Поправителен изпит по ДАА, 19.02.2015г.

Име: _____, ФН:_____, Спец./курс:_____

Задача	1	2	3	4	5	6	Общо
получени точки							
максимум точки	20	20	20	20	20	20	120

Забележка: За отлична оценка са достатъчни 100 точки!

Задача 1 Решете следните рекурентни уравнения:

а) $T(n) = 4T(\frac{n}{3}) + n$ б) $T(n) = 2T(\frac{n}{\sqrt{2}}) + n^2$

в) $T(n) = T(n-1) + \frac{1}{n}$ г) $T(n) = \sum_{i=0}^{n-1} T(i) + 2^n$

Задача 2 Алгоритъм сортирал масив с 2 000 000 числа за около 100 милисекунди, а масив с 6 000 000 числа — за около 900 милисекунди (на същия компютър).

С кой от следните алгоритми най-вероятно се е случило това: HeapSort, InsertionSort или MergeSort ? (Не се приема отговор без обосновка!)

Задача 3 Предложете алгоритъм, разпознаващ за време $o(n^2)$ дали два едномерни масива с дължина n имат общи елементи (не непременно на едни и същи позиции).

Задача 4 Опитайте да разделите на две групи с равна сума редиците от числа, дадени по-долу. Ако това е невъзможно, предложете кратко доказателство.

(a - 10 точки) $12, 7, 31, 14, 17, 22$

(b - 10 точки) $12, 9, 31, 15, 18, 27$

Задача 5 Разглеждаме следните две задачи за разпознаване:

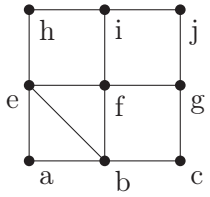
$3PLine$: Дадено е множество от n точки в равнината. Има ли сред тях три точки, лежащи на една права?

$TArea$: Дадено е множество от n точки в равнината и цяло неотрицателно число S. Има ли сред дадените точки три, които образуват триъгълник с лице, не по-голямо от S ?

Намерете полиномиална сводимост $3PLine \propto TArea$ или докажете съществуването на такава сводимост.

Забележка: Предполага се, че координатите на всички точки са цели числа.

Задача 6 Котка гони мишка в нарисувания по-долу граф:



Отначало котката е във връх a, а мишката — в j. Двете се придвижват последователно, като първа е котката. Всеки ход е придвижване по ребро в графа. Котката побеждава, ако достигне върха, в който е мишката.

Намерете печеливша стратегия за котката.

РЕШЕНИЯ

Задача 1.

а) С помощта на мастър-теоремата: $\log_3 4 > \log_3 3 = 1$, т.е. $\log_3 4 > 1$, където 1 е степента на свободния член, затова $T(n) = \Theta\left(n^{\log_3 4}\right) \approx \Theta\left(n^{1{,}26}\right)$.

б) Отново с помощта на мастър-теоремата: $\log_{\sqrt{2}} 2 = 2$, където 2 е степента на свободния член, затова $T(n) = \Theta\left(n^2 . \log n\right)$.

в) Уравнението се решава чрез разгръщане: развиваме дясната страна, като последователно заместваме аргумента n с n–1, n–2 и т.н. до 1 вкл.

$$T(n) = T(n-1) + \frac{1}{n}\ , \quad T(n-1) = T(n-2) + \frac{1}{n-1}\ , \quad T(n-2) = T(n-3) + \frac{1}{n-2}\ ,$$

и тъй нататък до $T(3) = T(2) + \frac{1}{3}$, $T(2) = T(1) + \frac{1}{2}$ и $T(1) = T(0) + 1$.

Следователно

$$T(n) = T(n-1) + \frac{1}{n} = T(n-2) + \frac{1}{n-1} + \frac{1}{n} = T(n-3) + \frac{1}{n-2} + \frac{1}{n-1} + \frac{1}{n} = \ldots$$

$$= T(0) + 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n-2} + \frac{1}{n-1} + \frac{1}{n} \approx C + \gamma + \ln n,$$

където $C = T(0) = \text{const.}$, $\gamma \approx 0{,}5772$ е константата на Ойлер. След като пренебрегнем константните събираеми, получаваме $T(n) = \Theta(\ln n)$.

Заб.1. Използвахме следното твърдение от математическия анализ:

$$1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n} - \ln n \to \gamma \quad \text{при} \quad n \to \infty .$$

Заб.2. Вместо това можем да използваме интеграл (тъй като се интересуваме само от асимптотичния порядък):

$$1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n} \approx \int_1^n \frac{1}{x}\,\mathrm{dx} = \ln x \Big|_1^n = \ln n - \ln 1 = \ln n - 0 = \ln n .$$

г) Разписано подробно, уравнението гласи:

$T(n) = T(0) + T(1) + T(2) + \mathbf{...} + T(n-2) + T(n-1) + 2^n$.

Това е линейно рекурентно уравнение с променлива дължина на историята.

Заместваме n с n–1 и получаваме второ уравнение:

$T(n-1) = T(0) + T(1) + T(2) + \mathbf{...} + T(n-2) + 2^{n-1}$.

От първото уравнение вадим второто и повечето събираеми се унищожават:

$T(n) - T(n-1) = T(n-1) + 2^n - 2^{n-1}$, т.е. $T(n) = 2T(n-1) + 2^{n-1}$, т.е.

$T(n) = 2T(n-1) + \frac{1}{2} \cdot 2^n$, което е линейно рекурентно уравнение с фиксирана дължина на историята. Съответното му характеристично уравнение е $\lambda^n = 2\lambda^{n-1}$. Тъй като не се интересуваме от нулевите корени, делим на λ^{n-1} и получаваме единствен (прост) корен $\lambda = 2$. Към него добавяме основата 2 на показателната функция в свободния член и получаваме мултимножеството $\{ 2 ; 2 \}_{\text{М}}$ — с негова помощ съставяме формулата $T(n) = C_1 \cdot 2^n + C_2 \cdot n \cdot 2^n$. Запазваме само събираемото от най-висок порядък и пренебрегваме константните множители. Окончателно, $T(n) = \Theta(n \cdot 2^n)$.

<u>Задача 2.</u> Щом 3 пъти по-голям масив се сортира 9 пъти по-бавно, времевата сложност е квадратична. От трите алгоритъма само InsertionSort е такъв.

<u>Задача 4.</u> Числата <u>не могат</u> да се разделят в две групи с равни суми, защото:

а) сборът им е нечетен (има три нечетни числа); ако разделянето беше възможно, сборът щеше да бъде четен ($2.S$, където S е сумата на числата във всяка група);

б) всички числа се делят на 3 с изключение на едно число (числото 31); както и да ги разпределяме в две групи, сборът на числата в едната група (която не съдържа числото 31) ще се дели на 3, а в другата група — не.

Задача 3. Нека n_1 и n_2 са дължините на масивите. Наивният алгоритъм, сравняващ всеки елемент на единия масив с всеки елемент на другия масив, изисква време $\Theta(n_1 . n_2)$ в най-лошия случай (когато масивите нямат общи елементи). При $n_1 = n_2 = n$ времето е $\Theta(n^2)$, т.е. повече от допустимото.

Проверката за общи елементи се ускорява с помощта на сортиране:

```
bool Intersect(A1[1...n1], A2[1...n2]: arrays of integers)
Sort(A1); Sort(A2) // сортираме масивите във възходящ ред
k1 ← 1 ; k2 ← 1
while k1 < n1 and k2 < n2
  if A1[k1] < A2[k2]
    k1 ← k1 + 1  // A1[k1] не се среща в масива A2
  else if A1[k1] > A2[k2]
    k2 ← k2 + 1  // A2[k2] не се среща в масива A1
  else // A1[k1] = A2[k2]
    return true   // намерен е общ елемент
return false      // масивите нямат общи елементи
```

Този алгоритъм изисква време $\Theta(n_1 . \log\ n_1) + \Theta(n_2 . \log\ n_2) + \Theta(n_1) + \Theta(n_2)$ (първите две събираеми са времето за сортиране на двата масива, а третото и четвъртото събираемо са времето за обхождане на масивите, т.е. същинската проверка за наличие на общи елементи). Като опростим този израз, получаваме времева сложност $\Theta(n . \log\ n)$, където n е по-голямото от числата n_1 и n_2 (или тяхната обща стойност, когато са равни). Тъй като $\Theta(n . \log\ n) = o(n^2)$, то този алгоритъм е достатъчно бърз.

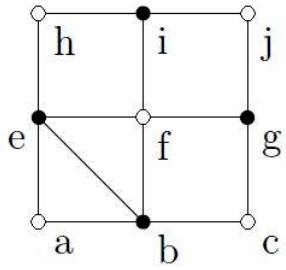
Ако дължините на масивите се различават значително, то този алгоритъм може да се окаже по-бавен от наивния. Например, ако $n_2 = \Theta(\log\ \log\ n_1)$, то времето на алгоритъма със сортиране е $\Theta(n_1 . \log\ n_1)$, докато на наивния алгоритъм е $\Theta(n_1 . n_2) = \Theta(n_1 . \log\ \log\ n_1)$, което е на порядък по-малко.

Задача 5. Задачата *3PLine* се свежда до *TArea* при $S = 0$: лицето на триъгълник е нула точно тогава, когато върховете му са колинеарни. По-формално, ако `bool Triangle_Area(A[1...n]: array of points; S: number)` е функция, която връща `true` тогава и само тогава, когато някои три точки образуват триъгълник с лице, ненадвишаващо неотрицателното число S, то

```
bool Three_Points_On_Line(A[1...n]: array of points)
return Triangle_Area(A, 0)
```

е функция, която връща `true` само тогава, когато някои три точки са колинеарни. Редукцията е полиномиална, защото изисква константно време.

Задача 6. Оцветяваме върховете на графа шахматно — както е показано на картинката. Печелившата стратегия на котката е да отиде от върха **a** до върха **f**, като премине през върховете **b** и **e**, т.е. **a–b–e–f** или **a–e–b–f** (общо три хода).



На всеки ход се променя цветът на полето на мишката, затова след три хода мишката ще се намира на черно поле, т.е. на **b**, **e**, **i** или **g**. Затова котката (която в този момент е на полето **f**) може да хване мишката на четвъртия ход.

Да отбележим, че котката непременно трябва да използва реброто **be**: в противен случай цветът на нейното поле също ще се променя на всеки ход, поради което тя след всеки свой ход ще се намира на поле, различно по цвят от полето на мишката. Следователно, ако котката не използва реброто **be**, тя не може да залови мишката (освен ако мишката сама не отиде при котката).