Семестриално контролно по "Дизайн и анализ на алгоритми" (СУ, ФМИ, 13 май 2017 г.)

Име: ________________________________ ФН: _____ Спец.: ____ Курс: ___

Задача	1	2	3	4	5	6	Общо
получени точки							
максимум точки	20	20	20	20+10	10+20	20	140

Забележка: За отлична оценка са достатъчни 100 точки!

Задача 1. Намерете времевата сложност на алгоритъма.

```
P(A[1...n]: array of integers)
    s ← 901
    for k ← 1 to n
        s ← s + A[k] × A[k]
    if n = 1
        return s
    s ← s + P(A[1...n−1])
    return s
```

Задача 2. Намерете времевата сложност на алгоритъма.

```
Q(A[1...n]: array of integers)
    s ← 648
    for i ← 1 to n
        for j ← 1 to n
            s ← s + A[i] + A[j]
    if n < 200
        return s
    for k ← 1 to 81
        h ← k + ⌊n/3⌋ − 1
        x ← Q(A[k...h])
        s ← s + x
    return s
```

Задача 3. Какво връща следният алгоритъм?

```
f(a, b: non-negative integers)
    x ← a
    y ← b
    while y > 1 do
        x ← x + 2
        y ← y − 2
    if y = 1
        x ← x + 1
    return x
```

Дайте строга обосновка (например с инварианта).

Задача 4. Няколко души (n на брой) се канят да Ви гостуват. Тази информация е представена в масив $A[1 \ldots n]$ от цели положителни числа (може да са много големи): $A[i] = k$ означава, че i-тият гост ще дойде след k дена ($k = 1$ — "утре"; $k = 2$ — "вдругиден" и т.н.). Предложете алгоритъм с максимална времева сложност $\mathrm{O}(n)$ за намиране на първия от предстоящите дни, през който няма да имате гост.

Бонус: Дават се още 10 точки, ако алгоритъмът използва допълнителна памет с размер $\mathrm{O}(\log n)$ при спазено изискване за коректност и за времева сложност $\mathrm{O}(n)$ в най-лошия случай.

Задача 5. Разглеждаме алгоритмичната задача за намиране на броя на различните стойности в числов масив $A[1 \ldots n]$. Установете, че времевата сложност на тази задача е $\Theta(n \log n)$, ако тя се решава чрез сравнения. За целта:

а) съставете алгоритъм с време $\mathrm{O}(n \log n)$, основан на сравнения; **(10 точки)**

б) докажете, че задачата изисква време $\Omega(n \log n)$ в най-лошия случай. **(20 точки)**

Задача 6. Предложете алгоритъм за сливане на k сортирани масива с общо n елемента за време $\mathrm{O}(n \log k)$.

РЕШЕНИЯ

Задача 1. Времевата сложност на алгоритъма удовлетворява рекурентното уравнение $T(n) = T(n-1) + \Theta(n)$, където $T(n-1)$ е времето за рекурсивното извикване от ред № 6. Събираемото $\Theta(n)$ е времето за изпълнение на всички други команди; от тях най-много време изразходва цикълът на редове № 2 и № 3 (тялото му се изпълнява n пъти); останалите команди се изпълняват по веднъж.

Уравнението $T(n) = T(n-1) + \Theta(n)$ е линейно-рекурентно уравнение. То може да се реши чрез характеристично уравнение или чрез развиване. Получава се $T(n) = \Theta\left(n^2\right)$.

Задача 2. Тялото на цикъла на редове № 7 – № 10 се изпълнява 81 пъти. Всяка итерация изразходва време $T\left(\dfrac{n}{3}\right)$ за рекурсията на ред № 9, време $\Theta(1)$ — за останалите команди. Затова цикълът на редове № 7 – № 10 изразходва общо време $81\, T\left(\dfrac{n}{3}\right)$.

Вложените цикли на редове № 2 – № 4 изискват време $\Theta\left(n^2\right)$. За другите команди — на редове № 1, № 5 и № 6 — е нужно време $\Theta(1)$.

Следователно $T(n) = 81\, T\left(\dfrac{n}{3}\right) + \Theta\left(n^2\right)$. От мастър-теоремата намираме $T(n) = \Theta\left(n^4\right)$.

Задача 3. Алгоритъмът връща сбора на двете числа, т.е. $f(a,\, b) = a + b$. Това се доказва с помощта на инварианта на цикъла с начало на ред № 3: всеки път, когато се проверява условието за край на цикъла, е в сила равенството $x + y = a + b$ и променливите a и b имат първоначалните си стойности.

Задача 4. Търси се най-малкото цяло положително число, липсващо в масива $A[1 \ldots n]$. То е някое от числата $1,\ 2,\ 3,\ \ldots,\ n,\ n+1$, следователно е малко в сравнение с n. Ето защо можем да използваме идеята на алгоритъма *сортиране чрез броене*.

```
FirstFreeDay(A[1...n]: array of integers)
C[1...n]: array of Boolean values    // C[k] = true ⟺ ∃ i (A[i] = k)
for k ← 1 to n
    C[k] ← false    // no k found yet
for k ← 1 to n
    if A[k] ≤ n
        C[A[k]] ← true
for k ← 1 to n
    if C[k] = false
        return k
return n + 1
```

Анализ на алгоритъма: Всеки от трите цикъла изразходва време $\Theta(n)$, откъдето следва, че общото време на алгоритъма е $T(n) = \Theta(n)$. Количеството допълнителна памет е $M(n) = \Theta(n)$ заради масива $C[1 \ldots n]$.

Можем да намалим сложността по памет, като се възползваме от възможността за промяна на входните данни. Знаците на числата от масива A ще поемат ролята на логически стойности: положителните числа ще съответстват на false, а отрицателните — на true. Така отпада нуждата от масива C.

```
FirstFreeDay(A[1...n]: array of integers)
for k ← 1 to n
    if abs(A[k]) ≤ n
        A[abs(A[k])] ← −abs(A[abs(A[k])])
for k ← 1 to n
    if A[k] > 0
        return k
return n + 1
```

Анализ на алгоритъма: Всеки от двата цикъла изразходва време $\Theta(n)$, откъдето следва, че общото време е $T(n) = \Theta(n)$. Сложността по памет е $M(n) = \mathrm{O}(\log n)$ заради брояча k.

Задача 5. Алгоритмичната задача CntUnique има времева сложност $\Theta(n \log n)$, ако се решава чрез сравнения.

а) За да докажем горната граница $\mathrm{O}(n \log n)$ на времевата сложност на задачата, съставяме достатъчно бърз алгоритъм:

```
CntUnique(A[1...n]: array of numbers)
Sort(A)   // например пирамидално сортиране или сортиране чрез сливане
cnt ← 1
for k ← 2 to n
    if A[k] ≠ A[k − 1]
        cnt ← cnt + 1
return cnt
```

Анализ на алгоритъма: Нужно е време $\Theta(n \log n)$ за сортирането и $\Theta(n)$ — за цикъла след това. Общото време на алгоритъма е $T(n) = \Theta(n \log n) + \Theta(n) = \Theta(n \log n)$.

б) Долната граница $\Omega(n \log n)$ се доказва чрез редукция от задачата ElementUniqueness, за която знаем, че изисква време $\Omega(n \log n)$, когато се решава чрез сравнения.

```
ElementUniqueness(A[1...n]: array of numbers)
if CntUnique(A[1...n]) = n
    return true
else return false
```

Коректност на редукцията: Няма повторения $\Longleftrightarrow$ броят на различните стойности е n.

Бързина на редукцията: Редукцията се състои в сравнението на върнатата стойност с n. То изисква константно време $\Theta(1) = \mathrm{o}(n \log n)$, следователно редукцията е достатъчно бърза.

Задача 6. За да получим бърз алгоритъм, ще използваме къса приоритетна опашка — с k, а не с n елемента. Приоритетна опашка може да се реализира чрез двоична пирамида или чрез пирамида на Фибоначи. Елементите ѝ ще бъдат наредени двойки $\langle value,\ array \rangle$, където $value$ е някое от дадените n числа, а $array$ е номерът на масива, от който е взето то (следователно $array$ приема целочислени стойности от 1 до k включително).

Отначало слагаме в опашката първия елемент на всеки от дадените k масива. После извличаме от опашката елемента $\langle value,\ array \rangle$ с най-малка стойност на полето $value$, изпращаме числото $value$ към изхода на алгоритъма и добавяме към опашката следващото число от масива $array$. За целта ще ни трябват k индекса — по един за всеки масив, — които да показват докъде е обходен всеки от дадените k масива.

Продължаваме да изваждаме и добавяме елементи към опашката, докато през нея премине всяко от дадените n числа. Тези две операции изразходват време $\mathrm{O}(\log k)$, защото във всеки миг опашката съдържа не повече от k елемента. (Тя съдържа по-малко от k елемента по време на инициализацията, а също и към края на алгоритъма, когато масивите започнат да се изчерпват.)

Понеже всяко от дадените n числа се добавя и премахва от опашката точно веднъж, то времето на целия алгоритъм е $\mathrm{O}(n \log k)$.