EVALUASI RUNNING TIME

ANALISIS DAN DESAIN ALGORITMA 1

ATURAN ESTIMASI

- **Running time proporsional dengan term yang paling signifikan pada T (n)**
  - Pada saat ukuran input n menjadi besar, term yang paling signifikan adalah term yang memiliki pangkat tertinggi dari n.
  - Term ini naik lebih cepat dibandingkan dengan term-term lain (yang signifikansinya semakin berkurang)
- **Konstanta bergantung pada *compiler*, bahasa pemrograman, komputer, dll.**
  - Konstanta diabaikan ketika menganalisis algoritma.
  - Konstanta akan mengecil jika digunakan hardware yang lebih cepat
    - Tetapi ini tidak akan mempengaruhi perilaku algoritma untuk ukuran input yang besar .

OPERASI ELEMENTER DAN DATA INPUT

- **Operasi elementer:**
  - Operasi aritmetika (+; −; *; /; %)
  - Operasi relasional (==; !=; >; <; ≥; ≤)
  - Operasi Boolean (AND; OR; XOR; NOT)
  - Operasi percabangan
  - Return
- **Penyederhanaan asumsi: semua pernyataan/ekspresi dasar memerlukan waktu yang sama untuk dieksekusi.**
- **Running time algoritma: jumlah operasi elementer yang dominan digunakan (operasi khas).**
- **Input untuk domain permasalahan (arti dari n):**
  - Sorting: n item
  - Graf/ lintasan: n verteks/ sisi
  - Image processing: n pixel (2D image) atau voxel (3D image)
  - Text processing: n karakter (panjang string yang diinputkan)

MENGESTIMASI RUNNING TIME: LOOP

- **Loop tunggal memiliki kompleksitas linier yaitu λ . $T_{body\ of\ a\ loop}$ dengan λ jumlah eksekusi dari loop.**
- **Loop bersarang memiliki running time polinomial $T(n) = cn^k$**
  - Dalam hal ini jumlah operasi pada loop paling dalam berupa suatu konstanta (k adalah level tertinggi dari persarangan, c adalah konstanta).
- **Tiga nilai pertama k memiliki nama khusus :**
  - Waktu linier untuk k = 1 (loop tunggal);
  - Waktu kuadratik untuk k = 2 (2 loop bersarang), dan
  - Waktu kubik untuk k = 3 (3 loop bersarang).

MENGESTIMASI RUNNING TIME: KONDISIONAL

- **Perhitungan running time untuk pernyataan kondisional / switch lebih kompleks.**

  if {condition} then {const time T1} else {const time T2}

- **Kita harus mengetahu frekuensi (probabilitas) dari percabangan $f_{condition=true}$**
  - $f_{condition=false} = 1 - f_{condition=true}$
- **Dengan demikian, running time dari algoritma adalah: $T = f_{true}T_1 + (1 - f_{true})\, T_2 \leq \max\{T_1, T_2\}$**

MENGESTIMASI RUNNING TIME: FUNGSI

- **Running time pemanggilan fungsi:**

  $$\mathbf{T_{function} = \Sigma\ T_{statements\ in\ function}}$$

  - Secara detil: $T_{function} = \Sigma_i\ T_{statement\ i}$

    ... x.myMethod(5, ...);

    ...

    public void myMethod(int a, ...) {

    statement 1,2,...,M }

- **Komposisi fungsi: T(f(g(n))) = T(g(n)) + T(f(n))**
  - Secara detil:
    - Komputasi dari x = g(n) ⟶ T (g(n))
    - Komputasi dari y = f(n) ⟶ T(f(n))
    - T(f(g(n))) = T(g(n)) + T(f(n))

CONTOH

- **Kasus paling sederhana untuk loop : T(i) tidak bergantung pada i**
  - Contoh :

    for (i = 1; i <= n; i++) // waktu: c_1 , n+1 kali

    {...sejumlah konstan operasi} // waktu: c_2 , n kali
  - Setiap T(i) memerlukan waktu c_2 → $T = (n+1)c_1 + nc_2 = O(n)$
- **Menggunakan while loop:**

```
i = 1                  // → c1
while i ≤ m {          // → (m+1) c2
        ....           // → mc3
        i = i + 1      // → mc4
```

  - Waktu total : $T \leq (c_2 + c_3 + c_4)\, m + c_1 + c_2$.

CONTOH: MENGESTIMASI WAKTU UNTUK MENJUMLAHKAN M+1 SUBARRAY

- **m subarray dengan ukuran masing-masing m = n/2:**
  - Abaikan inisialisasi data
  - Lakukan penjumlahan secara "Brute-force" dengan 2 loop bersarang:

```
for (j=0; j<=m; j++)
  s[ j ] = 0;   {  //m+1kali
  for ( k = 0; k < m; k++ )
      s[ j ] += a[ k ];   { // m kali
   }
}
```

- **T (n) = m(m + 1) = n/2 (n/2 + 1) = $0.25n^2 + 0.5n$**

- **Term kuadratik vs linier dalam penjumlahan m + 1 subarray masing-masing berukuran m = n/2 :**

  $T(n) = 0.25n^2 + 0.5n$

n	$T(n)$	$0.25n^2$	$0.5n$	
			value	% of T(n)
10	30	25	5	16.7
50	650	625	25	3.8
100	2550	2500	50	≈2.0
500	62750	62500	250	≈0.4
1000	250500	250000	500	≈0.2
5000	6252500	6250000	2500	≈0.04

CONTOH: MENGANALISIS ALGORITMA INSERTION-SORT

- **Algoritma insertion sort :**
  - INSERTION-SORT(A)

1. for $j \leftarrow 2$ to *length*[A] do /*c_1 (n kali) */
2. *key* $\leftarrow A[j]$ /*c_2 (n-1 kali) */
3. /* Insert $A[j]$ into the sorted sequence $A[1 \ldots j-1]$*/ /*0 (n-1 kali) */
4. $i \leftarrow j-1$ /*c_4 (n-1 kali) */
5. while $i > 0$ and $A[i] >$ *key do* /*c_5 ($\sum_{j=2}^{n} t_j$ kali) */
6. $A[i+1] \leftarrow A[i]$ /*c_6 ($\sum_{j=2}^{n} (t_j-1)$ kali) */
7. $i \leftarrow i-1$ /*c_7 ($\sum_{j=2}^{n} (t_j-1)$ kali) */
8. $A[i+1] \leftarrow$ *key* /*c_8 (n-1 kali) */

- $\boldsymbol{T(n)=c_1n+c_2(n-1)+c_4(n-1)+c_5\sum_{j=2}^{n} t_j+c_6\sum_{j=2}^{n} (t_j-1) +c_7 \sum_{j=2}^{n} (t_j-1)+c_8(n-1)}$

- **Best case: kasus ketika jumlah perhitungan minimal.**
- **Best-case untuk insertion sort: array sudah terurut.**
  - $t_2 = t_3 \ldots = t_n = 1$.
  - $T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1)$
  - $= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8)$.
  - Merupakan fungsi **linier** dari n.

- **Worst case: waktu terlama yang diperlukan oleh sembarang instance dengan ukuran n.**
- **Worst-case pada insertion sort: array terurut secara terbalik.**
  - $t_2 = 2, t_3 = 3, \ldots, t_n = n$
  - $T(n)=c_1 n+c_2 (n-1)+c_4 (n-1)+c_5 (n(n+1)/2 -1) + c_6 (n(n-1)/2) + c_7(n(n-1)/2 + c_8 (n-1)$

    $=(c_5/2 +c_6/2 +c_7/2)n^2+(c_1+c_2 +c_4 +c_5/2 -c_6/2 -c_7/2 +c_8)n - (c_2 +c_4 +c_5 +c_8)$
  - Merupakan fungsi **kuadrat** dari *n*.
- **Catatan: $\sum_{k=1}^{n} k = 1 + 2 + 3+ \ldots + n = n(n+1)/2 \rightarrow$ deret aritmetik**

LATIHAN

- **Untuk nested loop berikut, berupa fungsi apakah running time-nya?**

```
m = 1;
for ( j = 1; j <= n; j++ ) {
      if ( j == m ) {
          m = m * (n – 1);
          for ( i = 1; i <= n; i++ ) {
             ...sejumlah konstan operasi elementer
          } // end for
      } // end if
 } // end for
```

- **Inner loop dieksekusi hanya 2 kali, untuk j = 1 dan j = n − 1**
- **Waktu totalnya: T(n) = 2n → running time linier.**

LATIHAN

- **Tentukan running time yang diperlukan oleh algoritma berikut, jika diasumsikan bahwa operasi khas dari masing-masing fungsi adalah perkalian, dan running time funct1 adalah i^2, sedangkan running time dari funct2 adalah j^2.**

```
i = 5
j = 10
while i < j do
        i = i + 1
        j = j – 1
        funct1(i)
        funct2(j)
```