Упражнение 4 (сложност на итеративни и рекурсивни алгоритми)

Някои важни суми:

$$\sum_{i=1}^{n} 1 = n, \qquad \sum_{i=a}^{b} 1 = b - a + 1, \qquad \sum_{i=1, i=i+k}^{n} 1 = \left\lceil \frac{n}{k} \right\rceil \approx \frac{n}{k}$$

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \Theta(n^2), \qquad \sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} = \Theta(n^3)$$

$$\sum_{i=1}^{n} i^{\alpha} = \begin{cases} \Theta(n^{\alpha+1}) & , \alpha > -1 \\ \Theta(\log n) & , \alpha = -1 \\ \Theta(1) & , \alpha < -1 \end{cases}$$

Тези отношения могат да бъдат изведени по различни начини. Първите 5 могат да се докажат лесно по индукция или да се проверят непосредствено. Шестото е следствие от интегралния критерий за суми.

В следващите примери условието е да се намери сложността по време на дадения фрагмент.

Пример 1:

```
for (int i = 1; i <= n; i++) printf("a");
```

Времето, за което се изпълнява цикълът може да се изрази със сумата:

$$\sum_{i=1}^{n} 1 = n$$

Идеята е, че той прави n стъпки като на всяка от тях извършва работа за константно време (ще отбелязваме това с 1).

Пример 2:

```
for (int i = 1; i <= n; i++) for (int j = 1; j <= n; j++)
printf("a");
```

Времето, за което се изпълняват циклите може да се изрази със сумата:

$$\sum_{i=1}^{n} \sum_{j=1}^{n} 1 = \sum_{i=1}^{n} n = n^2$$

Пример 3:

```
for (int i = 1; i <= n; i++) for (int j = 1; j <= i; j++)
     printf("a");
```

Времето, за което се изпълняват циклите може да се изрази със сумата:

$$\sum_{i=1}^{n}\sum_{j=1}^{i} 1 = \sum_{i=1}^{n} i = \Theta(n^2)$$

Пример 4:

```
for (int i = 1; i <= n; i++) for (int j = 1; j <= n; j += i)
     printf("a");
```

Времето, за което се изпълняват циклите може да се изрази със сумата:

$$\sum_{i=1}^{n}\sum_{j=1,j=j+i}^{n} 1 \approx \sum_{i=1}^{n}\frac{n}{i} = n\sum_{i=1}^{n}\frac{1}{i} = \Theta(n\log n)$$

Пример 5:

```
for (int i = 1; i <= n; i++) for (int j = 1; j <= n; j++)
     if (i == j) for (int k = 1; k <= n; k++) printf("a");
```

Първите два вложени цикъла определят време за изпълнение n^2. Третият цикъл се изпълнява само в случайте, когато $i = j$, а те са точно n на брой ($i = j = 1, i = j = 2$ и така нататък). Той от своя страна е с време n, така че за него имаме общо време n^2.

Цялото време на изпълнение се определя от времето на двата вложени цикъла и случаите, когато се изпълнява третият цикъл, т.е. $n^2 + n^2 = 2n^2 = \Theta(n^2)$.

Пример 6:

```
for (int i = 1; i <= n; i++) for (int j = 1; j <= n; j+=i)
     for (int k = 1; k <= n; k += i) printf("a");
```

Времето, за което се изпълняват циклите може да се изрази със сумата:

$$\sum_{i=1}^{n}\sum_{j=1,j=j+i}^{n}\sum_{k=1,k=k+i}^{n} 1 \approx \sum_{i=1}^{n}\sum_{j=1,j=j+i}^{n}\frac{n}{i} = \sum_{i=1}^{n}\left(\frac{n}{i}.\sum_{j=1,j=j+i}^{n} 1\right) \approx \sum_{i=1}^{n}\frac{n^2}{i^2} = n^2.\sum_{i=1}^{n}\frac{1}{i^2} = \Theta(n^2)$$

Пример 7: Да се намери сложността и резултатът от следната рекурсивна програма:

```
int a(int n)
{
    int s = 0;
    for (int i = 1; i < n; i++) s += 2*a(i) + 1;
    return s;
}
```

Анализ на сложност:

На всяка итерация на цикъла, програмата се обръща към себе си с по-малък аргумент.

Умножението с 2 обаче е константа операция, така че реално програмата върши само веднъж работата за този по-малък аргумент, а не два пъти!

Ако изразът беше `s += a(i) + a(i) + 1`, то резултатът щеше да е същият, но сложността щеше да е друга, тъй като в този случай програмата реално щеше да върши два пъти работата за по-малкия аргумент.

Сложността може да се опише със следната рекурентна зависимост:

$$T(n) = T(1) + 1 + T(2) + 1 + \ \ldots + T(n-1) + 1 = \sum_{i=1}^{n-1} T(i) + n - 1$$

Тази единица в израза показва допълнителната константна работа, която върши алгоритъмът.

Аналогично за $T(n-1)$ имаме:

$$T(n-1) = \sum_{i=1}^{n-2} T(i) + n - 2$$

, откъдето:

$$T(n) - T(n-1) = \sum_{i=1}^{n-1} T(i) + n - 1 - \sum_{i=1}^{n-2} T(i) - n + 2 = T(n-1) + 1 \Rightarrow$$

$$\Rightarrow T(n) = 2.T(n-1) + 1$$

Това рекурентно уравнение има общо решение: $T(n) = c_1.2^n + c_2 = \Theta(2^n)$, където $c_1 > 0$

Намиране на резултат:

Резултатът може да се опише със следната рекурентна зависимост:

$$S(n) = 2S(1) + 1 + 2S(2) + 1 + \ \ldots + 2S(n-1) + 1 = 2\sum_{i=1}^{n-1} S(i) + n - 1$$

Аналогично за $T(n-1)$ имаме:

$$S(n-1) = 2\sum_{i=1}^{n-2} S(i) + n - 2$$

, откъдето:

$$S(n) - S(n-1) = 2\sum_{i=1}^{n-1} S(i) + n - 1 - 2\sum_{i=1}^{n-2} S(i) - n + 2 = 2S(n-1) + 1 \Rightarrow$$

$$\Rightarrow S(n) = 3.S(n-1) + 1$$

Това рекурентно уравнение има общо решение: $S(n) = c_1.3^n + c_2$

Коефицентите ще намерим от $S(1) = 0$ и $S(2) = 1$

Получаваме системата:

$$\begin{cases} 0 = 3c_1 + c_2 \\ 1 = 9c_1 + c_2 \end{cases}$$

, с решение $c_1 = \frac{1}{6},\ c_2 = -\frac{1}{2}$

В такъв случай $S(n) = \frac{3^n - 3}{6}$, което е и резултатът от програмата.

Същият резултат можем да намерим и с много по-бърз алгоритъм:

```
int a1(int n)
{
    return (exp_by_sqr(3, n) - 3)/6;
}
```

, където `exp_by_sqr` е алгоритъмът от миналото упражнение.

Тъй като той винаги извиква себе си с два пъти по-малък вход относно n, са му необходими около $\lg n$ стъпки за да стигне до базите си, което и определя сложността му - $\Theta(\lg n)$. Това е несравнимо по-добре от 2^n.

За вход 10 `exp_by_sqr` ще прави около 4 стъпки, докато алтернативният алгоритъм ще прави поне 1024 стъпки.