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" — СУ, ФМИ
(ЗА СПЕЦИАЛНОСТ "КОМПЮТЪРНИ НАУКИ", 1. ПОТОК; 28 МАРТ 2018 Г.)

Забележка: Уточнявайте името на всеки използван алгоритъм за сортиране или търсене!

Задача 1. Учителка в детска градина иска да купи пъзели за k деца. В магазина има n пъзела, като $n > k$. Пъзелите имат различен брой части. Учителката иска да купи такива k пъзела, че разликата в броя на частите им да е минимална. По-точно, ако M и m са съответно максималният и минималният брой части сред закупените k пъзела, трябва разликата $M - m$ да бъде възможно най-малка.

а) Съставете алгоритъм с времева сложност $O(n \log n)$ в най-лошия случай. Алгоритъмът трябва да връща най-малката възможна стойност на разликата $M - m$.

Опишете алгоритъма на псевдокод като функция от вида

`Puzzles(A[1...n]: array of positive integers, k: positive integer)`

Входни данни: k е цяло положително число — броят пъзели, които трябва да бъдат купени; n е цяло положително число — броят на пъзелите в магазина; в сила е неравенството $n > k$; A е масив от n цели положителни числа — броят на частите, от които се състои всеки пъзел.

Изход (върната стойност): цяло неотрицателно число — най-малката възможна разлика $M - m$, където M и m са съответно максималният и минималният брой части сред купените k пъзела. Не е нужно алгоритъмът да печата индексите на закупените пъзели.

Изпълнете алгоритъма стъпка по стъпка с входни данни: $k = 3$; $A = (110;\ 20;\ 60;\ 80;\ 50)$. Покажете междинните резултати (след всяка стъпка) и крайния резултат (върнатата стойност).

Обяснете кои k пъзела трябва да бъдат закупени, за да се получи търсеният минимум. Обяснението трябва да важи в общия случай и да бъде онагледено с примерните входни данни от предишния абзац. **(1 точка)**

б) Докажете, че поставената алгоритмична задача изисква време $\Omega(n \log n)$. **(1 точка)**

Задача 2. Масивът `A[1...n]` съдържа големините на n различни книги (в брой байтове). Имаме флашка, която побира максимум X байта информация. Съставете бърз алгоритъм за избор на максимален брой книги, които могат да се запишат на флашката. Няма смисъл да записваме една книга няколко пъти: искаме максимален брой ***различни*** книги.

Алгоритъмът трябва да намира не само максималния брой книги, но и самите книги.

Опишете алгоритъма с думи и го демонстрирайте с подходящи примерни данни. Демонстрацията трябва да показва както междинните сметки, така и крайния резултат. Анализирайте времето на изпълнение в най-лошия случай. Не се приемат алгоритми без коректен анализ на времевата сложност.

Ако максималната времева сложност на алгоритъма е $O(n)$, ще получите **2 точки.**

В противен случай, ако максималната сложност е $O(n \log n)$, ще получите **1 точка.**

По-бавни алгоритми не носят точки.

Оценката = 2 + броя на получените точки.
Не се зачитат решения, които не отговарят на изискванията!

Р Е Ш Е Н И Я

Задача 1.

а) Използваме някоя бърза сортировка, например пирамидалното сортиране.

```
Puzzles(A[1...n]: array of positive integers, k: positive integer)
Sort(A) // HeapSort или MergeSort, или QuickSort
minDiff ← +∞
for i ← 1 to n+1-k do
   currentDiff ← A[i+k-1] - A[i] // M - m
   if currentDiff < minDiff
      minDiff ← currentDiff
      bestStart ← i      // Купуваме пъзелите
print bestStart          // от № bestStart до № bestStart+k−1 вкл.
return minDiff           // (това са номерата им след сортирането).
```

Анализ на алгоритъма: сортирането изразходва време $\Theta(n \log n)$ в най-лошия случай, обхождането на сортирания масив — $\Theta(n)$; общо: $\Theta(n \log n)$.

Демонстрация на алгоритъма при $k = 3$ и $A = (110;\ 20;\ 60;\ 80;\ 50)$.

1) Сортираме масива: $A = (20;\ 50;\ 60;\ 80;\ 110)$.
2) Сравняваме разликите от вида $A[\,i+k-1\,]-A[\,i\,]$, тоест $A[\,i+2\,]-A[\,i\,]$:
   $A[\,3\,]-A[\,1\,] = 60-20 = 40;$
   $A[\,4\,]-A[\,2\,] = 80-50 = 30;$
   $A[\,5\,]-A[\,3\,] = 110-60 = 50.$
   Най-малката от тези разлики е равна на 30.
3) Това е най-малката възможна разлика между максималния и минималния брой части на три пъзела: 30 части (стойността, върната от алгоритъма). Минимумът се достига, като закупим втория, третия и четвъртия пъзел — тези с 50, 60 и 80 пъзела съответно.

б) Използваме редукция от алгоритмичната задача `ElementUniqueness` (проверка за липса на повторения).

```
ElementUniqueness(A[1...n])
k ← 2
if Puzzles(A[1...n], k) > 0
   return true
else
   return false
```

Бързина на редукцията: Редукцията се състои от присвояването `k ← 2` и от проверката дали върнатата стойност е положителна. И двете действия изискват константно време $\Theta(1)$. Понеже $1 \prec n \log n$ (желаната долна граница), то редукцията е достатъчно бърза.

Коректност на редукцията: Ако най-малката разлика между броя на частите на два пъзела е равна на нула, то има пъзели с равен брой части, тоест има повторения.

Обратно, ако най-малката разлика между броя на частите на два пъзела е положителна, то и другите са положителни, затова няма пъзели с равен брой части, тоест няма повторения.

За да бъде решението напълно точно, трябва да бъде поправен следният недостатък. Входните данни на задачата `ElementUniqueness` могат да бъдат всякакви цели числа — положителни, отрицателни, нули. А входните данни на задачата `Puzzles` по условие са само положителни цели числа. Затова при тази редукция има опасност да бъдат подадени недопустими входни данни на функцията `Puzzles`. Това може да се поправи: намираме най-малкото число във входния масив и го изваждаме, намалено с единица, от всички числа (така те стават положителни). Ако най-малкото число е например –7, то от всички числа изваждаме –8, тоест добавяме 8.

```
ElementUniqueness(A[1...n])
min ← A[1]
for i ← 2 to n do
   if A[i] < min
      min ← A[i]
min ← min - 1
for i ← 1 to n do
   A[i] ← A[i] - min
k ← 2
if Puzzles(A[1...n], k) > 0
   return true
else
   return false
```

Бързина на редукцията: $\Theta(n)$ заради двата последователни цикъла. Понеже $n \prec n \log n$, то редукцията е достатъчно бърза.

Коректност на редукцията: Прибавянето на достатъчно голямо число гарантира, че всички числа в масива са станали положителни, тоест функцията `Puzzles` получава допустими входни данни. От друга страна, събирането на едно и също число с всички числа в масива запазва техните разлики, а значи запазва и върнатата стойност от `Puzzles`, поради което можем да се позовем на разсъжденията от предишната страница.

Проблемът с недопустимите стойности на `Puzzles` представлява логическа тънкост, поради което се приемат и решения, в които този проблем не е забелязан. Решения, в които проблемът е забелязан и отстранен, носят допълнителна единица към оценката.

Задача 2 може да бъде решена по различни начини.

Първи начин: чрез някоя от бързите сортировки, например пирамидалното сортиране.

1) Сортираме книгите, т.е. подреждаме файловете в растящ ред на размерите им.
2) Копираме файловете върху флашката, като започваме от най-малките и продължаваме последователно до изчерпване на свободното място или до изчерпване на файловете. Тази стъпка се реализира с цикъл, в който обхождаме сортирания масив `A` и натрупваме сбора на елементите му, докато ги изчерпим или докато сборът им надхвърли X.

Анализ на алгоритъма: сортирането изисква време $\Theta(n \log n)$ в най-лошия случай, обхождането и сумирането на сортирания масив — $\Theta(n)$; общо: $\Theta(n \log n)$.

Втори начин: без сортиране. Вместо това използваме алгоритъма PICK.

1) Намираме медианата на масива `A` с помощта на алгоритъма PICK.
2) Разделяме файловете на големи и малки относно медианата.
3) Пресмятаме сбора Y от големините на малките файлове.
4) Ако Y = X, копираме всички малки файлове, отказваме се от всички големи. Край.
5) Ако Y > X, отказваме се от всички големи файлове и рекурсивно извикваме алгоритъма върху малките файлове, като използваме същия капацитет X. Край на алгоритъма.
6) Ако Y < X, копираме всички малки файлове и рекурсивно извикваме алгоритъма върху големите файлове, като използваме останалия капацитет X – Y в ролята на X. Край.

Дъното на рекурсията е при $n = 1$. Тогава сравняваме капацитета X на флашката с размера `A[1]` на единствения файл: копираме файла, ако и само ако `A[1]` ≤ X.

Анализ на алгоритъма: Най-лошият случай е, когато стъпка № 4 не се изпълнява, тоест на всеки етап алгоритъмът изпълнява или стъпка № 5, или стъпка № 6.

Стъпки № 1, № 2, № 3, отказването и приемането на файлове в стъпки № 5 и № 6 — всички тези операции изискват линейно време $\Theta(n)$. От двете рекурсивни извиквания в стъпки № 5 и № 6 се изпълнява само едното; всяко от тях работи върху половината масив, защото разбиването на масива на големи и малки стойности е извършено спрямо медианата. Ето защо времевата сложност на алгоритъма удовлетворява рекурентното уравнение

$$T(n) = T\left(\frac{n}{2}\right) + \Theta(n).$$

От третия случай на мастър-теоремата следва, че решението на уравнението е $T(n) = \Theta(n)$, тоест алгоритъмът има линейна времева сложност.

Демонстрация на алгоритмите. Нека `A` = (50; 20; 80; 10) и X = 45.

Първият алгоритъм сортира масива: `A` = (10; 20; 50; 80). После започва да събира елементите му, като сравнява получените суми с капацитета X = 45:
10 ≤ 45, следователно файлът с размер 10 байта ще бъде копиран;
10 + 20 = 30 ≤ 45; следователно и файлът с размер 20 байта ще бъде копиран;
обаче 30 + 50 = 80 > 45, следователно за другите файлове няма достатъчно място.

Извод: Можем да копираме най-много два файла — тези с големини 10 и 20 байта.

Вторият алгоритъм намира медианата 35 и разбива масива: `A` = (20; 10 | 50; 80).
Сборът от размерите на малките файлове е Y = 20 + 10 = 30 < X = 45.
Затова алгоритъмът копира двата малки файла и преминава рекурсивно към изследване на големите файлове, като сега разполага с капацитет X – Y = 15. Това е новото X, а новият масив `A` = (50; 80). Неговата медиана е 65, разбиването е `A` = (50 | 80), сборът от размерите на малките файлове е Y = 50 > X = 15, затова алгоритъмът се отказва от големия файл (с размер 80). Следва рекурсия върху първата половина от масива, тоест новият масив е `A` = (50), капацитетът остава същият: X = 15.
Достигнато е дъното на рекурсията — масив с дължина единица. Неравенството 50 > 15 показва, че единственият останал файл не може да бъде копиран.

Извод: Можем да копираме най-много два файла — тези с големини 10 и 20 байта.