Zložitosť a Insert Sort

Bc. Katarína Olejková

KATEDRA INFORMATIKY

UNIVERZITA PALACKÉHO V OLOMOUCI

Časová zložitosť

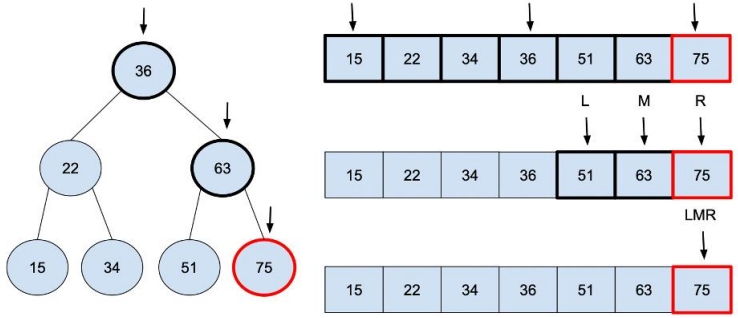
- Závisí od veľkosti vstupu
- Funkcia, ktorá veľkosti vstupu priradí trvanie výpočtu

- Veľkosť vstupu rozumieme, ako:
  - Hodnota čísla
  - Dĺžka textu
  - Počet prvkov v poli
  - Počet cifier
  - Počet uzlov v grafe
  - Atď..

- Trvanie výpočtu rozumieme, ako počet výpočetných krokov algoritmu (inštrukcií)
  - Trvanie výpočtu **nemeriame** skutočným časom (napr. počtom sekúnd) lebo ten je závislý od HW počítača

Časová zložitosť

- Často sa vyskytujúce časové zložitosti T(n):
  - Konštantná – vrátenie prvého prvku v poli
  - Logaritmická – binárne vyhľadávanie
  - Lineárna - nájdenie maxima v poli
  - Logaritmicko lineárna – merge-sort, quick-sort..
  - Kvadratická – dvojitý vnorený cyklus (insert-sort, select-sort)
  - Kubická – trojitý vnorený cyklus
  - Exponenciálna – rekurzívny výpočet Fibonacciho postupnosti
  - Faktoriál – výpočet permutácií poľa



Príklad 1 - algoritmus JeKladne(n) – určite jeho časovú zložitosť v najhoršom prípade

```
1. JeKladne(n)
2.     if n > 0
3.         print(“áno“)
4.     else
5.         print(“nie“)
```

Príklad 2 – algoritmus SudeNeboLiche(n) – určite jeho časovú zložitosť v najhoršom prípade

```
1. SudeNeboLiche(n)
2.     if n mod 2 = 0
3.         print(“sudé“)
4.     else
5.         print(“liché“)
```

Príklad 3 – algoritmus Signum(n) – určite jeho časovú zložitosť v najhoršom prípade

```
1. Signum(n)
2.     if n > 0
3.         return 1
4.     else if n = 0
5.         return 0
6.     else
7.         return - 1
```

Príklad 4 – algoritmus Sucet(n) – určite jeho časovú zložitosť v najhoršom prípade

1. Sucet(n)
2. sum ← 0
3. **for** i ← 1 **to** n
4. sum ← sum + i
5. return sum

Príklad 5 – algoritmus Nasobilka(n) – určite jeho časovú zložitosť v najhoršom prípade

```
1. Nasobilka(n)
2.     for i ← 1 to n
3.         for j ← 1 to n
4.             soucin ← i * j
5.             print(i ‘*’ j ‘=‘ soucin)
```

Príklad 6 – algoritmus SucetPole(A[0..n-1], n) – určite jeho časovú zložitosť v najhoršom prípade

```
1. SucetPole(A[0..n-1], n)
2.     sum ← 0
3.     for i ← 0 to n - 1
4.         sum ← sum + A[i]
5.     return sum
```

Príklad 7 – algoritmus AritmetickyPrumer(A[0..n-1], n) – určite jeho časovú zložitosť v najhoršom prípade

```
1. AritmetickyPrumer(A[0..n-1], n)
2.     sum ← 0
3.     for i ← 0 to n - 1
4.         sum ← sum + A[i]
5.     prumer ← sum / n
6.     return prumer
```

Insertion Sort

- Triedime zľava doprava
- Každý prvok porovnáme s prvkami na jeho ľavej strane
  - Opakujeme - Ak je prvok menší ako prvok na ľavej strane, vymeníme ich pozície
- Ak sa už prvok nedá vymeniť, tak je na správnej pozícii

Insertion Sort

```
1. Insert-Sort(A[0..n-1], n)
2.     for j ← 1 to n - 1
3.         t ← A[j]
4.         i ← j - 1
5.         while i >= 0 and A[i] > t
6.             A[i + 1] ← A[i]
7.             i ← i – 1
8.         A[i + 1] ← t
```

Insertion Sort - časová zložitosť

- Veľkosť vstupu – veľkosť vstupného poľa

- V najhoršom prípade
  - Vstupné pole je setřídené sestupně (od najväčšieho prvku)
  - T(n) = n^2 kvadratická

- V priemernom prípade
  - T(n) = n^2 kvadratická

- V najlepšom prípade
  - Vstupné pole je setřídené vzestupně (od najmenšieho prvku)
  - T(n) = n lineárna

Insertion Sort

- Animácia:
  - https://liveexample.pearsoncmg.com/dsanimation/InsertionSortNeweBook.html

Úkol

- Pomocou šablóny InsertSort-Sablona.c/.py na GitHube naprogramujte v C alebo Pythone Insertion Sort
  - (pomôžte si pseudokódom nemusíte ho písať zpamäti)