ДАА семинар 9

Зад. Да се състави алгоритъм, който по даден масив $A[1..n]$ и число T преброява максималният брой елементи, чиято сума не надхвърля това T. Намерете сложността му по време.

Реш:

I н. Сортиране + лин. обхождане

$$T(n) = \Theta(n\lg n) + \Theta(n) = \Theta(n\lg n)$$

II н.

```
MaxSumT(A[1..n], T)
    if n ≤ 0
        return 0
    if n = 1
        return A[1] < T ? 1 : 0
    median ← Select(A, ⌊(n+1)/2⌋)   //Намираме медианата

    Partition(A, m)   //Прехвърляме „малките" отляво
    S ← 0
    for i ← 1 to ⌊(n+1)/2⌋
        s ← s + A[i]
    if s = T
        return ⌊(n+1)/2⌋
    if s > T
        return MaxSumT(A[1...⌊(n+1)/2⌋-1], T)   // търсим измежду „малките"
    if s < T
        return MaxSumT(A[⌊(n+1)/2⌋+1...n], T-S)   // с колко от „големите" трябва да допълним сумата
```

Коректността се доказва чрез силна индукция. А Select и Partition са готово коректни от лекции.

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

$k = \log_2 1 = 0$ Сравн. n^0 с $\Theta(n)$ $n^{0+\varepsilon} \leq \Theta(n)$

$\Rightarrow$ III сл. МТ $T(n) = \Theta(n)$

Зад. Даден е масив $A[1..n]$ за $n \geq 2$ от цели числа. Да се предложи алгоритъм, който намира $\max\{A[i] - A[j] \mid 1 \leq i < j \leq n\}$ и да се докаже коректността му.

Решение:

```
AlgMax(A[1..n]: n ≥ 2)
1  max ← max(A[1], A[2])          // Макс. ел-т досега
2  maxDiff ← A[1] - A[2]          // Макс. разлика досега
3  for i ← 3 to n
4     if max - A[i] > maxDiff
5        maxDiff ← max - A[i]
6     if A[i] > max
7        max ← A[i]
8  return maxDiff
```

Инв. При всяко достигане на ред 3:
1) max съдържа максималния елемент на $A[1..i-1]$
2) maxDiff съдържа $\max\{A[p] - A[q] \mid 1 \leq p < q \leq i-1\}$

База: ДР

Поддръжка: I сл. $max - A[i] > maxDiff$

$$max - A[i] \overset{\text{ИТ}}{=} \max\{A[1], \dots, A[i-1]\} - A[i] =$$
$$= \max\{A[1] - A[i], \dots, A[i-1] - A[i]\}$$

На ред 5:

$$maxDiff' = \max\{maxDiff, max - A[i]\} =$$
$$= \max\{\max\{A[p] - A[q] \mid 1 \leq p < q \leq i-1\}, \max\{A[1] - A[i], \dots, A[i-1] - A[i]\}\}$$
$$= \max\{A[p] - A[q] \mid 1 \leq p < q \leq i\}$$

В термините на новото i инв. се запазва.

II сл. ДР

За другия if също разглеждаме 2 случая. ДР

Терминация ДР

Зад. Даден е масив A[1..n], чиито елементи са сортирани и „завъртяни" между 1 и n пъти. Например ако масивът [0, 1, 2, 4, 5, 6, 7] е завъртян 4 пъти, ще се получи [4, 5, 6, 7, 0, 1, 2]. Знаем също, че A е непразен и елем. му са 2 по 2 различни.
Да се напише алгоритъм, който по даден масив A намира минималният му елемент и работи в $O(\lg n)$ време.

Решение: FindMin(A[1..n] : n ≥ 1)

```
left ← 1, right ← n
while left < right
    mid ← ⌊(left + right) / 2⌋

    if A[mid] > A[right]
        left = mid+1
    else
        right = mid
return A[left]   // може и right
```

Не е нужно да имаме конкретна стойност за търсене

// Знаем, че търсеното е надясно

Идеята на алгоритъма е да приближи границите left и right до минималния елемент докато те не съвпаднат.

Доказателството за коректност е подобно на това на двоичното търсене, като трябва да се обоснове защо са верни твърденията.

Сложността е ясна $T(n) = \Theta(\lg n)$

Зад. В даден масив „склон" е максимален по включване непразен подмасив, в който елементите са ненамаляващи. Да се предложи алгоритъм, който намира броя склонове в A[1..n].

Реш.: NumSlopes(A[1..n] : n ≥ 1)

```
1  cnt ← 1
2  for i ← 2 to n
3      if A[i-1] > A[i]      ← започване нов склон
4          cnt ← cnt + 1
5  return cnt
```

Инв.: При вс. дост. на ред 2 cnt съдържа броя на склоновете в A[1.. i-1].