat k subproblem. Setiap subproblem mempunyai solusinya masing-masing, sehingga akan diperoleh k subsolusi. Kemudian, dari k subsolusi akan diperoleh solusi yang optimal atau solusi yang diharapkan.

Jika subproblem dianggap masih terlalu besar, maka metode DANDC dapat digunakan lagi. Dalam keadaan tersebut, pemakaian ulang metode DANDC dinyatakan menggunakan teknik rekursif.

Pemecahan n input menjadi k input sehingga menimbulkan k subproblem dapat dilakukan apabila k subproblem tersebut mempunyai sifat yang sama terhadap persoalan semula atau awalnya.

Skema umum algoritma divide dan conquer

```
Procedure DNC ( i,j : integer )
Var K : integer ;
        If  SMALL (i,j) then SOLVE (i,j)
        Else begin
                K : = DIVIDE (i,j)
                COMBINE (DNC(i,k),DNC(k+1,j))
        End if
```

Keterangan :
1. SMALL adalah fungsi yang mengirim BOOLEAN, menentukan apakah ukuran telah cukup kecil sehingga solusi dapat diperoleh. Ukuran dinyatakan sebagai telah berukuran kecil bergantung masalah.
2. DIVIDE adalah fungsi membagi menjadi 2 bagian pada posisi K. Biasanya bagian berukuran sama.
3. COMBINE adalah fungsi menggabungkan solusi X dan Y submasalah. Solusi diperoleh dengan memanggil prosedur rekursif DNC.

Jika ukuran kedua submasalah sama, waktu komputasi DNC dideskripsikan hubungan rekuren berikut :

$$T(n) = \begin{cases} g(n), & n \text{ kecil} \\ 2\,T(n/2) + f(n), & \text{selainnya} \end{cases}$$

dimana :
- T(n) adalah waktu untuk DNC dengan n masukan,
- g(n) adalah waktu komputasi jawaban secara langsung untuk masukan kecil dan
- f(n) adalah waktu COMBINE.

Untuk algoritma divide dan conquer yang menghasilkan submasalah-submasalah dengan tipe masalah yang sama dengan masalah awal, sangat alami untuk mendeskripsikan algoritma secara rekursi. Kemudian untuk meningkatkan efisiensi dilakukan penerjemahan menjadi bentuk iterasi.

Pemakaian teknik Divide dan Conquer banyak digunakan dalam menyelesaikan berbagai macam persoalan, antara lain :
1. Sorting/Pengurutan → Quick Sort
2. Searching/Pencarian → Binary Search

7.2. ALGORITMA QUICK SORT

Jika suatu barisan yang terdiri dari n elemen yang ditempatkan dalam suatu array dan ururtan yag diinginkan adalah urutan yang tidak menurun (non decreasing) maka dapat digunakan metode Quick Sort yang menggunakan teknik DANDC.

Quick Sort merupakan suatu algoritma pengurutan data yang menggunakan teknik pemecahan data menjadi partisi-partisi, sehingga metode ini disebut juga dengan nama *partition exchange sort*. Metode ini diperkenalkan pertama kali oleh C.A.R. Hoore pada tahun 1962.

Untuk memulai iterasi pengurutan, pertama-tama pilih sebuah elemen dari data yang akan diurutkan. Misalnya variabel x berisi elemen yang dipilih dari data, kemudian elemen-elemen data yang akan diurutkan diatur sedemikian rupa, sehingga nilai variabel x berada di suatu posisi ke I yang memenuhi kondisi sebagai berikut:
1. Semua elemen di posisi ke-1 sampai dengan ke I-1 adalah lebih kecil atau sama dengan x.
2. semua elemen di posisi ke I+1 samap dengan N adalah lebih besar atau sama dengan x.

Untuk mendapatkan kondisi di atas, lakukan algoritma sebagai berikut:
1. x = data[bawah]
2. i = bawah
3. j = atas

4. selama i < j, kerjakan langkah 5 – 8
5. selama data[j] > x kerjakan j = j – 1
6. tukar data[i] dengan data[j]
7. selama data[i] < x kerjakan i = i +1
8. tukar data[j] dengan data[i]

Contoh:

1. Urutkan data pada array di bawah ini yang terdiri dari 9 elemen:
   A(1) = 65 A(4) = 80 A(7) = 55
   A(2) = 70 A(5) = 85 A(8) = 50
   A(3) = 75 A(6) = 60 A(9) = 45

2. Urutkan data pada array di bawah ini yang terdiri dari 12 elemen:
   A(1) = 33 A(4) = 7 A(7) = 57 A(10) = 10
   A(2) = 45 A(5) = 5 A(8) = 25 A(11) = 40
   A(3) = 18 A(6) = 99 A(9) = 55 A(12) = 50

7.3. ALGORITMA BINARY SEARCH

Binary Search (Pencarian Biner) dapat dilakukan jika data sudah dalam keadaan urut. Dengan kata lain, apabila data belum dalam keadaan urut, pencarian biner tidak dapat dilakukan. Dalam kehidupan sehari-hari, sebenarnya kita juga sering menggunakan pencarian biner. Misalnya saat ingin mencari suatu kata dalam kamus.

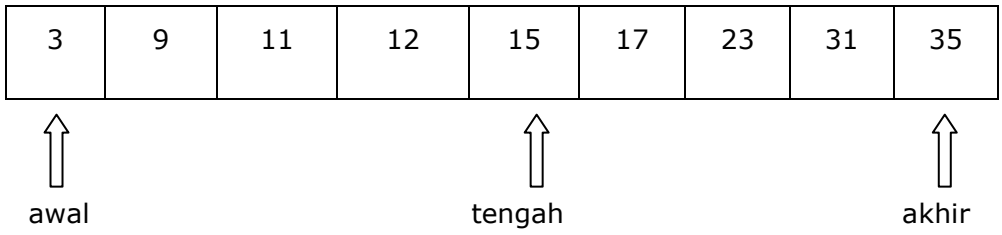
Prinsip dari pencarian biner dapat dijelaskan sebagai berikut :

1. Mula-mula diambil posisi awal = 1 dan posisi akhir = N
2. Cari posisi data tengah dengan rumus (posisi awal + posisi akhir) / 2
3. Data yang dicari dibandingkan dengan data tengah.
4. Jika lebih kecil, proses dilakukan kembali tetapi posisi akhir dianggap sama dengan posisi tengah – 1.
5. Jika lebih besar, proses dilakukan kembali tetapi posisi awal dianggap sama dengan posisi tengah + 1.
6. Demikian seterusnya sampai data tengah sama dengan yang dicari.

Untuk lebih jelasnya, perhatikan contoh berikut. Misalkan kita ingin mencari 17 pada sekumpulan data berikut :

Iterasi 1:

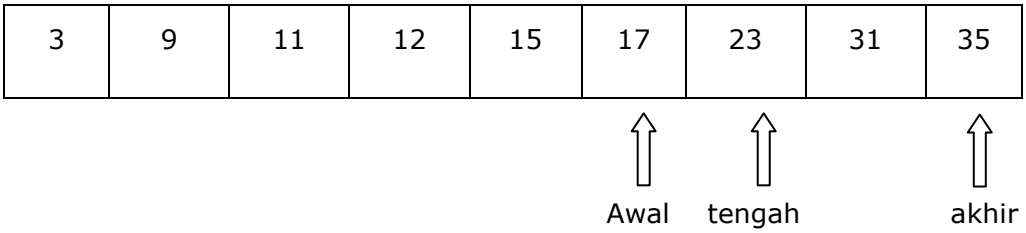
1. Mula–mula dicari data tengah, dengan rumus (1+ 9) / 2 = 5.
2. Berarti data tengah adalah data ke-5, yaitu 15.



3. Data yang dicari, yaitu 17, bandingkan dengan data tengah ini.
4. Karena 17 > 15, berarti proses dilanjutkan tetapi kali ini posisi awal dianggap sama dengan posisi tengah + 1 atau 6.

Iterasi 2:

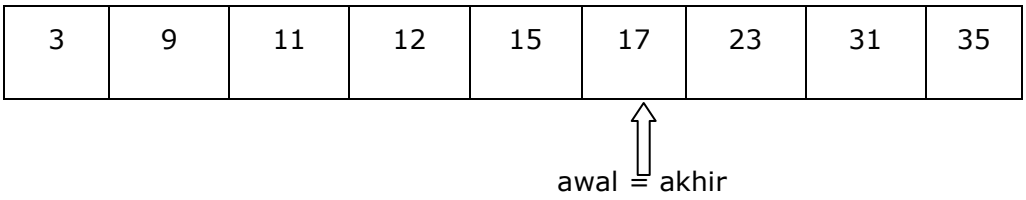
1. Data tengah yang baru didapat dengan rumus (6 + 9) / 2 = 7. Berarti data tengah yang baru adalah data ke-7, yaitu 23.



2. Data yang dicari, yaitu 17 dibandingkan dengan data tengah ini.
3. Karena 17 < 23, berarti proses dilanjutkan tetapi kali ini posisi akhir dianggap sama dengan posisi tengah – 1 atau 6.

Iterasi 3:

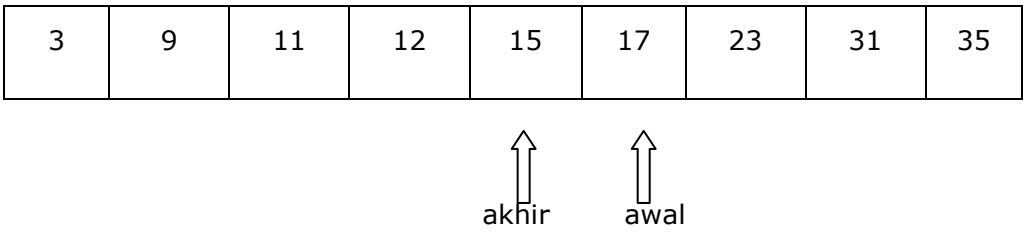
1. Data tengah yang baru didapat dengan rumus (6 + 6) / 2 = 6. Berarti data tengah yang baru adalah data ke-6, yaitu 17.



2. Data yang dicari dibandingkan dengan data tengah ini dan ternyata sama. Jadi data ditemukan pada indeks ke-6.

Bagaimana jika data yang dicari tidak ada, misalnya 16?
Pencarian biner ini akan berakhir jika data ditemukan atau posisi awal lebih besar dari posisi akhir. Jika posisi awal sudah lebih besar daripada posisi akhir berarti data tidak ditemukan.

Untuk lebih jelasnya perhatikan proses pencarian 16 pada data di atas. Prosesnya hampir sama dengan pencarian 17. Tetapi setelah posisi awal = posisi akhir = 6, proses masih dilanjutkan lagi dengan posisi awal = 6 dan posisi akhir = 5



Disini dapat dilihat bahwa posisi awal lebih besar daripada posisi akhir, yang artinya data tidak ditemukan.

Berikut ini adalah contoh fungsi untuk mencari data menggunakan pencarian biner.

```
Function BinarySearch (x: word) : integer;
var
      l, r, m : word;
      ketemu : boolean;
begin
      l : = 1;
      r : = N;
      ketemu : = false;
      while (1 <= r )  and  ( not ketemu )  do
      begin
            m : = (1 + r )  div  2;
            if  (Data [m] = x ) then
                  Ketemu := true
            else if  (x  <  Data [m] ) then
                  r : = m – 1
            else
                  l : = m + 1;
      end;

      if  ( ketemu )  then
            BinarySearch : = m
      else
            BinarySearch : = -1;
end;
```

Fungsi di atas akan mengembalikan indeks dari data yang dicari. Apabila data tidak ditemukan, maka yang yang dikembalikan adalah –1.

Jumlah pembandingan minimum pada pencarian biner adalah 1 kali, yaitu bila data yang dicari tepat berada di tengah-tengah. Jumlah pembandingan maksimum yang dilakukan dengan pencarian biner dapat dicari dengan rumus logaritma, yaitu C = $^{2}\log(N)$