Pseudokódy – Časová zložitosť v najhoršom prípade

Bc. Katarína Olejková

KATEDRA INFORMATIKY

UNIVERZITA PALACKÉHO V OLOMOUCI

Príklad 1 - JeKladne(n)

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

1x porovnanie
1x print – buď “áno” alebo “nie”
(podľa toho ako dopadne podmienka if)

1 + 1 = 2

T(n) = 2
Konštantná čas. slož.

Príklad 2 – SudeNeboLiche(n)

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

1x porovnanie
1x operácia modulo
1x print – buď “sudé” alebo “liché”
(podľa toho ako dopadne podmienka if)

1 + 1 + 1 = 3

T(n) = 3
Konštantná čas. slož.

Príklad 3 – Signum(n)

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

2x porovnanie
1x return – buď 0 alebo -1
(podľa toho ako dopadne podmienka else if)

2 + 1 = 3

T(n) = 3
Konštantná čas. slož.

Pozn. ak by na vstupe bolo kladné číslo a platila by podmienka if n > 0 tak by šlo o **nejlepší prípad** lebo by sa vykonali iba dve operácie – porovnanie a return 1 (T(n) = 2)

My ale riešime časovú zložitosť v najhoršom prípade

V oboch prípadoch ale môžeme povedať, že sa jedná o konštantnú časovú zložitosť

Príklad 4 – Sucet(n)

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

n-krát

1x priradenie
1x return sum

1 + 1 = 2

n-krát priradenie
n-krát priradenie
n-krát sčítanie

n + n + n = 3n

T(n) = 3n + 2
Lineárna čas. slož.

Príklad 5 – Nasobilka(n)

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

n-krát (for j)

n-krát (for i)

n-krát priradenie = n

n-krát n-krát priradenie
n-krát n-krát priradenie
n-krát n-krát sučin
n-krát n-krát print

(n * n) + (n * n) + (n * n) + (n * n) = $4n^2$

T(n) = $4n^2$ + n
Kvadratická čas. slož.

Príklad 6 – SucetPole(A[0..n-1], n)

```
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
```

n-krát

1x priradenie
1x return sum

1 + 1 = 2

n-krát priradenie
n-krát priradenie
n-krát sčítanie

n + n + n = 3n

T(n) = 3n + 2
Lineárna čas. slož.

Príklad 7 – AritmetickyPrumer(A[0..n-1], n)

```
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
```

n-krát

2x priradenie
1x delenie
1x return prumer

2 + 1 + 1 = 4

n-krát priradenie
n-krát priradenie
n-krát sčítanie

n + n + n = 3n

T(n) = 3n + 4
Lineárna čas. slož.

Dobrovolný 1 – Znamka(n)

```
1.  Znamka(n)
2.      if n ≥ 90 and n ≤ 100
3.          print('A')
4.      else if n ≥ 80 and n ≤ 89
5.          print('B')
6.      else if n ≥ 70 and n ≤ 79
7.          print('C')
8.      else if n ≥ 60 and n ≤ 69
9.          print('D')
10.     else if n ≥ 50 and n ≤ 59
11.         print('E')
12.     else
13.         print('F')
```

10x porovnanie
5x logicke and
1x print – buď 'E' alebo 'F'

10 + 5 + 1 = 16

(podľa toho ako dopadne podmienka else if n ≥ 50 and n ≤ 59)

T(n) = 16

Konštantná čas. slož.

Dobrovolný 2 – PocetLichychASudych(A[0..n-1], n)

```
1.  PocetLichychASudych(A[0..n-1], n)
2.      liche ← 0
3.      sude ← 0
4.      for i ← 0 to n - 1
5.          if A[i] mod 2 = 0
6.              sude ← sude + 1
7.          else
8.              liche ← liche + 1
9.      print(“pocet sudych: “ sude)
10.     print(“pocet lichych: “ liche)
```

n-krát (riadky 4–8)

2x priradenie
2x print
} 2 + 2 = 4

n-krát priradenie
n-krát modulo
n-krát porovnanie
} n + n + n = 3n

Podľa toho ako dopadne podmienka tak inkrementujeme buď sude alebo liche, nikdy nie oboje.
Riadok 6 a 8 sa dokopy vykoná n-krát

n-krát priradenie
n-krát sčítanie
} n + n = 2n

T(n) = 5n + 4
Lineárna čas. slož.

Dobrovolný 3 – MinMaxRozdiel(A[0..n-1], n)

```
1.  MinMaxRozdiel(A[0..n-1], n)
2.      min ← A[0]
3.      max ← A[0]
4.      for i ← 1 to n - 1
5.          if A[i] < min
6.              min ← A[i]
7.          else if A[i] > max
8.              max ← A[i]
9.      rozdiel ← max – min
10.     return rozdiel
```

n – 1 krát

3x priradenie
1x odčitanie
1x return rozdiel

3 + 1 + 1 = 5

n-1 krát priradenie
n-1 krát porovnanie v if
n-1 krát porovnanie v else if
n–1 krát priradenie do max

4*(n-1) = 4n - 4

T(n) = 4n + 1
Lineárna čas. slož.

Pozn. v najhoršom prípade if podmienka nie je splnená a kontroluje sa ďalšia podmienka v else if, ktorá bude pravdivá a prebehne priradenie do max

Keby if podmienka platila tak by sme mali o n – 1 operácii (porovnanie v else-if) menej