Decrease and Conquer

Bahan Kuliah IF2211 Strategi Algoritma

Oleh: Rinaldi Munir

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika ITB

- *Decrease and conquer*: metode desain algoritma dengan mereduksi persoalan menjadi beberapa sub-persoalan yang lebih kecil, tetapi selanjutnya hanya memproses satu sub-persoalan saja.

- Berbeda dengan *divide and conquer* yang memproses *semua* sub-persoalan dan menggabung semua solusi setiap sub-persoalan.

- *Decrease and conquer* terdiri dari dua tahapan:

  1. *Decrease*: mereduksi persoalan menjadi beberapa persoalan yang lebih kecil (biasanya dua sub-persoalan).

  2. *Conquer*: memproses satu sub-persoalan secara rekursif.

- Tidak ada tahap *combine* dalam *decrease and conquer.*

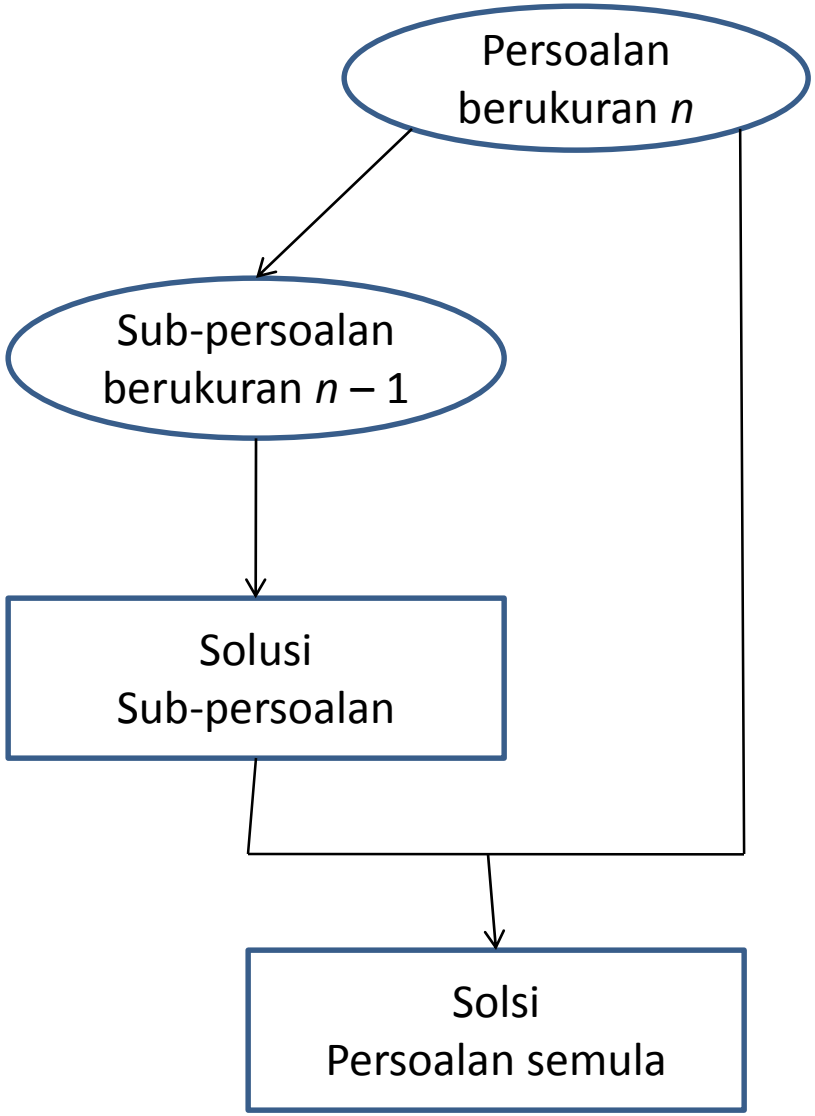
- Tiga varian *decrease and conquer*:

  1. ***Decrease by a constant***: ukuran instans persoalan direduksi sebesar konstanta yang sama setiap iterasi algoritma. Biasanya konstanta = 1.

  2. ***Decrease by a constant factor***: ukuran instans persoalan direduksi sebesar faktor konstanta yang sama setiap iterasi algoritma. Biasanya faktor konstanta = 2.

  3. ***Decrease by a variable size***: ukuran instans persoalan direduksi bervariasi pada setiap iterasi algoritma.

Decrease by a Constant



- **Contoh 1**: Persoalan perpangkatan a^n

  Dengan metode *decrease and conquer*:

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

  Kompleksitas waktu (berdasarkan jumlah operasi kali):

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

  Bila diselesaikan:

$$T(n) = T(n-1) + 1 = \ldots. = O(n)$$

  sama seperti algoritma *brute-force*.

function exp(*a* : **real**; *n* : **integer**) → **real**

Deklarasi

k : **integer**

Algoritma:

if *n* = 0 **then**

return 1

else

return exp(*a*, *n* – 1) * *a*

endif

Contoh 2: *Selection Sort*

```
procedure SelectionSort(input/output A : TabelInt, input i,j: integer)

{ Mengurutkan tabel A[i..j] dengan algoritma Selection Sort.
  Masukan: Tabel A[i..j] yang sudah terdefinisi elemen-elemennya.
  Keluaran: Tabel A[i..j] yang terurut menaik.
}

Algoritma:
   if  i < j then          { Ukuran(A) > 1 }
     Bagi(A, i, j)
     SelectionSort(A, i+1, j)
   endif
```

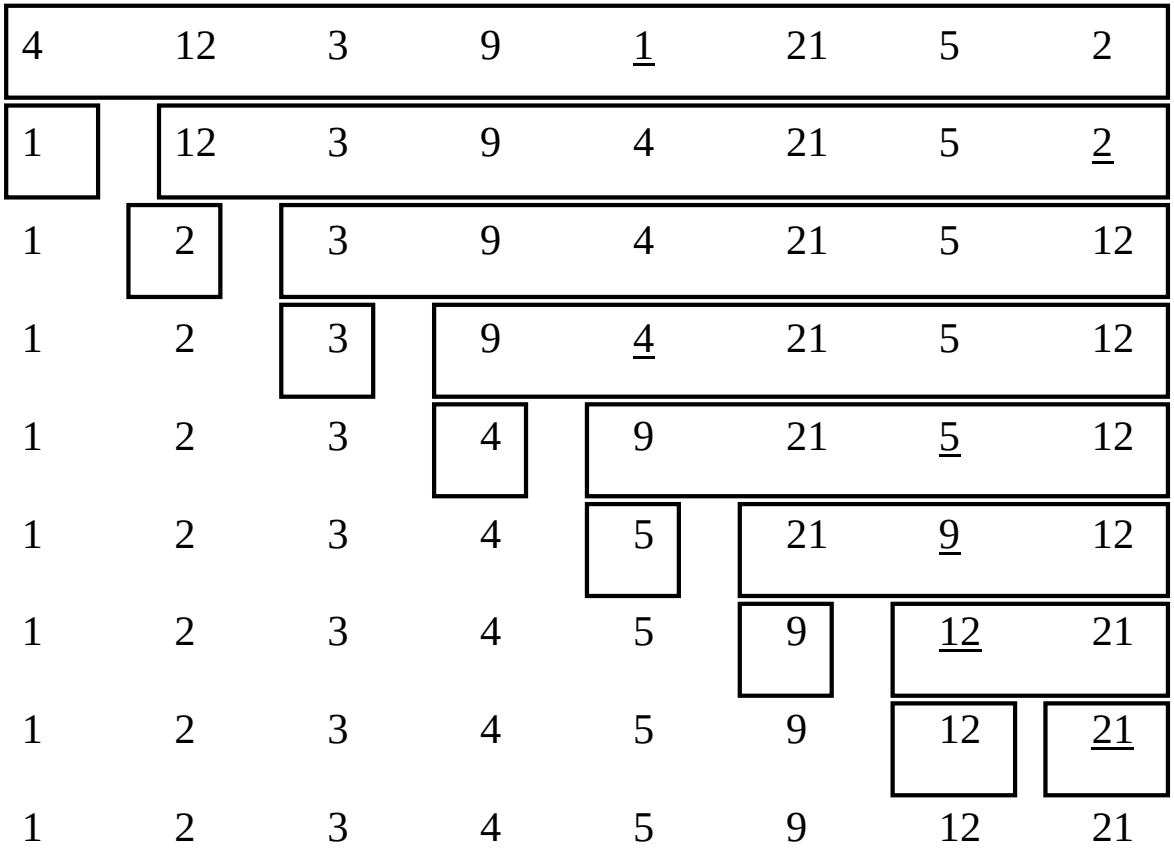
```
procedure Bagi(input/output A : TabInt, input i,j: integer)

{ Mencari elemen terkecil di dalam tabel A[i..j], dan menempatkan
elemen terkecil sebagai elemen pertama tabel.

  Masukan: A[i..j]
  Keluaran: A[i..j] dengan Ai adalah elemen terkecil.
}
Deklarasi
  idxmin, k, temp : integer

Algoritma:
  idxmin←i
  for k←i+1 to jdo
    if Ak < Aidxmin then
      idxmin←k
    endif
  endfor

 { pertukarkan Ai dengan Aidxmin }
 temp←Ai
 Ai←Aidxmin
 Aidxmin←temp
```

- **Contoh** *Selection sort*

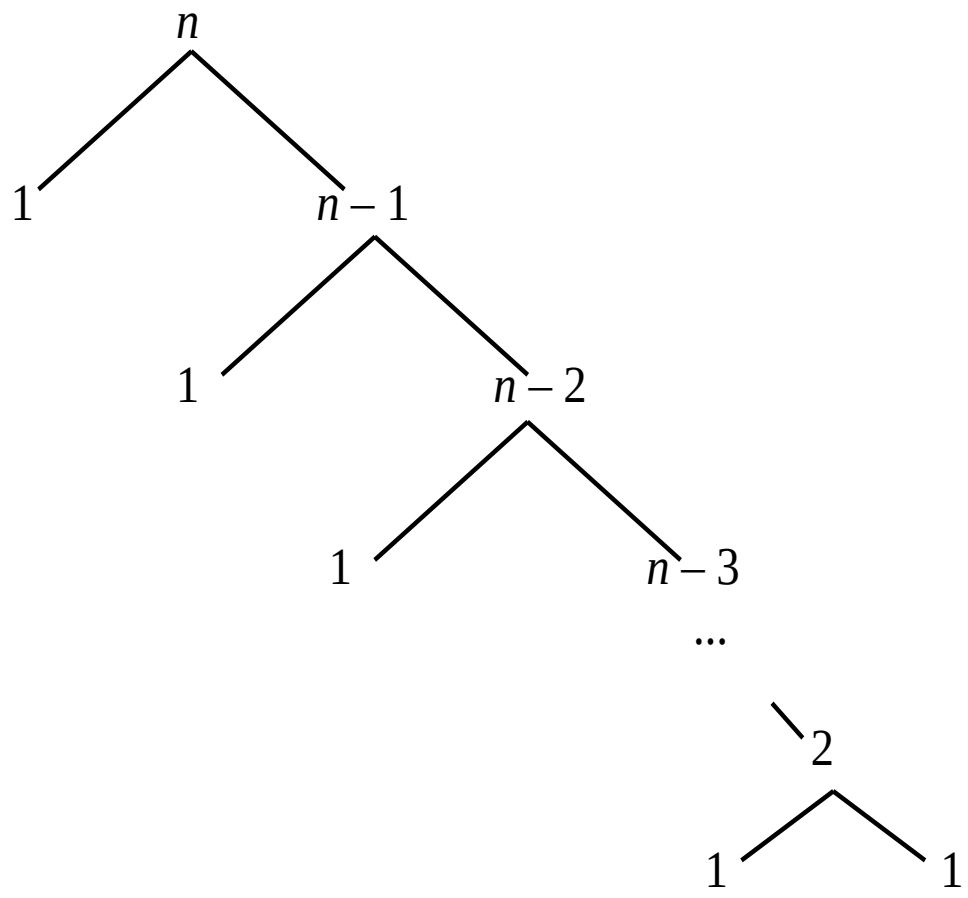
  Misalkan tabel *a* berisi elemen-elemen berikut:

  4 12 3 9 1 21 5 2

  Langkah-langkah pengurutan dengan *Selection Sort*:



Pohon pembagian larik:



- Kompleksitas waktu algoritma *Selection Sort:*

  $T(n)$ = waktu pembagian + waktu pemanggilan rekurens *Selection Sort* untuk bagian tabel kanan yang berukuran n elemen.

$$T(n) = \begin{cases} a & , n = 1 \\ T(n-1) + cn & , n > 1 \end{cases}$$

Persamaan pada bagian rekurensi bila diselesaikan menghasilkan $T(n) = O(n^2)$.

Contoh 3: *Insertion Sort*

```
procedure InsertionSort(input/output A : TabelInt,
                        input i, j : integer)
{ Mengurutkan tabel A[i..j] dengan algoritma Insertion Sort.
  Masukan: Tabel A dengan n elemen
  Keluaran: Tabel A yang terurut
}
Deklarasi:
   k : integer

Algoritma:
   if i < j then          { Ukuran(A)> 1}
     k←i
     InsertionSort(A, k+1, j)
     Merge(A, i, k, j)
   endif
```

Contoh 4.4. Misalkan tabel *A* berisi elemen-elemen berikut:

4 12 23 9 21 1 5 2

DIVIDE, CONQUER, dan SOLVE::

4 12 3 9 1 21 5 2

4 12 3 9 1 21 5 2

4 12 3 9 1 21 5 2

4 12 3 9 1 21 5 2

4 12 3 9 1 21 5 2

4 12 3 9 1 21 5 2

4 12 3 9 1 21 5 2

4 12 3 9 1 21 5 2

4 12 3 9 1 21 5 2

MERGE: 4 12 3 9 1 21 2 5

4 12 3 9 1 2 5 21

4 12 3 9 1 2 5 21

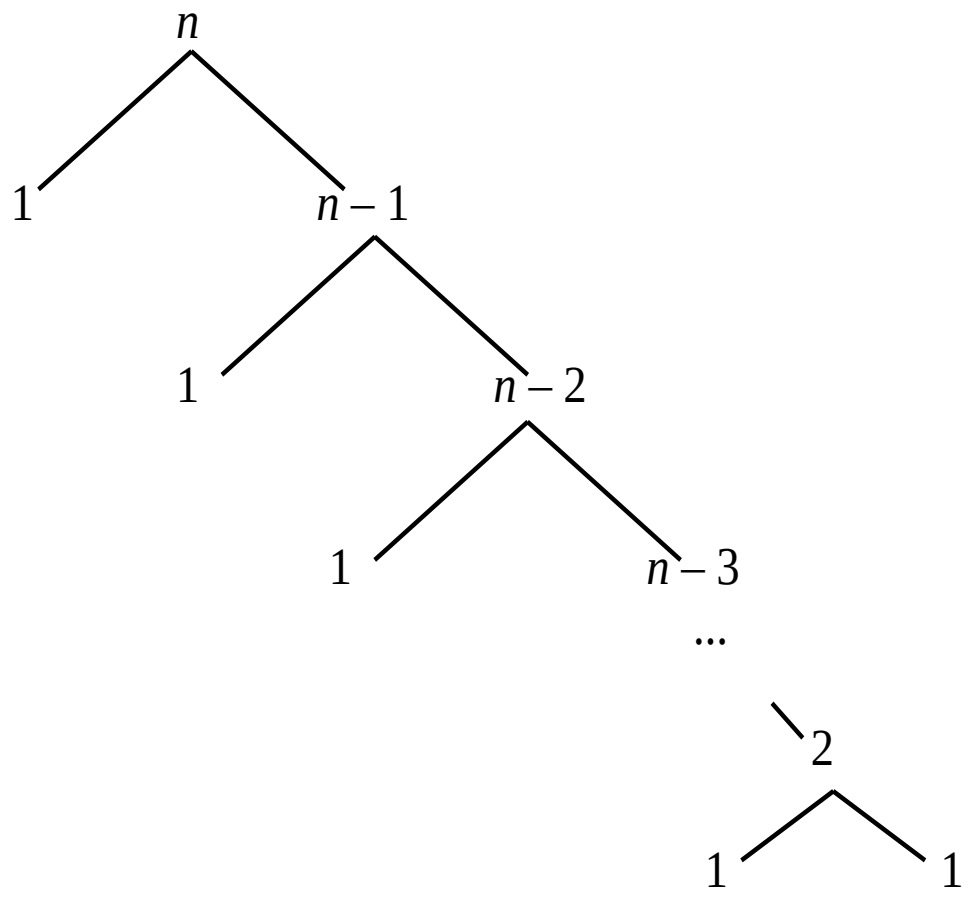
4 12 3 1 2 5 9 21

4 12 1 2 3 5 9 21

4 1 2 3 5 9 12 21

1 2 3 4 5 9 12 21

Pohon pembagian larik:



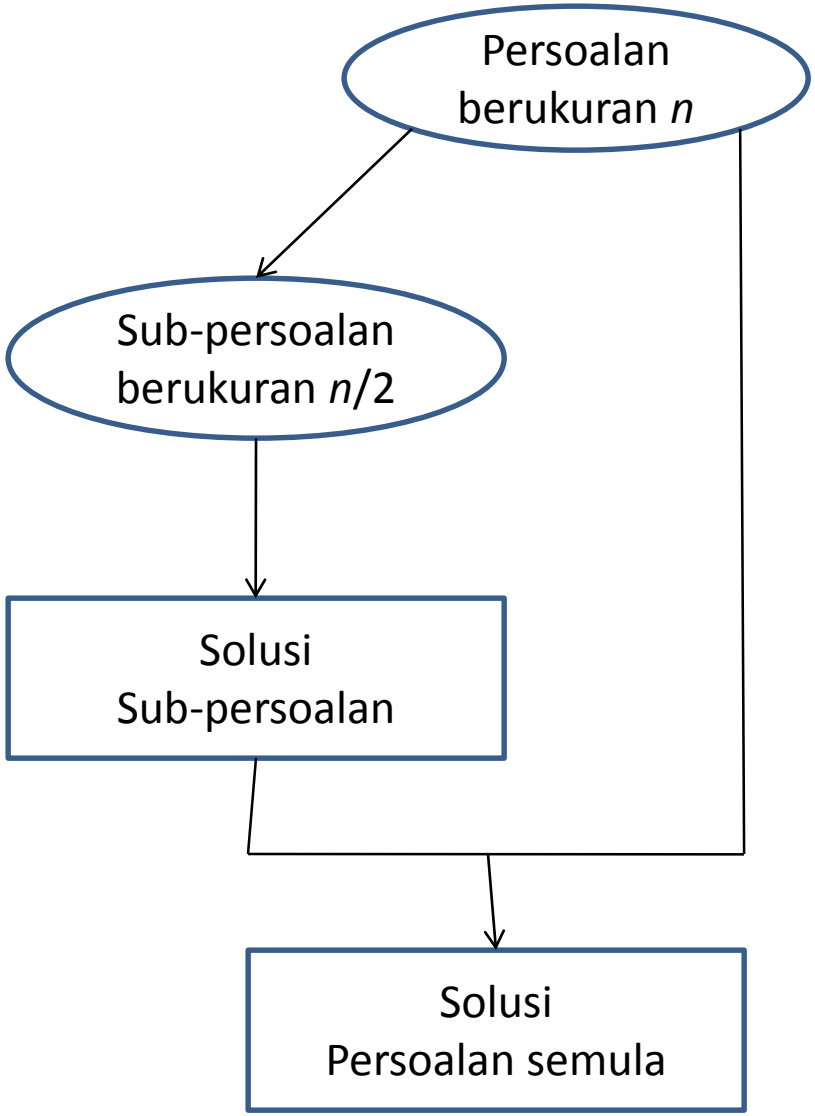
Kompleksitas waktu algoritma *Insertion Sort*:

$$T(n) = \begin{cases} a & ,n = 1 \\ T(n-1) + cn & ,n > 1 \end{cases}$$

Penyelesaian:

$$\begin{aligned}
T(n) &= cn + T(n-1) \\
&= cn + \{\ c \cdot (n-1) + T(n-2)\ \} \\
&= cn + c(n-1) + \{\ c \cdot (n-2) + T(n-3)\ \} \\
&= cn + \mathrm{c} \cdot (n-1) + c \cdot (n-2) + \{c(n-3) + T(n-4)\ \} \\
&= \ldots \\
&= cn + c \cdot (n-1) + c(n-2) + c(n-3) + \ldots + c2 + T(1) \\
&= c\{\ n + (n-1) + (n-2) + (n-3) + \ldots + 2\ \} + a \\
&= \mathrm{c}\{\ (n-1)(n+2)/2\ \} + a \\
&= cn^2/2 + cn/2\ + (a - c\ ) \\
&= O(n^2)
\end{aligned}$$

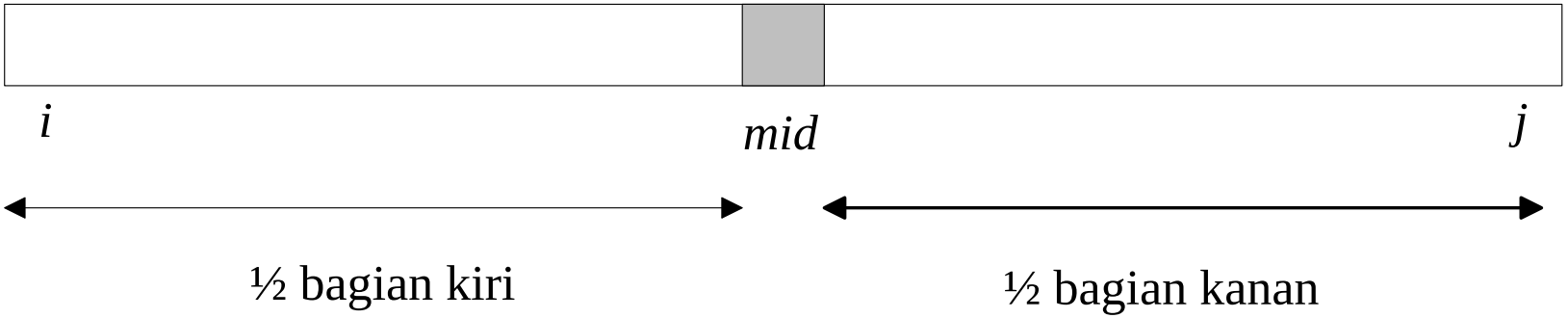
Decrease by a Constant Factor



- **Contoh 3**: *Binary search*

  Kondisi awal: larik *A* sudah terurut menaik

  K adalah nilai yang dicari



Jika elemen tengah (mid) ≠ *k*, maka pencarian dilakukan hanya pada setengah bagian larik (kiri atau kanan)

Ukuran persoalan selalu berkurang sebesar setengah ukuran semula. Hanya setengah bagain yang diproses, setengah bagian lagi tidak.

```
procedure bin_search(input A : ArrayOfInteger;
                     input i, j : integer; input K : integer; output idx : integer)
Deklarasi
    mid : integer

Algoritma:
    if i > j then   { ukuran larik sudah 0}
        idx ← -1    { k tidak ditemukan }
    else
        mid ← (i + j)/2
        if A(mid) = K then   { k ditemukan }
            idx ← mid        { indeks elemen larik yang bernilai = K }
        else
            if  A(mid) > K  then
                bin_search(A, i, mid – 1, K, idx)
            else
                bin_search(A, mid + 1, j, K, idx)
            endif
        endif
endif
```

- Jumlah operasi perbandingan:

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

- Relasi rekursif tsb diselesaikan sbb:

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

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

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

= ... = j + $T(n/2^j)$

Asumsi: $n = 2^j \rightarrow j = {}^2\log n$

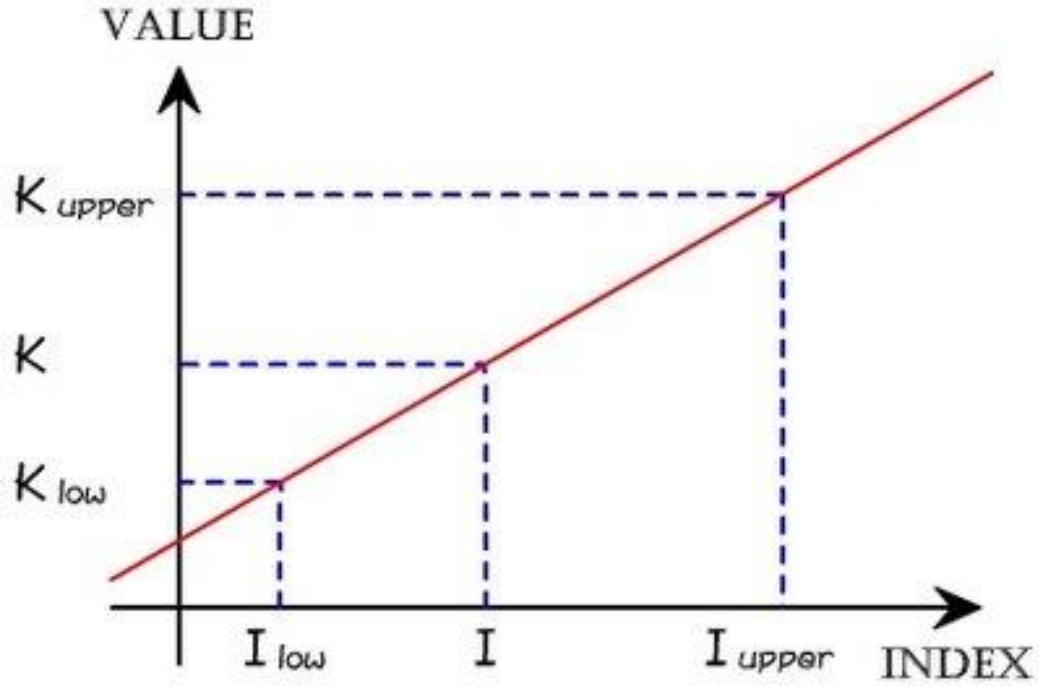
T(n) = $^2\log n$ + T(1) = $^2\log n$ + (1 + T(0)) = 1 + $^2\log n$ = O($^2\log n$)

- **Contoh 4:** *Interpolation Search*

  - Analog dengan pencarian data di dalam kamus dengan cara perkiraan letak.

  - Kondisi awal: larik *A* sudah terurut menaik

    K adalah nilai yang dicari



$$\frac{K - K_{low}}{K_{upper} - K_{low}} = \frac{I - I_{low}}{I_{upper} - I_{low}}$$

$$I = I_{low} + (I_{upper} - I_{low}) \times \frac{K - K_{low}}{K_{upper} - K_{low}}$$

```
procedure interpolation_search(input A : ArrayOfInteger;
                input i, j : integer; input K : integer; output idx : integer)
Deklarasi
      mid : integer

Algoritma:
    if i > j then      { ukuran larik sudah 0}
        idx ← -1       { K tidak ditemukan }
    else
        mid ← i  + (j – i) *(K – A(i))/ (A(j) – A (i))
        if A(mid) = K then   { K ditemukan }
            idx ← mid        { indeks elemen larik yang bernilai = K }
        else
            if  A(mid) > K  then
                interpolation_search(A, i, mid – 1, K, idx)
            else
                interpolation_search (A, mid + 1, j, K, idx)
            endif
        endif
endif
```

- Kompleksitas algoritma *interpolation search*:

  - Kasus terburuk: $O(n)$, untuk sembarang distribusi data

  - Kasus terbaik: $O(\log \log n)$, jika data di dalam senarai terdistribusi *uniform*

- **Contoh 5** (Mencari koin palsu). Diberikan *n* buah koin yang identik, satu diantaranya palsu. Asumsikan koin yang palsu mempunyai berat yang lebih ringan daripada koin asli. Untuk mencari yang palsu, disediakan sebuah timbangan yang teliti. Carilah koin yang palsu dengan cara penimbangan.

Algoritma *decrease and conquer*:

1. Bagi himpunan koin menjadi dua sub-himpunan, masing-masing $\lfloor n/2 \rfloor$ koin. Jika *n* ganjil, maka satu buah koin tidak dimasukkan ke dalam kedua sub-himpunan.
2. Timbang kedua sub-himpunan dengan neraca.
3. Jika beratnya sama, berarti satu koin yang tersisa adalah palsu.
4. Jika beratnya tidak sama, maka ulangi proses untuk sub-himpunan yang beratnya lebih ringan (salah satu koin di dalamnya palsu).

- Ukuran persoalan selalu berkurang dengan faktor setengah dari ukuran semula. Hanya setengah bagian yang diproses, setengah bagian yang lain tidak diproses.

- Jumlah penimbangan yang dilakukan adalah:

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

- Penyelesaian relasi rekurens T(n) mirip seperti *binary search*:

$T(n) = 1 + T(\lfloor n/2 \rfloor) = \ldots. = O(^2 \log n)$

Decrease by a Variable Size

- **Contoh 6**: Menghitung median dan *Selection Problem*.
  - *Selection problem*: mencari elemen terkecil ke-*k* di dalam sebuah senarai beranggotan *n* elemen.
  - Jika k = 1 → elemen paling kecil (minimum)
  - Jika k = n → elemen paling besar (maksimum)
  - Jika k = $\lceil n/2 \rceil$ → elemen median

  Bagaimana mencari median dari senarai yang tidak terurut namun tidak perlu mengurutkan senarai terlebih dahulu?

Algoritmanya:

1. Lakukan partisi pada senarai seperti proses partisi pada algoritma Quick Sort (varian 2). Partisi menghasilkan setengah elemen senarai lebih kecil atau sama dengan *pivot p* dan setengah bagian lagi lebih besar dari *pivot p*.

$$\underbrace{a_{i_1} \cdots a_{i_{s-1}}}_{\leq p} \; p \; \underbrace{a_{i_{s+1}} \cdots a_{i_n}}_{\geq p}$$

2. Misalkan *s* adalah posisi pem-partisian.

   Jika s = $\lceil n/2 \rceil$, maka pivot *p* adalah nilai median yang dicari

   Jika s > $\lceil n/2 \rceil$, maka median terdapat pada setengah bagian kiri

   Jika s < $\lceil n/2 \rceil$, maka median terdapat pada setengah bagian kanan

- **Contoh**: Temukan median dari 4, 1, 10, 9, 7, 12, 8, 2, 15.

  pada contoh ini, $k = \lceil 9/2 \rceil = 5$, sehingga persoalannya adalah mencari elemen terkecil ke-5 di dalam senarai.

  Partisi senarai dengan memilih elemen pertama sebagai pivot:

  4 1 10 9 7 12 8 2 15

  Hasil partisi:

  2 1 **4** 9 7 12 8 10 15

  Karena $s = 3 < 5$, kita memproses setengah bagian kanan:

  9 7 12 8 10 15

  8 7 **9** 12 10 15

  Karena $s = 6 > 5$, kita memproses setengah bagian kiri:

  8 7

  7 **8**

  Sekarang $s = k = 5$ → stop. Jadi median = 8

- Kompleksitas algoritma:

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

- Solusi dari relasi rekurens tersebut adalah (dengan menggunakan Teorema Master):

T(n) = T(n/2) + cn = … = O(n)