СОРТИРАНЕ И ТЪРСЕНЕ

ПРИМЕРНО КОНТРОЛНО № 2 ПО "ДИЗАЙН И АНАЛИЗ НА АЛГОРИТМИ" — СУ, ФМИ (ЗА СПЕЦИАЛНОСТ "КОМПЮТЪРНИ НАУКИ", 1. ПОТОК, ФЕВРУАРИ – ЮНИ 2018 Г.)

Задача 1. Предложете бърз алгоритъм, който разделя $2n$ числа на двойки с най-голям сбор от абсолютните разлики. Входът е числов масив $A[1 \ldots 2n]$, а изходът е пермутация $i_1, j_1, i_2, j_2, \ldots, i_n, j_n$ на $1, 2, 3, \ldots, 2n$, за която

$$\left|A[i_1] - A[j_1]\right| + \left|A[i_2] - A[j_2]\right| + \ldots + \left|A[i_n] - A[j_n]\right| = \max.$$

Точкуване: Алгоритъм с $T(n) = \mathrm{O}(n \log n)$ и $M(n) = \mathrm{O}(\log n)$ носи 10 т. Алгоритъм със сложност $T(n) = \mathrm{O}(n)$ и $M(n) = \mathrm{O}(n)$ носи други 15 т. Обосновката за коректност носи 15 точки. Намирането на времевата сложност на алгоритмичната задача носи още 10 точки.

Решение: *Първи алгоритъм:* За време $\Theta(n \log n)$ *сортираме* масива A на място с помощта на пирамидалното сортиране. После за време $\Theta(n)$ групираме елементите на масива по двойки, например така**:**
$\left|A[2n] - A[1]\right| + \left|A[2n-1] - A[2]\right| + \ldots + \left|A[n+1] - A[n]\right|;$
или така**:** $\left|A[n+1] - A[1]\right| + \left|A[n+2] - A[2]\right| + \ldots + \left|A[2n] - A[n]\right|.$
Тези суми (и много други) имат една и съща (максимална) стойност**:**
$S = \big(A[n+1] + A[n+2] + \ldots + A[2n]\big) - \big(A[1] + A[2] + \ldots + A[n]\big).$

Коректност: Нека X е множеството на сумите, образувани според условието. След разкриване на модулите всяка такава сума има n събираеми с плюс и n събираеми с минус. Тоест $X \subseteq Y$, където Y е множеството на сумите от различни елементи на масива, като пред n от тях знакът е плюс, а пред другите n — минус. За да докажем, че max сума от X е S, трябва да покажем:
1) че в множеството X има сума със стойност S (това бе направено по-горе);
2) че максималната сума от Y има стойност S.
Второто твърдение се доказва така**:** за всяка сума от Y групираме събираемите с плюс в един голям израз (умаляемо), групираме и събираемите с минус в друг голям израз (умалител). За пример вж. скобите на S по-горе. Получената сума е най-голяма, когато умаляемото е максимално, а умалителят — минимален. Затова в умалителя трябва да влязат (със знак плюс) най-големите n елемента, а в умалителя (със знак минус) — най-малките n елемента. Остава да групираме елементите по два**:** един голям с един малък, без значение точно как.

Втори алгоритъм: От горните разсъждения се вижда, че е достатъчно да разделим елементите на големи и малки. За целта първо намираме медианата с алгоритъма PICK, после разбиваме масива (partition) относно медианата. И двете стъпки имат сложност $\mathrm{O}(n)$ по време и памет.

Алгоритмичната задача има времева сложност $\Theta(n)$**:** горна граница $\mathrm{O}(n)$ от втория алгоритъм и тривиална долна граница $\Omega(n)$ по размера на входа.

Задача 2. Сървър трябва да обработи n различни заявки през следващите няколко милисекунди. Обработката на всяка заявка отнема една милисекунда. Информацията е представена в масив $A\,[1 \ldots n]$ от цели положителни числа: $A\,[\,i\,] = k$ означава, че обработката на i-тата заявка ще започне k милисекунди след текущия момент. Намерете кога най-рано сървърът ще бъде свободен.

Точкуване: Алгоритъм с $T(n) = \mathrm{O}(n \log n)$ и $M(n) = \mathrm{O}(1)$ носи 10 точки. Алгоритъм със сложност $T(n) = \mathrm{O}(n)$ и $M(n) = \mathrm{O}(n)$ носи други 20 точки. Оптимизация на сложността на втория алгоритъм до памет $M(n) = \mathrm{O}(1)$ при запазване на времето $T(n) = \mathrm{O}(n)$ носи допълнителни 20 точки.

Решение: *Първи алгоритъм:* За време $\Theta(n \log n)$ *сортираме* масива A на място с помощта на пирамидалното сортиране. После за време $\Theta(n)$ търсим най-малкия елемент, различен от своя индекс и връщаме въпросния индекс. (Ако не намерим такъв индекс, то връщаме $n + 1$).

Втори алгоритъм: Поредният номер на първата свободна милисекунда не надвишава $n + 1$, т.е. върнатата стойност е малко число (в сравнение с n). Затова можем да използваме идеята на алгоритъма *сортиране чрез броене:* използваме допълнителен логически масив с n елемента, всеки от които показва дали съответният индекс на логическия масив се среща като стойност в числовия масив A.

Инициализираме елементите на логическия масив със стойности “неистина”. С едно обхождане на A определяме правилните стойности на елементите на логическия масив, пренебрегвайки стойностите в A, които са по-големи от n. После с едно обхождане на логическия масив намираме първия елемент със стойност “неистина”. Алгоритъмът връща неговия индекс.

Сложността на този алгоритъм е линейна — и по време (све обхождания), и по памет (заради логическия масив).

Оптимизация: Вторият алгоритъм може да се реализира с константно количество допълнителна памет (само за броячите на циклите) с помощта на следната идея. Тъй като входните данни са цели положителни числа, то следва, че техните знаци могат да играят ролята на битовете от логическия масив. Така логическият масив става излишен: вместо да записваме стойност “истина” в някой от елементите на логическия масив, правим отрицателно съответното число от входните данни.