PERTEMUAN 10

METODE DIVIDE AND CONQUER

Metode D And C

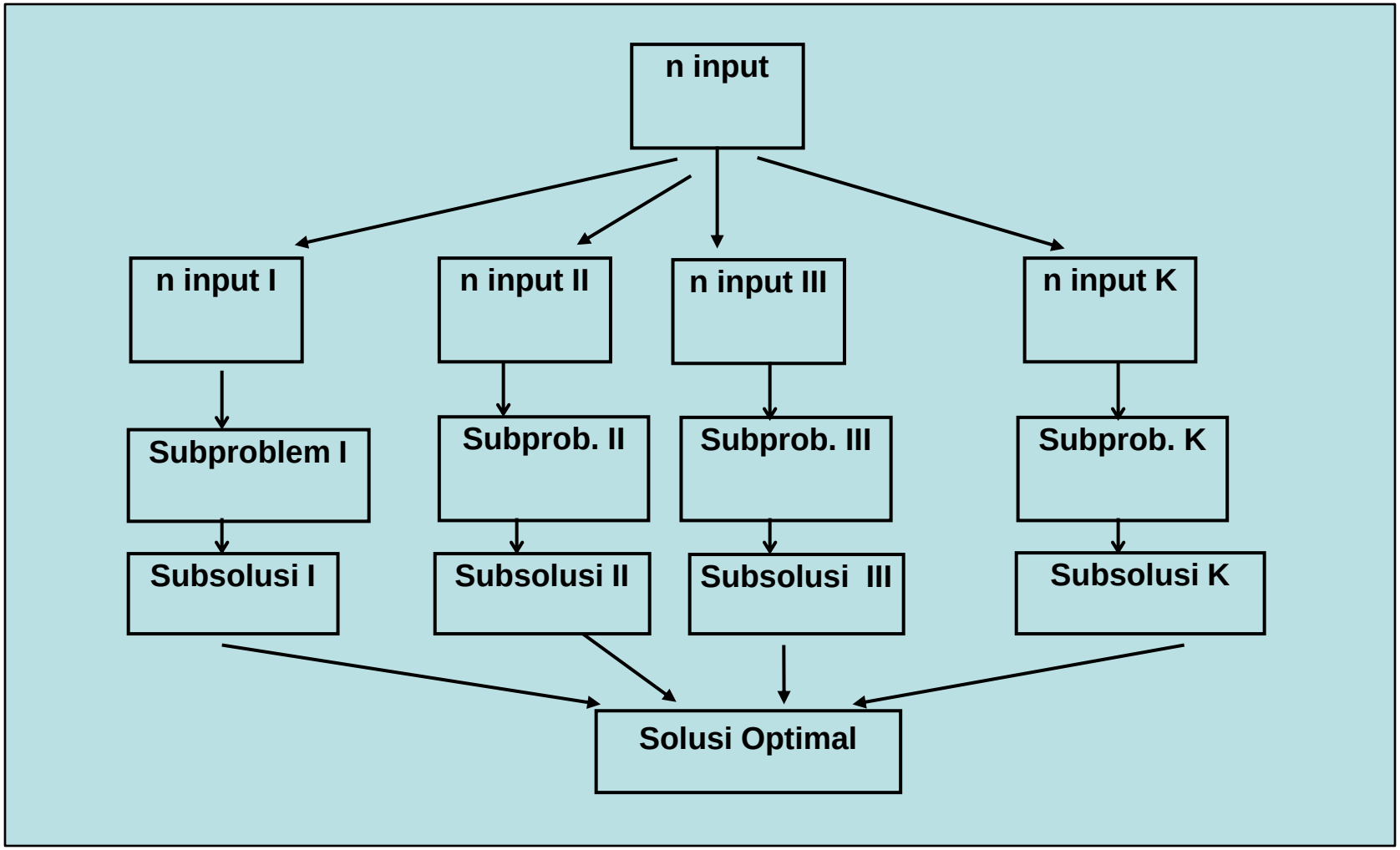
Divide
Memilah data nilai elemen–elemen dari rangkaian data menjadi dua bagian dan mengulangi pemilahan hingga satu elemen terdiri maksimal dua nilai (Sonita & Nurtaneo, 2015).

Conquer
Mengurutkan masing-masing data nilai elemen (Sonita & Nurtaneo, 2015).

Prinsip Dasar
- Membagi n input menjadi k subset input yang berbeda ($1<k\leq n$).
- k subset input tersebut akan terdapat k subproblem.
- Setiap subproblem mempunyai solusi menjadi k subsolusi.
- Dari k subsolusi akan mendapatkan solusi yang optimal

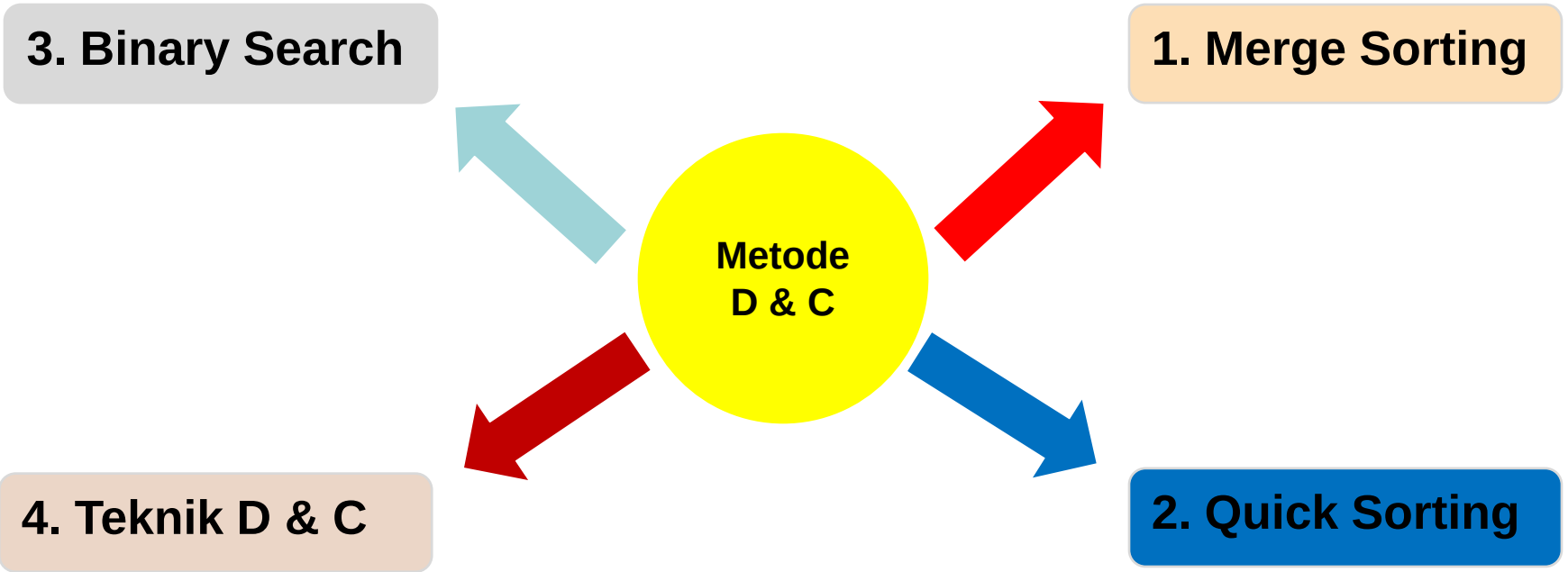
Jika subproblem masih besar → D and C

Bentuk Umum Metode D and C



Metode D AND C Lanjutan

Metode D AND C
Menggunakan teknik Rekursif yang membagi masalah menjadi dua atau lebih submasalah dengan ukuran yang sama. Masalah umum untuk teknik ini seperti pengurutan, perkalian.



MERGE SORT

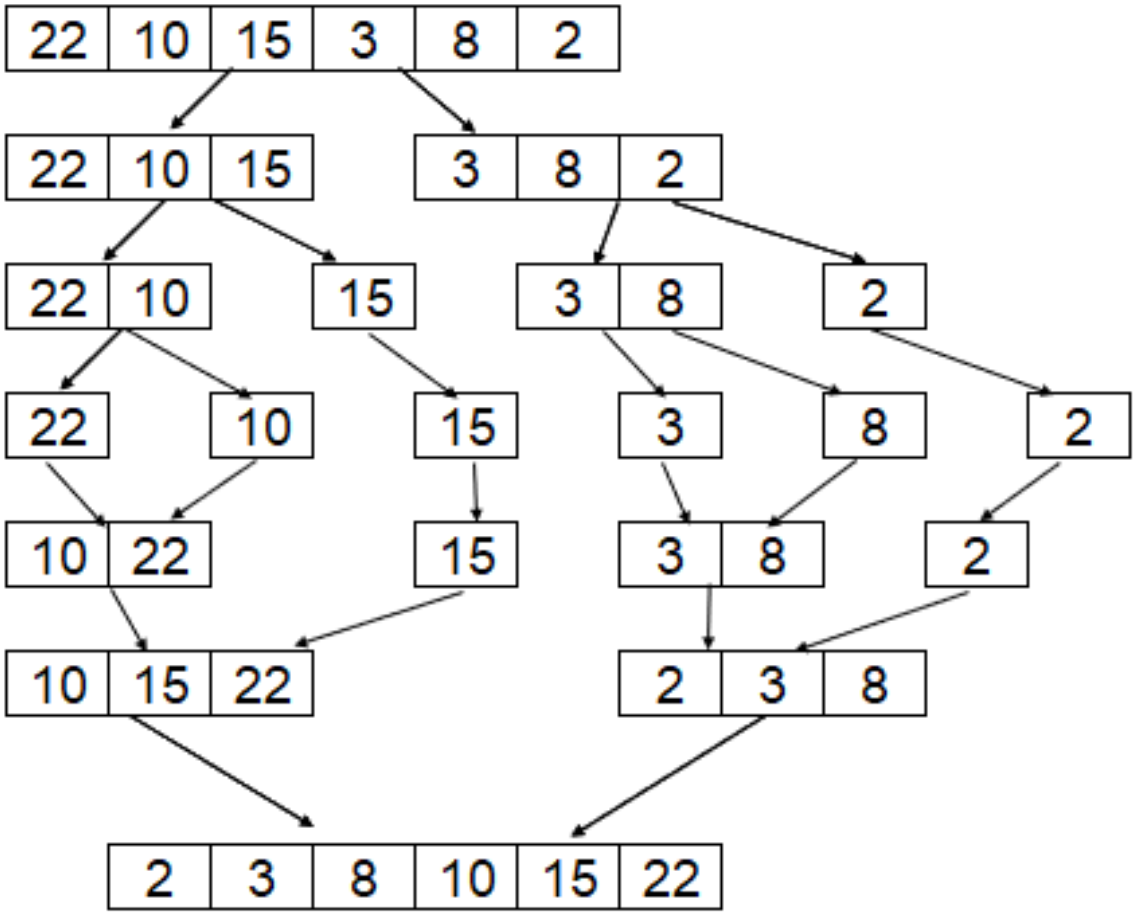
- Menggabungkan dua array yang sudah terurut (Utami, 2017)
- Metode merge sort merupakan metode yang membutuhkan fungsi rekursif untuk penyelesaiannya.

Prinsip Kerja Merge Sort :

1. Kelompokkan deret bilangan kedalam 2 bagian, 4 bagian, 8 bagian, ...dst.
2. Lakukan pengurutan sesuai dengan kelompok sebelumnya
3. Lakukan pengurutan sejumlah pembagian

MERGE SORT (Lanjutan)

Studi Kasus 1:



MERGE SORT (Lanjutan)

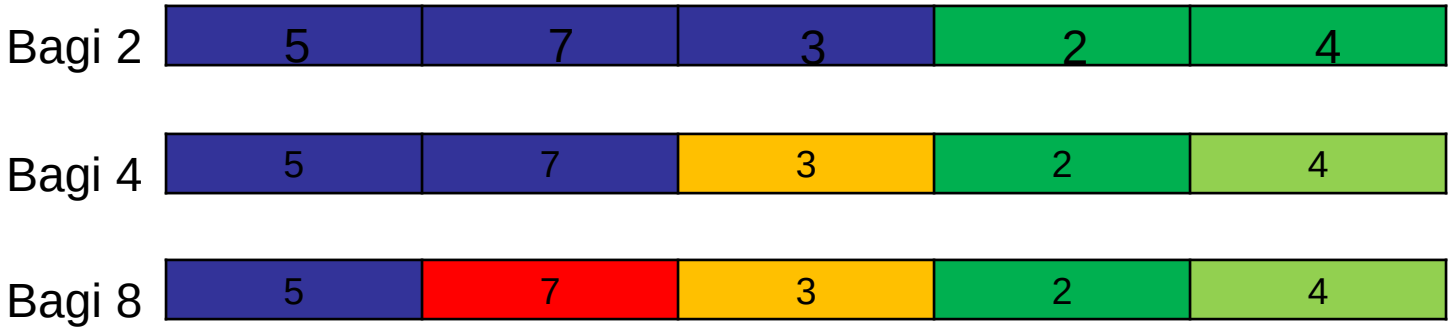
Studi Kasus 2:

Awal	5	7	3	2	4

Prinsip Kerja Merge Sort:

1. Kelompokkan deret bilangan kedalam 2 bagian, 4 bagian, 8 bagian, ...dst.

Hasilnya:

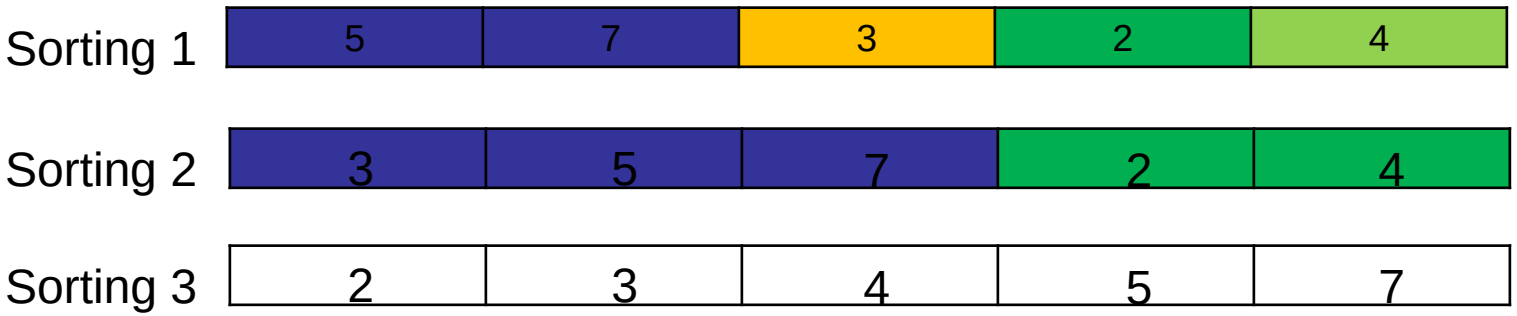


MERGE SORT (Lanjutan)

Prinsip Kerja Merge Sort:

2. Lakukan pengurutan sesuai dengan kelompok sebelumnya
3. Lakukan pengurutan sejumlah pembagian

Hasilnya:



MERGE SORT (Lanjutan)

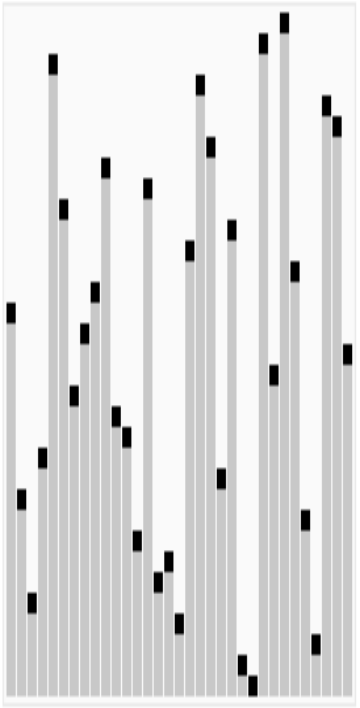
```
def mergeSort(X):
    print("Bilangan diurutkan ",X)
    if len(X)>1:
        mid = len(X)//2
        lefthalf = X[:mid]
        righthalf = X[mid:]
        mergeSort(lefthalf)
        mergeSort(righthalf)
        i=j=k=0
        while i < len(lefthalf) and j < len(righthalf):
            if lefthalf[i] < righthalf[j]:
                X[k]=lefthalf[i]
                i=i+1
            else:
                X[k]=righthalf[j]
                j=j+1
            k=k+1
while i < len(lefthalf):
            X[k]=lefthalf[i]
            i=i+1
            k=k+1
            while j < len(righthalf):
            X[k]=righthalf[j]
            j=j+1
            k=k+1
            print("Merging ",X)
X = [22,10,15,3,8,2]
mergeSort(X)
print(X)
```

Hasil Program:

```
Bilangan diurutkan  [22, 10, 15, 3, 8, 2]
Bilangan diurutkan  [22, 10, 15]
Bilangan diurutkan  [22]
Bilangan diurutkan  [10, 15]
Bilangan diurutkan  [10]
Bilangan diurutkan  [15]
Merging  [10, 15]
Bilangan diurutkan  [3, 8, 2]
Bilangan diurutkan  [3]
Bilangan diurutkan  [8, 2]
Bilangan diurutkan  [8]
Bilangan diurutkan  [2]
Merging  [2, 3, 8]
[2, 3, 8, 10, 15, 22]
```


QUICK SORTING

- Merupakan metode yang tercepat
- Quicksort diperkenalkan oleh C.A.R. Hoare. Quicksort partition exchange sort, karena konsepnya membuat bagian-bagian, dan sort dilakukan perbagian.
- Pada algoritma quick sort, pemilihan pivot merupaka hal yang menentukan apakah algoritma quicksort tersebut akan memberikan performa terbaik atau terburuk (Nugraheny, 2018).



QUICK SORTING (Lanjutan)

Misal ada N elemen dalam keadaan urut turun, adalah mungkin untuk mengurutkan N elemen tersebut dengan N/2 kali, yakni pertama kali menukarkan elemen paling kiri dengan paling kanan, kemudian secara bertahap menuju ke elemen yang ada di tengah. Tetapi hal ini hanya bisa dilakukan jika tahu pasti bahwa urutannya adalah urut turun.

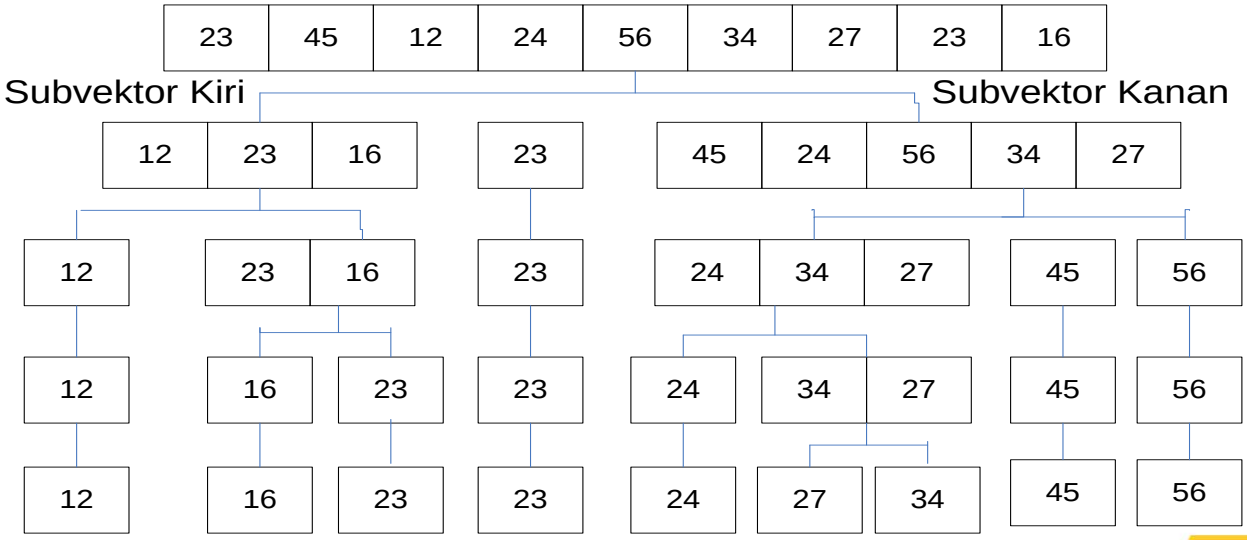
Secara garis besar metode ini dijelaskan sebagai berikut,

Misal: akan mengurutkan vektor A yang mempunyai N elemen. Pilih sembarang dari vektor tsb, biasanya elemen pertama misalnya X. Kemudian semua elemen tersebut disusun dengan menempatkan X pada posisi J sedemikian rupa sehingga elemen ke 1 sampai ke j-1 mempunyai nilai lebih kecil dari X dan elemen ke J+1 sampai ke N mempunyai nilai lebih besar dari X.

QUICK SORTING (Lanjutan)

Dengan demikian mempunyai dua buah subvektor, subvektor pertama nilai elemennya lebih kecil dari X, subvektor kedua nilai elemennya lebih besar dari X.
Pada langkah berikutnya, proses diatas diulang pada kedua subvektor, sehingga akan mempunyai empat subvektor. Proses diatas diulang pada setiap subvektor sehingga seluruh vektor semua elemennya menjadi terurutkan.

Studi Kasus 1:



QUICK SORTING (Lanjutan)

Studi Kasus 2

Pilih vektor X → elemen pertama

Iterasi 1:

22	**10**	**15**	**3**	**8**	**2**
22					
10	**15**	**3**	**8**	**2**	**22**

QUICK SORTING (Lanjutan)

Iterasi 2 Pilih lagi vektor X berikutnya

10	15	3	8	2	22
10					
3	8	2	10	15	22

Iterasi 3 Pilih lagi vektor X berikutnya

3	8	2	10	15	22
3					
2	3	8	10	15	22

BINARY SEARCH

Binary Search (Untuk data yang sudah terurut)

Digunakan mencari sebuah data pada himpunan data-data yang tersusun secara urut, yaitu data yang telah diurutkan dari besar ke kecil/sebaliknya. Proses dilaksanakan pertama kali pada bagian tengah dari elemen himpunan, jika data yang dicari ternyata < elemen bagian atasnya, maka pencarian dilakukan dari bagian tengah ke bawah.

Binary Search (Lanjutan)

Algoritma Binary Search:

1. Low = 1 , High = N
2. Ketika Low <= High Maka kerjakan langkah No .3, Jika tidak Maka kerjakan langkah No.7
3. Tentukan Nilai Tengah dengan rumus
   (Low + High) Div 2
4. Jika X < Nilai Tengah, Maka High = Mid –1, Ulangi langkah ke-1
5. Jika X > Nilai Tengah, Maka Low = Mid +1, Ulangi langkah ke-1
6. Jika X = Nilai Tengah, Maka Nilai Tengah = Nilai yang dicari
7. Jika X > High Maka Pencarian GAGAL

Binary Search (Lanjutan)

Contoh:

Data A = { 1, 3, 9, 11, 15, 22, 29, 31, 48 }

Dicari **3**

Langkah Pencariannya:

Langkah 1: Low = 1 dan High = 9

Langkah 2: Low <= High (jika YA ke **L-3**, jika TDK ke **L-7**)

Langkah 3: Mid = (1+9) div 2 = 5 yaitu **15**

Langkah 4: 3 < 15, maka High = 5 – 1 = 4 yaitu **11**

Langkah 1: Low = 1 dan High = 4

Langkah 2: Low <= High

Langkah 3: Mid = (1+4) div 2 = 2 yaitu **3**

Langkah 4: 3 < 3, ke langkah 5

Langkah 5: 3 > 3, ke langkah 6

Langkah 6: 3 = 3 (Pencarian berhasil)

Binary Search (Lanjutan)

Kode Program Binary Search

```
def BinSearch(data, key):
        awal = 1
        akhir = len(data) + 1
        ketemu = False
        while (awal <= akhir) and not ketemu:
                tengah = int((awal + akhir)/ 2)
                if key == data[tengah]:
                        ketemu = True
                        print('data', key, 'ditemukan diposisi', tengah+1)
                elif key < data[tengah]:
                        akhir = tengah - 1
                else:
                        awal = tengah + 1
        if not ketemu:
                print('data tidak ditemukan')
```

```
data = [1,  3,  9,  11,  15,  22,  29,  31,  48 ]
BinSearch(data, 3)
```

Hasil Program:

```
data 3 ditemukan diposisi 2
```

Teknik D AND C

Teknik D AND C

Dengan Prinsip Dasar Metode Divide & Conquer akan dapat dipecahkan suatu permasalahan proses Searching elemen Max&Min dengan teknik D and C

Menghasilkan solusi optimal menemukan nilai Maximum dan Minimum

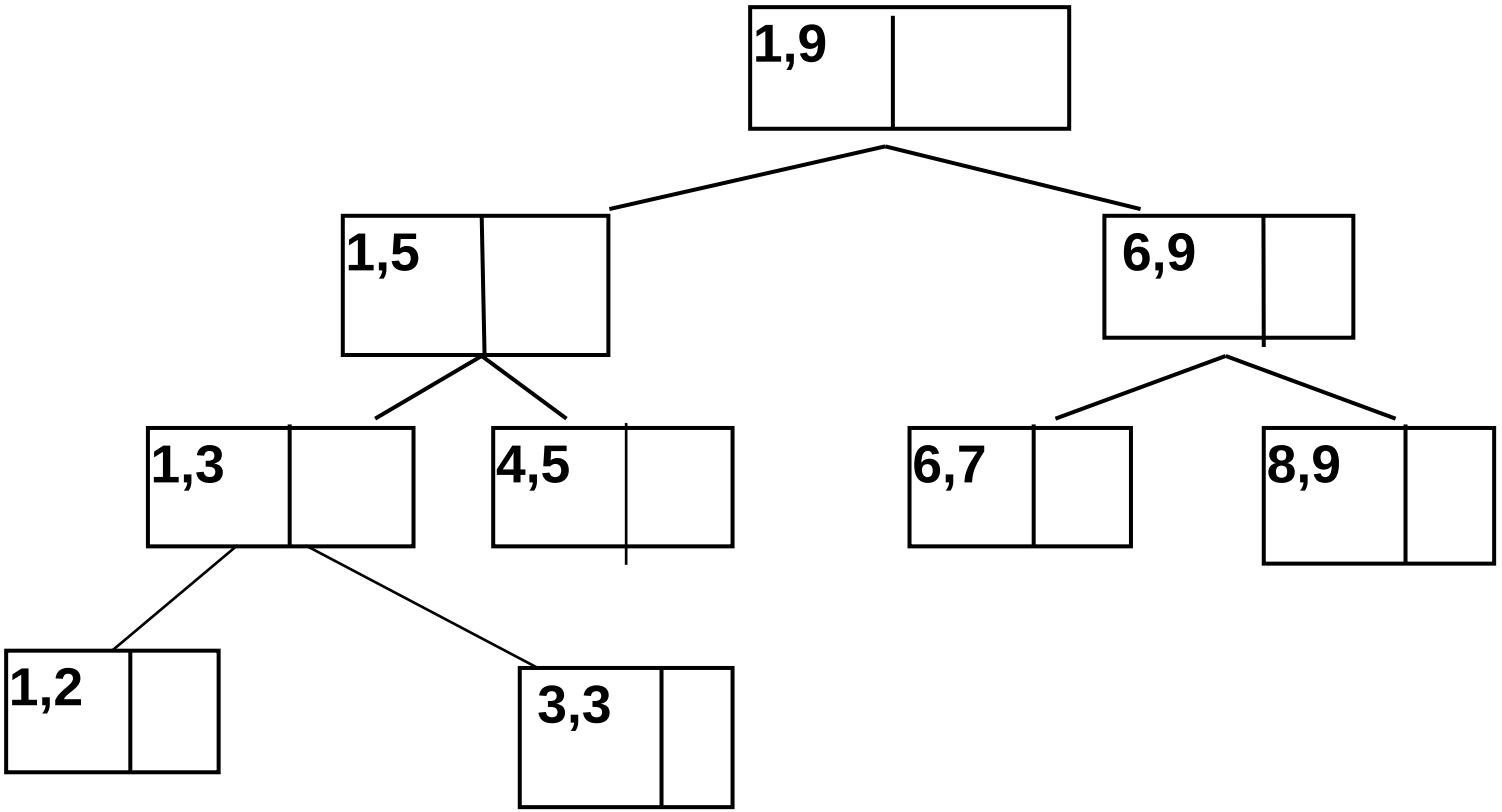
Contoh :

Tentukan elemen MaxMin suatu array A yang terdiri dari 9 bilangan :

A[1] = 22,	A[4] = -8,	A[7] = 17
A[2] = 13,	A[5] = 15,	A[8] = 31
A[3] = -5,	A[6] = 60,	A[9] = 47

Teknik D AND C (Lanjutan)

Penyelesaian **Teknik D and C**
A= {22,13, -5, -8, 15, 60, 17, 31, 47}

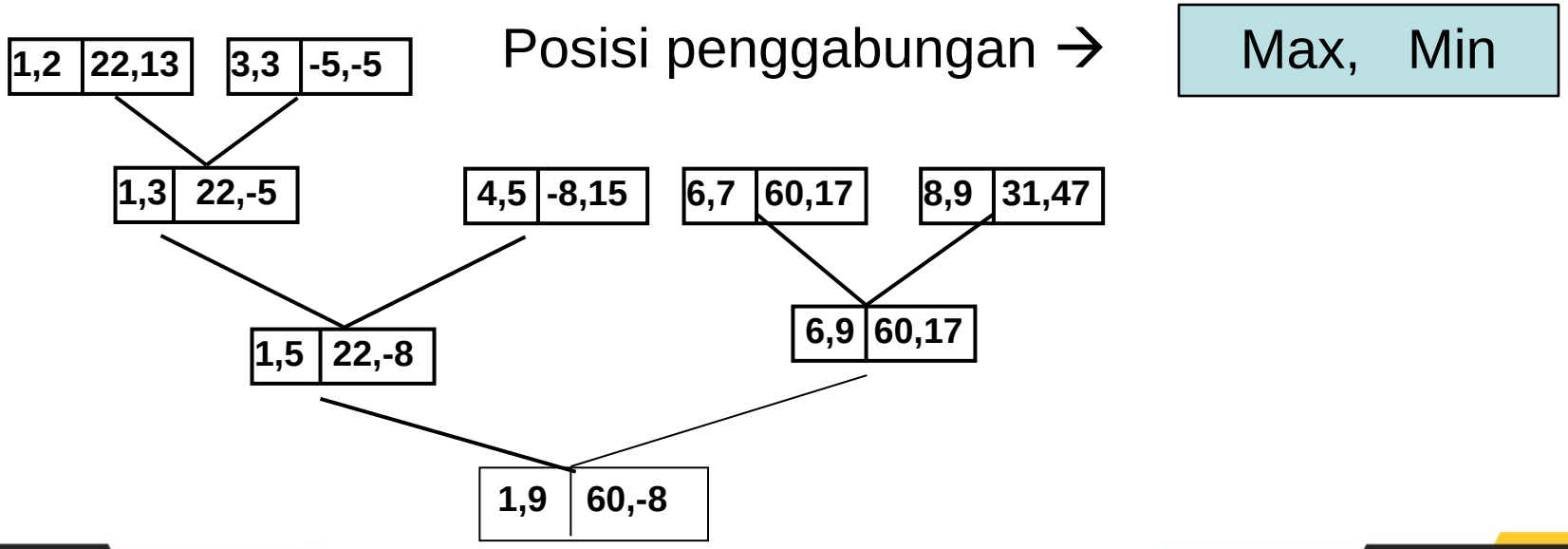


Teknik D AND C (Lanjutan)

Penyelesaian Teknik D AND C

Lalu *Proses tree call* dr setiap elemen yang ditunjuk pada bagan tree tersebut diatas. Dengan cara, membalik terlebih dahulu posisi tree dari bawah ke atas. Lalu mengisinya dengan elemen-elemnnya sesuai dengan bagan tree. Perhatikan bagan *tree call* ini:

A = { 22, 13, -5, -8, 15, 60, 17, 31, 47 }



Tugas Kelompok

- Ada sebuah data dengan urutan sebagai berikut:
  25, 20, 15, 3, 7, 2, 1
  Bagaimana hasil pengurutan data diatas dengan metode D AND C Merge Sort, dan Quick Sort?

- Ada sebuah deretan angka 3 6 9 13 16 26 38 58 menggunakan Binary Search tentukan untuk pencarian data 13, 16 dan 10?

- Soal dikerjakan secara berkelompok