① Да се сортира масив $A[1..n]$ като $A[i] \in \{2, 3, 4, 5, 6\}$.

② Дадени са ви n подаръка и масив $A[1..n]$ с цените им т.е. $A[i] > 0$ цена на i-ти подарък (нищо не е безплатно).
Имате S лв.
Искаме да купим макс. брой подаръци с тези S лв.

③ Предложете алгоритъм за сливане на k сортирани масива с общо n елемента със сложност по време $O(n.\lg k)$

④ Няколко души (n на брой) се канят да ви гостуват. Тази информация е представена в масив $A[1...n]$ от цели положителни числа (може да са много големи):
$A[i]=k$ означава, че i-тият гост ще дойде след k дни ($k=1$ - утре итн.)
Предложете алгоритъм с максимална времева сложност $O(n)$ за намиране на първия от предстоящите дни, през който няма да имате гост.

Hint: Може да минете и с $O(\log n)$ допълнително памет.

За останалите задачи има линкове.

Зад 1) Използване CountingSort.
Т.к. 6-2=4, то дъже масива е константна (зависи от големината на n вече, ако е много голяма, то броят битове, които ни трябват, за да съхраняваме число се увеличава).

```
alg(A[1..n]: array of nums)
  B[2..6]: array of ints
O(1) { for i ← 1 to 4 do
         B[i] ← 0
O(n) { for i ← 1 to n do
         B[A[i]] ← B[A[i]] + 1
O(1) { i ← 1
O(n) { for j ← 2 to 6 do
         while B[j] > 0 do
           A[i] ← j
           B[j] ← B[j] - 1
           i ← i+1
```

$T(n) \asymp n$.

390 (2)

Вариант 1: Brute force с пълно изчерпване т.е. всички подмножества генерираме и най-доброто взимаме (с макс. цена).
$T(n) \asymp 2^n$

Вариант 2: Чрез сортиране и линейно обхождане като добавяме елементи докато не сме надвишили цената (т.е. взимаме всички най-малки от началото докато не надвишим лимита).

Вариант 3: Ще използваме binary search, PICK и partition.
Идеята е следната:

① Намираме median с PICK(A[1..n])

② partition(A[1..n], median) вече в A[1,...,n/2] са евтините и скъпите са в A[n/2+1,...,n].

③ Смятаме $\sum_{i=1}^{n/2} A[i] = S$.

④ Ако $S=M$, то купуваме евтините $A[1,\dots,n/2]$ и приключваме.

⑤ Ако $S>M$, то минаваме към стъпка ① с масив $A[1,\dots,n/2]$, със сума M (търсим още по-евтини)

⑥ Ако $S<M$, то минаваме към стъпка ① с масив $A[n/2,\dots,n]$, вече с $\underbrace{M-S}$

купуваме $A[1,\dots,n/2]$, но искаме още подаръци.

Имаме бюджет.

Да видим $T(n)$?

$$T(n) \asymp \underbrace{n}_{①} + \underbrace{n}_{②} + \underbrace{n}_{③} + \underbrace{T(n/2)}_{⑤ \text{ и } ⑥} \asymp$$

в едно от двете влизаме

$$T(n/2) + O(n)$$

$k = \log_2 1 = 0 \to n^k = 1$, $f(n) = n^1$

$n^{k+0,1} \prec n^1$ за $\varepsilon = 0,1$

По МТ знаем $\underline{T(n) \asymp n}$

322 (3) Може да използваме MinHeap.
Алгоритъмът протича така:
1. Инициализираме MinHeap като от всеки от k-те масива се взима първия елемент и се слага в MinHeap.
2. Стъпка „pop" (ExtractMin) на върха → взимаме най-малкия елемент.
Гледаме от кой от оригиналните k сме го взели (за целта може в MinHeap да държиме <елемент, името масив> и да сортираме по I-вата компонента) и взимаме друг от елемент от този масив и го добавяме в MinHeap.
Стъпка 2. се повтаря докато не свършат елементите от масивите и MinHeap-а.
На всяка стъпка pop-натия елемент го слагаме в нов масив.
Сложността е $O(n.\log k)$, защото n пъти правим Heapify, което е със сложност $O(\log k)$.

Зад.4) Тази задача я решавахме в час: Iвото липсващо число в масива. Има няколко варианта в Seminar9 разписани решения. Допълнителната памет идва от брояча i (ако n е много голямо, то ни трябват logn бита, за да го съхраняваме). Реално го пренебрегваме този факт, когато решаваме задачи като приемаме, че произволно големи числа можем да съхраняваме в една клетка и константни операции за обработката им. Ако видите такова намекване и не виждате откъде е този logn, то не забравяйте броячите!