С О Р Т И Р А Н Е И Т Ъ Р С Е Н Е
КОНТРОЛНО № 2 ПО "ДИЗАЙН И АНАЛИЗ НА АЛГОРИТМИ"
ЗА СПЕЦИАЛНОСТ "КОМПЮТЪРНИ НАУКИ", 2. ПОТОК
(СУ, ФМИ, ЛЕТЕН СЕМЕСТЪР НА 2019 / 2020 УЧ. Г.)

Всеки алгоритъм да се опише ясно и да се изпълни с подходящи входни данни до крайния си резултат. Изучените алгоритми могат да се използват наготово. При всяко сортиране да се уточнява названието на използваната сортировка. Имената на алгоритмите и структурите от данни да бъдат на български език.

Задача 1. В електрическа верига има n консуматора, свързани последователно. Един консуматор е изгорял, но не се знае точно кой. Можем да изпробваме веригата, като подаваме напрежение между кои да е две нейни точки. Смятаме, че във веригата има $n + 1$ точки, в които можем да подаваме ел. напрежение: $n - 1$ от тях са точките, в които консуматорите се свързват един с друг, и има още две точки — крайните точки на веригата. Нека точките са номерирани с целите числа от 0 до n, а консуматорите — с целите числа от 1 до n. Тогава консуматор № k е включен между точките № $k - 1$ и № k.

Искаме да намерим изгорелия консуматор с възможно най-малко проби. Опишете алгоритъма като функция `findDamage(n: integer): integer`, многократно извикваща функцията `test(i, j: integers): boolean`, която връща истина (`true`) точно тогава, когато участъкът от ел. веригата от точка № i до № j провежда ел. ток. Целта е алгоритъм, в който броят на извикванията на функцията `test` е минимален в най-лошия случай. Функцията `findDamage` трябва да връща номера на изгорелия консуматор. Алгоритъмът трябва да бъде описан на псевдокод.

Анализирайте времевата сложност на предложения алгоритъм. Не е нужно да доказвате, че той е най-бърз. **(3 точки)**

Задача 2. Съставете алгоритъм, проверяващ за време $O(n)$ в най-лошия случай дали в даден числов масив $A[1...n]$ повече от 40% от елементите са равни, тоест дали има число, което се среща повече от $0{,}4\,n$ пъти.

Опишете алгоритъма с думи, като го съставите от изучени алгоритми с времева сложност $O(n)$ в най-лошия случай. **(4 точки)**

Задача 3. Съставете алгоритъм, който за време $O(n \log n)$ в най-лошия случай намира по даден масив $A[1 \ldots n]$ от реални числа най-близкия до нулата сбор от два елемента на A с различни индекси. Опишете алгоритъма словесно. Анализирайте неговата времева сложност. **(3 точки)**

СХЕМА НА ТОЧКУВАНЕ

Контролното носи 10 точки, разпределени по задачи, както е показано по-горе. Всяка задача носи 1 точка за анализ на времевата сложност; останалите точки са за съставяне на алгоритъма.

РЕШЕНИЯ

Задача 1. Намираме изгорелия консуматор с помощта на двоично търсене:

```
findDamage(n: integer): integer
i ← 0
j ← n
while j - i > 1 do
    k ← ⌊(i + j) / 2⌋
    if test(i, k)
        i ← k
    else
        j ← k
return j
```

Заради последователното деление на две, времето на алгоритъма е $\Theta(\log n)$ във всички случаи. Тоест всички случаи са еднакво лоши по порядък.

Забележка: В алгоритъма се предполага, че съществува поне един изгорял консуматор. Ако има повече от един, алгоритъмът връща най-левия от тях, тоест изгорелия консуматор с най-малък номер. Ако искаме да проверим дали има поне една повреда, добавяме в началото проверка `test(0, n)`.

Задача 2. Интервала от 0 % до 100 % разбиваме на най-малък брой интервали, всеки с дължина, по-малка от 40 %. Понеже 100 : 40 = 2,5, ще са нужни поне три интервала, например от 0 % до 35 %, от 35 % до 65 % и от 65 % до 100 %. Ако повече от 40 % от елементите на масива $A[1...n]$ имат еднаква стойност, то тя непременно е стойност на някой от общите краища на подинтервалите, тоест 35-ия и 65-ия процентил.

Намираме стойностите на 35-ия и 65-ия процентил чрез алгоритъма PICK. За всяка от тези две стойности проверяваме колко пъти се среща, с помощта на едно обхождане на дадения числов масив $A[1...n]$. Ако някоя от тях се среща повече от $0{,}4n$ пъти, тя е търсената стойност. В противен случай никое число не се среща повече от $0{,}4n$ пъти в масива $A[1...n]$.

Алгоритъмът PICK и обхождането на масива имат времева сложност $\Theta(n)$.

Задача 3. Първи начин: сортираме числата по абсолютни стойности с някоя от бързите сортировки, например с пирамидално сортиране, за време $\Theta(n \log n)$. Две числа, чийто сбор е близък до нулата, имат близки абсолютни стойности. (Обратното не е вярно: сбор на две числа с близки абсолютни стойности може да бъде далече от нулата, ако числата са с еднакви знаци.) След сортирането проверяваме сборовете на всеки две съседни числа; връщаме най-близкия до 0.

Анализ на бързодействието: Сортирането изразходва време $\Theta(n \log n)$, а обхождането на масива след това — време $\Theta(n)$. Времето на целия алгоритъм е сборът от времената на двата етапа. Асимптотиката зависи от събираемото, което е по-голямо по порядък, затова времевата сложност на целия алгоритъм е равна на $\Theta(n \log n)$.

Втори начин: Сортираме числата с някоя от бързите сортировки, например сортиране чрез сливане. После за всеки елемент x на масива търсим елемента y (с различен индекс), който е най-близък до противоположното число $(-x)$; използваме двоично търсене, защото масивът вече е сортиран. Сборът $x + y$ е най-близък до 0 сред сборовете, в които участва елементът x. Измежду сборовете от вида $x + y$ (когато x пробягва всички елементи на дадения масив) избираме сбора, който е най-близък до 0.

Анализ на бързодействието: Сортирането изразходва време $\Theta(n \log n)$, а обхождането на масива след това — пак толкова, защото двоичното търсене на y отнема време $\Theta(\log n)$ за всеки отделен елемент x и се извършва n пъти — по веднъж за всеки елемент x на дадения масив. Времето на целия алгоритъм е сборът от времената на двата етапа, тоест $\Theta(n \log n)$.