ПЪРВО КОНТРОЛНО ПО ДИЗАЙН И АНАЛИЗ НА АЛГОРИТМИ
ЗА СПЕЦИАЛНОСТ “КОМПЮТЪРНИ НАУКИ” — 2. КУРС, 1. ПОТОК
(СУ, ФМИ, 24 ЮНИ 2020 УЧ. Г.)

Указания:

1) Изучените алгоритми за сортиране и търсене могат да се използват наготово.
2) За всяко сортиране или търсене да се уточнява името на избрания алгоритъм.
3) Имената на всички алгоритми и структури от данни да са на български език.

Задача 1. Даден е масив $A[1\,...\,n]$ от реални числа, между които няма равни. Съставете алгоритъм с времева сложност $O(n)$ при най-лоши входни данни, който пренарежда масива така, че всяко число с изключение на двете крайни да бъде или по-голямо от двата си съседа, или по-малко от тях. С други думи, трябва да пренаредите масива така, че $A[1] < A[2] > A[3] < A[4] > A[5] < \ldots$ или $A[1] > A[2] < A[3] > A[4] < A[5] > \ldots$

а) Опишете алгоритъма с думи или с псевдокод, или с код на Си. **(8 точки)**

б) Изпълнете алгоритъма върху масива $A = (6;\ 3;\ 5;\ 1;\ 4;\ 2;\ 7)$. **(8 точки)**

в) Да се анализира времевата сложност на алгоритъма. **(4 точки)**

Задача 2. Даден е масив $A[1\,...\,n]$ от произволни реални числа, между които може да има, а може и да няма еднакви. Разглеждаме алгоритъм за сортиране, известен като метод на джуджетата.

```
Sort(A[1...n]) // Метод на джуджетата
k ← 1
while k ≤ n do
    if k = 1
        k ← k + 1
    else if A[k] ≥ A[k-1]
        k ← k + 1
    else
        размени A[k] и A[k-1]
        k ← k − 1
```

За описания алгоритъм отговорете на следните въпроси с подробна обосновка:

а) На коя изучена сортировка прилича методът на джуджетата? **(6 точки)**

б) Какви входни данни са най-лоши за бързодействието на метода на джуджетата? **(6 точки)**

в) Каква е времевата сложност $T(n)$ на метода на джуджетата при най-лоши входни данни? **(8 точки)**

РЕШЕНИЯ

Задача 1. За всяко цяло k от 1 до $n-1$ вкл. разместваме A[k] и A[$k+1$] само когато k е четно и A[k] < A[$k+1$] или k е нечетно и A[k] > A[$k+1$].

```
ALG(A[1...n])
1) kIsOdd ← true
2) for k ← 1 to n-1 do
3)    if (A[k] > A[k+1]) = kIsOdd
4)       swap(A[k],A[k-1]) // размяна на A[k] и A[k-1]
5)    kIsOdd = not kIsOdd
```

Изпълнение на алгоритъма върху масива A = (6 ; 3 ; 5 ; 1 ; 4 ; 2 ; 7):
— Стъпка № 1 (нечетен номер): Елементите 6 и 3 се разместват, защото 6 > 3. След тази стъпка масивът изглежда така: A = (3 < 6 ; 5 ; 1 ; 4 ; 2 ; 7).
— Стъпка № 2 (четен номер): Елементите 6 и 5 не се разместват, защото 6 > 5. След тази стъпка масивът изглежда така: A = (3 < 6 > 5 ; 1 ; 4 ; 2 ; 7).
— Стъпка № 3 (нечетен номер): Елементите 5 и 1 се разместват, защото 5 > 1. След тази стъпка масивът изглежда така: A = (3 < 6 > 1 < 5 ; 4 ; 2 ; 7).
— Стъпка № 4 (четен номер): Елементите 5 и 4 не се разместват, защото 5 > 4. След тази стъпка масивът изглежда така: A = (3 < 6 > 1 < 5 > 4 ; 2 ; 7).
— Стъпка № 5 (нечетен номер): Елементите 4 и 2 се разместват, защото 4 > 2. След тази стъпка масивът изглежда така: A = (3 < 6 > 1 < 5 > 2 < 4 ; 7).
— Стъпка № 6 (четен номер): Елементите 4 и 7 се разместват, защото 4 < 7. След тази стъпка масивът изглежда така: A = (3 < 6 > 1 < 5 > 2 < 7 > 4).
— Последната подредба на масива е резултатът от работата на алгоритъма.

Инвариант на цикъла: При всяко изпълнение на проверката за край на ред 2 важат неравенствата A[1] < A[2] > A[3] < A[4] > . . . A[k] (знаците се редуват), а пък kIsOdd е истина, ако и само ако k е нечетно число.

Доказателство: с индукция по номера на проверката за край на цикъла.

База: При влизане в цикъла числото $k = 1$ е нечетно и kIsOdd е истина съгласно с ред № 1 от програмния код, а веригата от неравенства съдържа $k - 1 = 0$ неравенства, и е тривиално вярна (като празна конюнкция).

Индуктивна стъпка: Всяко изпълнение на редове 3 и 4 добавя неравенство, без да разваля постигнатото, защото при размяна елементът с по-малък индекс, ако е бил по-малък (по-голям) от предходния, става още по-малък (по-голям).

Полуинвариант е броячът k. Той расте с единица след всяко изпълнение на тялото на цикъла, като се мени от 1 до n, т.е. тялото се изпълнява $n - 1$ пъти, затова времевата сложност на алгоритъма е $\Theta(n)$ при всякакви входни данни.

Коректността на алгоритъма следва от инварианта при последната проверка. Тогава $k = n$ и A[1] < A[2] > A[3] < A[4] > . . . A[n].

Задача 1 може да се реши по още един начин:

1) С алгоритъма PICK намираме медианата на дадения масив A[1 . . . n].
2) Разделяме елементите на масива на големи и малки относно медианата. Причисляваме медианата към една от двете части на масива така, че дължините им да се различават с не повече от единица.
3) Пренареждаме елементите, като редуваме двете части на масива: слагаме ту голям, ту малък елемент. Ако двете части имат различни дължини, започваме с по-дългата част. Ако имат равни дължини, няма значение с коя част започваме.

П р и м е р : Нека входният масив е A = (6; 3; 5; 1; 4; 2; 7).

1) С алгоритъма PICK намираме медианата на масива. Тя е числото 4.
2) Разделяме елементите на масива на по-големи и по-малки от 4:
   — малки: **3**; **1**; **2**;
   — големи: **6**; **5**; **7**.
   Няма значение към коя половина ще причислим медианата. Да я сложим например при големите елементи:
   — малки: **3**; **1**; **2**;
   — големи: **4**; **6**; **5**; **7**.
3) Редуваме двете части, като започваме с по-дългата, тоест с голямо число:
   $\mathbf{4} > \mathbf{3} < \mathbf{6} > \mathbf{1} < \mathbf{5} > \mathbf{2} < \mathbf{7}$.

Алгоритъмът е коректен, защото големите числа са по-големи от медианата, а тя е по-голяма от малките числа (понеже между числата в масива няма равни).

Всяка стъпка има линейна времева сложност при произволни входни данни, затова общата времева сложност на алгоритъма също е линейна: $\Theta(n)$.

Задача 2. Методът на джуджетата прилича на сортирането чрез вмъкване, само че е записан с един цикъл вместо с два вложени цикъла. По-точно, външният цикъл на сортирането чрез вмъкване (този, който взима елементите на масива един по един) съответства на увеличението на индекса k с единица в метода на джуджетата (редове 4 и 6); а пък вътрешният цикъл на алгоритъма сортиране чрез вмъкване (този, който търси мястото на текущия елемент) съответства на намаляването на индекса k с единица в метода на джуджетата (редове 8 и 9). Разлика между алгоритмите: след като методът на джуджетата намери мястото на текущия елемент сред вече сортираните, индексът k не може да се върне към следващия текущ елемент направо, а трябва да стигне до него стъпка по стъпка (в сортирането чрез вмъкване мястото на текущия елемент се пази в брояча на външния цикъл). Следователно методът на джуджетата е около два пъти по-бавен от сортирането чрез вмъкване в най-лошия случай.

Най-лошият случай за входните данни е един и същ за двата алгоритъма: $A[1] > A[2] > \ldots > A[n]$. При такива входни данни броят на увеличенията и намаленията на индекса k е равен на

$$1 + 2 + 1 + 4 + 1 + 6 + 1 + 8 + 1 + 10 + \ldots + 1 + 2(n - 1),$$

където всяка единица съответства на промяната на текущия елемент, а всяко събираемо от вида $2(k - 1)$ съответства на броя на размените на k-тия елемент, докато стигне първото място в масива (множителят две идва от това, че всяко намаление на индекса k поражда съответно увеличение с единица по-късно — когато индексът се връща към мястото, откъдето е взел елемента).

Полученият сбор съдържа $n - 1$ единици, а четните събираеми образуват аритметична прогресия с $n - 1$ члена, първият от които е равен на 2; толкова е и разликата на прогресията. Затова сборът има стойност

$$(n - 1) + 2\,[1 + 2 + 3 + \ldots + (n - 1)] \;=\; (n - 1) + (n - 1)\,n \;=\; n^2 - 1 \;=\; \Theta(n^2).$$

Това е общият брой изпълнения на редове 4, 6 и 9 от метода на джуджетата, а следователно и времевата сложност на метода при най-лоши входни данни. Окончателно, времевата сложност на метода на джуджетата е $\mathrm{T}(n) = \Theta(n^2)$.

СХЕМА ЗА ТОЧКУВАНЕ

Контролното носи 40 точки, разпределени между двете задачи поравно, тоест по 20 точки за всяка задача, както е указано в условията на задачите.

Задача 1 носи точки само ако предложеният алгоритъм е верен и бърз.

Отговорите по задача 2 носят точки само ако са обосновани подробно.