Ćwiczenia 4 (45 min. 20.03.2020)

Wyznacz złożoność obliczeniową podanych pętli - gdzie n oznacza liczbę danych

i.	`for(int i=0; i<2020; i++)` `  //wykonanie operacji podstawowej;`
ii.	`for(int i=0; i<n; i++)` `  //wykonanie operacji podstawowej;`

iii.	`for(int i=0; i<2019; i++)`
	`  for(int j=0; j<2020; j++)`
	`  //wykonanie operacji podstawowej;`

iv.	`for(int i=0; i<2020; i++)`
	`  for(int j=0; j<n; j++)`
	`  //wykonanie operacji podstawowej;`

v.	`for(int i=0; i<n; i++)`
	`  for(int j=0; j<n; j++)`
	`  //wykonanie operacji podstawowej;`

vi.	`for(int i=0; i<n; i++)`
	`  for(int j=0; j<i; j++)`
	`  //wykonanie operacji podstawowej;`

vii.	`for(int i=1; i<=n; i++){`
	`  int j=n;`
	`  while(j>=1){`
	`  j= j/2;`
	`  //wykonanie operacji podstawowej;`
	`  }`
	`}`

viii.	`for(int i=1; i<=n; i++){`
	`  int j=n;`
	`  while(j>=1){`
	`  j= j/3;`
	`  //wykonanie operacji podstawowej;`
	`  }`
	`}`

ix.	`for(int i=1; i<=n*n; i++){`
	`  int k=1; int l=1;`
	`  while(l<n){`
	`  k+=2; l+=k;`
	`  //wykonanie operacji podstawowej;`
	`  }`
	`}`

Rozwiązanie przykład (iii)

Łatwo zauważyć, że w tym przykładzie „operacja podstawowa" zostanie zawsze wykonana **stałą liczbę razy** (2019*2020 razy) nie zależną od ilości danych. Złożoność takiego algorytmu można określić symbolem $\Theta(1)$. W praktyce jednak częściej używa się symbolu $O(1)$ (O duże).

Rozwiązanie przykład (v)

W tym przypadku zewnętrzna pętla `for` zostanie wykonana n razy (`i=1,2,...,n`). Wewnętrzna pętla `while` (przy każdej wartości zmiennej `i`) również zostanie wykonana n razy - stąd mamy $n*n$ powtórzeń „operacji podstawowej". Złożoność takiego algorytmu można określić symbolem $\Theta(n^2)$ albo $O(n^2)$.

Rozwiązanie przykład (vii)

W tym przypadku zewnętrzna pętla `for` zostanie wykonana n razy (`i=1,2,...,n`). Wyznaczając ilość powtórzeń pętli `while` przyjmijmy, że $n = 2^k$ (tzn. ilość danych jest potęgą liczby 2).
Wtedy pętla while przy $k \geq 1$ wykona się dla

$$j = 2^k$$
$$j = 2^{k-1}$$
$$\vdots$$
$$j = 2^0 = 1$$

Wykonując pętlę while dla $j = 2^0 = 1$ wartość zmiennej j zmieni i będzie równa 0 - stąd kolejny raz pętla while nie wykona się. Mamy więc $k+1$ powtórzeń pętli while dla $j = k, k-1, \ldots 0$.

Uwzględniając zewnętrzną pętlę for may $n(k+1)$ powtórzeń.

Chcąc przedstawić wynik jako funkcję liczby n otrzymamy równanie:

$$n = 2^k$$

a stąd

$$k = \log_2 n .$$

Stąd otrzymujemy $n(k+1) = n(\log_2 n + 1) = n\log_2 n + n$ powtórzeń.

Uwaga: można pokazać, że dla $n = 2^k, 2^k + 1, 2^k + 2, \ldots 2^{k+1} - 1$ ilość powtórzeń pętli while nie zmieni się i będzie wynosiła $\lfloor \log_2 n \rfloor$.

Podając złożoność algorytmu staramy się możliwie „uprościć” funkcję. Podajemy tylko „dominujący” składnik i pomijamy czynniki stałe. Stąd złożoność algorytmu możemy opisać symbolem $O(n\log_2 n)$ lub nawet $O(n\log n)$ (podstawa logarytmu nie wpływa na złożoność rząd wielkości funkcji).