ДАА Семинар 7
Алгоритми върху масиви

Преговор

① Да се напишат определенията на $O, \Omega, \omega, o, \theta$

② Да се сравнят асимптотично следните функции:

а) n^k vs $\left(\frac{3}{2}\right)^n$, $k>0$

Решение: Логаритмуваме и двете страни:

$$\log(n^k) \quad vs \quad \log\left(\left(\tfrac{3}{2}\right)^n\right)$$
$$k\log n \quad vs \quad n\log\left(\tfrac{3}{2}\right)$$
$$\log n \quad vs \quad n$$
$$\log n < n \quad (\square)$$

От $(\square)$ и $\log(f) < \log(g) \to f < g$ (за f, g - растящи и неограничени) $\Rightarrow n^k < \left(\frac{3}{2}\right)^n$

$\frac{3}{2} > 1 \Rightarrow \left(\frac{3}{2}\right)^n$ е растяща // ако беше $\left(\frac{2}{3}\right)^n$ не може да логаритмуване

б) $n^k a^n$ vs b^n, $a \geq 1, b > 1$ и $b > a$

Решение:

$$\lim_{n\to\infty} \frac{n^k a^n}{b^n} = \lim_{n\to\infty} n^k \cdot \left(\frac{a}{b}\right)^n = \lim_{n\to\infty} \frac{n^k}{\left(\frac{b}{a}\right)^n} \overset{a)}{=} 0 \Rightarrow n^k a^n < b^n$$

③ Да се определи асимптотичната сложност по време на:

а) AlgX ($n = 2^{2^k}$)

```
1. count ← 0
2. for i ← 1 to n
3.      j ← 2
4.      while j ≤ n
5.           j ← j^2
6.           count ← count+1
7. return count
```

Решение: За всяко изпълнение на for цикъла на ред 2, j заема последователно стойностите: $j = 2, 2^2, [2^2]^2, [[2^2]^2]^2, \ldots, 2^{2^k} = n$, тоест $j = 2, 2^2, 2^4, 2^8, \ldots, 2^{2^k} = n$ или $j = 2^{2^0}, 2^{2^1}, 2^{2^2}, 2^{2^3}, \ldots, 2^{2^k}$, тоест ред 5 се изпълнява точно k пъти, където $n = 2^{2^k}$. Тоест, $k = \log\log n$

// $n = 2^{2^k} \; / . \log_2 \Rightarrow \log_2 n = \log_2 2^{2^k} \Leftrightarrow \log_2 n = 2^k \log_2 2 \Leftrightarrow \log_2 n = 2^k \; / \log_2 \Rightarrow \log_2\log_2 n = \log_2 2^k$
$\Leftrightarrow \log\log n = k$

Следователно, общата сложност е $T(n) \asymp n\log\log n$.

б) AlgY

```
1. count ← 0
2. for i ← 1 to n^2
3.     for j ← n downto 2
4.         j ← √j
5.         count ← count + 1
6.     i ← 2.i
7. return count
```

Решение: Първо ще проверим колко пъти се изпълнява външния цикъл:
Ред 3 се изпълнява за всички валидни стойности на i, тоест за $i = 1, 2, 4, 8, 16, \dots, n^2$ или $i = 2^0, 2^1, 2^2, 2^3, 2^4, \dots, 2^k = n^2$. Валидните стойности на i са $k+1$ на брой, където $2^k = n^2 \;/\log \Rightarrow \log 2^k = \log n^2 \Leftrightarrow k\log 2 = 2\log n \Leftrightarrow k = 2\log n$. Тоест, валидните стойности на i са $2\log n + 1$ на брой.

Ще проверим колко пъти се изпълнява вътрешния цикъл за една конкретна стойност на i:
Аналогично, ред 5 се изпълнява за $j \in \{n, \sqrt{n}, \sqrt{\sqrt{n}}, \sqrt{\sqrt{\sqrt{n}}}, \dots, 2\}$. Тоест, $j = n, \sqrt{n}, \sqrt{\sqrt{n}}, \sqrt{\sqrt{\sqrt{n}}}, \dots, 2$ или $j = n^{\frac{1}{2^0}}, n^{\frac{1}{2^1}}, n^{\frac{1}{2^2}}, n^{\frac{1}{2^3}}, \dots, 2 = n^{\frac{1}{2^t}}$. Валидните стойности на j са $t+1$ на брой, където $2 = n^{\frac{1}{2^t}} \;/\log \Rightarrow$
$\log 2 = \log n^{\frac{1}{2^t}} \Leftrightarrow 1 = \frac{1}{2^t}\log n \;/\cdot 2^t \Leftrightarrow 2^t = \log n \;/\log \Rightarrow \log 2^t = \log\log n$
$\Rightarrow t\log 2 = \log\log n \Rightarrow t = \log\log n$.

Получихме, че за всяко изпълнение на външния цикъл, вътрешният цикъл се изпълнява $\log\log n + 1$ пъти. Асимптотиката задава ред 5, който се изпълнява $\asymp (2\log n + 1)(\log\log n) \asymp \log n * \log\log n$

4) Да се решат следните рекурентни уравнения

а) $S(n) = 4S(\frac{n}{2}) + n^2(\lg n)^3$

Решение:

Използваме МТ:

$a = 4$
$b = 2$
$k = \log_b a = \log_2 4 = 2$
$n^k = n^2$
$f(n) = n^2(\lg n)^3$

→ подсказва ни да опитаме с разширението на 2-ри)

Сравняваме n^2 с $n^2[\lg n]^3$

Разширението гласи, че ако $n^k \log^{t \in N^+}(n) \asymp f(n)$, то $S(n) \asymp n^k \log^{t+1}(n)$

В нашия случай е изпълнено, че $n^2 \log^3(n) \asymp n^2 [\log(n)]^3 \overset{2(a)}{\Rightarrow} S(n) \asymp n^2 \log^4(n)$

б) $T(n) = 4T(\frac{n}{2}) + n^{\lg n}$

Решение:

Използваме МТ:

$a := 4$

$b := 2$

$k := \log_b a = \log_2 4 = 2$

$n^k = n^2$

$f(n) = n^{\lg n}$

Сравняваме n^2 с $n^{\lg n}$ // $\lg n$ е растяща ф-я, а 2 - константа $\Rightarrow$ можем да вземем каквато си искаме константа $\varepsilon > 0$, за което да е в сила:

$n^{2+\varepsilon} \leq n^{\lg n}$, например $\varepsilon = 1$

Попадаме в сл 3) на МТ: Проверяваме условието за регулярност:

търсим $c \in (0,1)$, за което: $af(\frac{n}{b}) \leq cf(n)$, за $n \uparrow\uparrow$

$$4.\left(\frac{n}{2}\right)^{\lg \frac{n}{2}} \overset{?}{\leq} c.n^{\lg n} \iff \cancel{2n^{\lg \frac{n}{2}} \overset{?}{\leq} cn^{\lg n}}$$

$$4.\frac{n^{\lg \frac{n}{2}}}{2^{\lg \frac{n}{2}}} \overset{?}{\leq} cn^{\lg n} \iff 4.\frac{n^{\lg n - \lg 2}}{2^{\lg n - \lg 2}} = 4.\frac{n^{\lg n - 1}}{2^{\lg n - 1}} = 4\frac{\frac{n^{\lg n}}{n}}{\frac{2^{\lg n}}{2}} \overset{?}{\leq} cn^{\lg n}$$

$$\iff 4.\frac{n^{\lg n}}{n}.\frac{2}{n} \overset{?}{\leq} cn^{\lg n} \iff \frac{8}{n^2}.n^{\lg n} \leq cn^{\lg n}$$, последното е в сила, например за $c = \frac{8}{9}$ и $n \geq 3$.

$\overset{3сл) МТ}{\Rightarrow} f(n) \asymp n^{\lg n}$

в) $H(n) = 29H(\frac{n}{3}) + \binom{2n}{2}$

Решение:

Използваме МТ:

$a := 29$

$b := 3$

$k := \log_b a = \log_3 29$, където $3 < k < 4$ // $3^3 = 27 < 29$

n^k

$f(n) = \binom{2n}{2} \asymp (2n)^2 = 4n^2 \asymp n^2$

Сравняваме n^k с n^2 // От $k > 3$ следва, че $\exists \varepsilon > 0 : k - \varepsilon > 3 \rightarrow k - \varepsilon > 2$

$n^{k-\varepsilon} \succsim n^2$, $\forall \varepsilon \in (0, k-2) \overset{1сл) МТ}{\Rightarrow} H(n) \asymp n^k = n^{\log_3 29}$

г) $F(n) = 29F(\frac{n}{3}) + n^{\sqrt{n}} + (\sqrt{n})^n$

Решение: Ще използваме МТ, но първо ще опростим нехомогенната част

$f(n) = n^{\sqrt{n}} + (\sqrt{n})^n$

$f(n) = n^{\sqrt{n}} + (\sqrt{n})^n$

Сравняване $n^{\sqrt{n}}$ с $(\sqrt{n})^n$ $/\log$

$\log n^{\sqrt{n}}$ vs $\log \sqrt{n}^n$

$\sqrt{n}\log n$ vs $n\log\sqrt{n}$

$\lim_{n\to\infty} \frac{\sqrt{n}\log n}{n\log\sqrt{n}} = \lim_{n\to\infty} \frac{\log n}{\frac{1}{2}\sqrt{n}\log n} = 0 \Rightarrow \sqrt{n}\log n < n\log\sqrt{n} \Rightarrow n^{\sqrt{n}} < (\sqrt{n})^n$

$\Rightarrow f(n) \asymp (\sqrt{n})^n$

$a := 29$

$b := 3$

$k := \log_3 29$, $0 < k < 4$

$f(n) \asymp (\sqrt{n})^n$

Сравняване n^k с $(\sqrt{n})^n = n^{\frac{n}{2}}$, $\frac{n}{2}$ е неогр. растяща ф-я, а k - константа

$\Rightarrow$ може да вземем която си искаме константа $\varepsilon > 0$, за да е изпълнено

$$n^{k+\varepsilon} \le n^{\frac{n}{2}}$$

Попадане в 3-ти сл. Проверяваме условието за регулярност:

$$af\left(\frac{n}{b}\right) \overset{?}{\le} cf(n)$$

$$29\left(\sqrt{\frac{n}{3}}\right)^{\frac{n}{3}} \overset{?}{\le} c.\sqrt{n}^n \iff \frac{29}{3^{\frac{n}{6}}} . n^{\frac{n}{6}} \overset{?}{\le} c n^{\frac{n}{2}} \quad /:n^{\frac{n}{6}}$$

$\iff \frac{29}{3^{\frac{n}{6}}} \overset{?}{\le} c.n^{\frac{n}{3}}$, свидетели, например, са $c = \frac{29}{3.36}$ и $n_0 = 6$

// може да се каже, че очевидно съществува такова c в случая

9) $K(n) = 8K(n-1) - K(n-2) + 2n2^{2n} + 3n2^{3n}$

Решение: Чрез характеристично уравнение:

- Хомогенна част

$K(n) = 8K(n-1) - K(n-2)$

Характеристично:

$x^n = 8x^{n-1} - x^{n-2} \quad /:x^{n-2}$

$x^2 = 8x - 1$

$x^2 - 8x + 1 = 0$

$D = k^2 - ac = 16 - 1 = 15$

$x_{1,2} = \frac{-k \pm \sqrt{D}}{a} = 4 \pm \sqrt{15} \rightarrow \{4-\sqrt{15}, 4+\sqrt{15}\}_M$

- Нехомогенна част:

$2n2^{2n} + 3n2^{3n} = $ /привеждане в подходящ вид/ $= 2n4^n + 3n8^n \rightarrow \{4, 4, 8, 8\}_M$

Общо мултимножество: $\{4-\sqrt{15},\ 4+\sqrt{15},\ 4, 4, 8, 9\}_м$.

Търсим най-големия корен: $4+\sqrt{15} < 4+\sqrt{16} = 8$ => Най-големият корен е 8 =>

$K(n) \asymp n8^n$

Алгоритми върху масиви

зад 1 Нека $a_1a_2\ldots a_n$ е пермутация на мн-вото $\{1,2,\ldots,n\}$. Инверсия в тази пермутация се нарича всяка наредена двойка (i,j), такава се $1 \le i < j \le n$ и $a_i > a_j$. Инверсният вектор на пермутацията $a_1a_2\ldots a_n$ е векторът $(b_1, b_2, \ldots, b_n)$, където $\forall i; 1 \le i \le n$: b_i е броят на елементите в $a_1\ldots a_n$, които са вляво от i и са по-големи от i.

а) Напишете инверсния вектор на пермутацията 4 2 3 7 1 8 5 9 6

б) Да се конструира алгоритъм, който, по валиден инверсен вектор $(b_1, \ldots, b_n)$, извежда оригиналната пермутация на $\{1, \ldots, n\}$.

Иска се кратка обосновка + анализ на сложността по време и памет.

Решение:

а) Инверсният вектор е 4 1 1 0 2 3 0 0 0

б) Разсъждаваме така: „Намираме" последователно правилните места на елементи $1, 2, \ldots, n$ в този ред. Последното правим по следния начин:

За елемент '1' има точно един правилен индекс и той е b_1+1, тъй като всички елементи са по-големи от него и преди него трябва да има b_1 по-големи елемента - поставяме 1 на позиция b_1+1. Разглеждаме елемента 2 - всички елементи от все още непоставените са по-големи от 2, следователно поставяме 2 на позиция, преди която има точно b_2 свободни места. Аналогично, същото правим и за останалите елементи - по този начин еднозначно определяме оригиналния вектор.

Ето и псевдокода:

```
Alg (B[1...n] - коректен инверсен вектор)
1. A[1...n] ← 0
2. for i ← 1 to n
3.     count ← 0
4.     pos ← 1
5.     while count < b_i || (count = b_i && !A[pos] = 0)
6.         if A[pos] = 0
7.             count ← count + 1
8.         pos ← pos + 1
9.     A[pos] ← i
```

Примерна инварианта за външния цикъл:

При всяко достигане на ред 2 елементите от 1 до $i-1$ са на правилните си позиции в масива $A[1..n]$ (тоест, $\forall j \in \{1,...,i-1\}$: j е на такава позиция, че преди него има b_j клетки, в които или са празни, или съдържат някой от елементите от $\{j+1,...,i-1\}$) и в $A[1..n]$ има точно $n-i+1$ празни клетки

Д-во на инвариантата:

1) База: Разглеждаме първото достигане на ред 2.
В този момент $i=1$, а/масив елементите от 1 до $i-1$ са 0 на брой, а в този момен $A[1..n] = [0..0]$ има 0 на брой елементи на правилни позиции и има точно $n-1+1 = n$ празни клетки ✓

2) Поддръжка
Допускаме, че за някое достигане на ред 2, което не е последното, твърдението е изпълнено. Щом достигането не е последното, следва навлизане в цикъла. Ще разгледаме работата на while цикъла в текущото изпълнение на for-а.

Инварианта за вътрешния цикъл:

При всяко достигане на ред 5 е в сила:
count съдържа броя свободни клетки в масива $A[1..pos-1]$ и $pos \le n$

Д-во на инвариантата:

1) База: При първото достигане на ред 5 е изпълнено:
$count = 0 \wedge pos = 1 \wedge A[1..pos-1] = A[1..0] = []$. Броят на свободните клетки в празния масив е 0 и $1 \le n$ ✓

2) Поддръжка: Допускаме, че инвариантата е изпълнена за някое достигане на ред 5, което не е последното. Следва навлизане в цикъла:
1(а) $A[pos] = 0$ е истина. При което $count' \leftarrow count+1$ е новата стойност на count и $pos' \leftarrow pos+1$ е новата ст-ст на pos. При следващото достигане на ред 5 е в сила, че $count'$ е броят на свободните клетки в $A[1..pos'-1]$ // от допускането count = брой свободни в $A[1..pos-1]$, но $A[pos]$ е свободна $\Rightarrow count+1 = count'$ е броят свободни клетки в $A[1...pos] = A[1...pos'-1]$

Дали $pos' \le n$? От допускането имаме, че $pos = pos'-1 \le n$. (Δ)
Тъй като сме навлезли в цикъла, то е в сила:

$count < b_i \vee (count == b_i \wedge A[pos]$ не е свободна$)$ (*)

Допускаме, че $pos' = pos+1 > n$. От (Δ) следва, че $pos = n$.
1(а) $count < b_i$ е вярно, при което в $A[1...pos] = A[1...n]$ не може да се постави елемент a_i, така че преди него да има b_i свободни

клетки, което е противоречие с валидността на инверсния вектор

2сл) $count == b_i$ & $A[pos]$ не е свободна е в сила. Аналогично, противоречие с валидността на инверсния вектор.

$\Rightarrow pos' \le n$

2сл) $A[pos]$ е заета. Тогава броят на свободните клетки (count) в $A[1..pos-1]$ е равен на броя свободни клетки в $A[1..pos]$. В този случай count не променя стойността си: $count' = count$. $pos' = pos+1$ и при следващо достигане е в сила, че $count'$ е броят свободни клетки в $A[1..pos'-1]$ и, отново $pos' \le n$

3) Терминация. При последното достигане на ред 5 $count = b_i$ && $A[pos]$ е свободна. Тогава, на ред 9 $A[pos]$ заема стойност i

От допускането, че елементите от $1..i-1$ са на правилните си позиции в масива и от факта, че ел-тите се слагат последователно, следва, че в свободните b_i клетки преди $A[pos] = i$ ще се разположат елементи, по-големи от i. Следователно, $1, .., i$ са на правилните си места и $A[1..n]$ има $(n-i+1)-1 = n-i$ свободни клетки

3) Терминация: При последното достигане на ред 2: $i = n+1$ и спрямо инвариантата: елементите от $1, .., n+1-1$ са на правилните си места и в $A[1..n]$ има $n-(n+1)+1 = 0$ свободни клетки.

- Сложност по памет: $M(n) \asymp n$ заради резултатния масив
- Сложност по време: $T(n) \asymp n + n^2 \asymp n^2$

Зад. 2 Липсващо число

Дадено $N \in \mathbb{N}^+$ и масив $A[1...N-1]$, съдържащ уникални числа $\in [1...N]$

// тоест $A[1..N-1]$ съдържа всички числа от 1 до N с изключение на едно липсващо //

Да се посочи липсващото число.

// Пример $N=5$
$A = [1, 2, 4, 5]$
Липсващо: 3

Решение:

1н) Brute force

Последователно проверяване наличността на всяко от числата в $[1..N]$ с линейно обхождане

// Пример: $A = [1, 2, 4, 5]$

1 ✓

2 ✓

3 ✗ → връщаме 3

Псевдокод:

```
Brute(N, A[1...N-1])
1. for i ← 1 to N
2.     bool found ← false
3.     for j ← 1 to N-1
4.         if A[j] = i
5.             found ← true
6.             break;
7.     if found = false
8.         return i
```

→ $T(n) \asymp n^2$

↳ В най-лошия случай ред 4 се изпълнява $N-1 + N-2 + \dots + 1 = \frac{(N-1)N}{2} \asymp N^2$

↳ ако, например масивът е сортиран в низходящ ред.

// $A = [\cancel{9}, 8, 7, 6, 5, 4, 3, 2, 1]$, $N = 9$

$M(n) \asymp 1$

↳ „без" допълнителна памет

2н) Подобрение

Инициализиране помощен масив $B[1..N]$. Линейно обхождане $A[1...N-1]$, маркиране наличните елементи: $B[A[i]] \leftarrow 1$. Линейно обхождане B и намиране празната клетка, чийто индекс връщаме

// Пример: $A = [1, 3, 5, 4, 6]$

$B = [1, 0, 1, 1, 1, 1]$ → липсва ел. 2
(индекси 1 2 3 4 5 6)

Псевдокод:

```
Better(N, A[1..N-1])
1. B[1..N] ← 0
2. for i ← 1 to N-1
3.     B[A[i]] ← 1
4. for i ← 1 to N
5.     if B[i] = 0
6.         return i
```

→ $T(n) \asymp \Theta(1) + N + N+1 \asymp N$

$M(n) \asymp N$

↳ заради допълнителната памет $B[1..N]$

3н) Оптимално

Знаем, че $\sum_{i=1}^{N} i = \frac{N(N+1)}{2}$ (Δ), още знаем, че в $A[1..N-1]$ липсва точно едно от числата в $[1..N]$ и всички други се срещат точно веднъж. Нека липсващото е x. Тогава $\sum_{i=1}^{N-1} A[i] = \frac{N(N+1)}{2} - x$ $(\square)$. Тогава е в сила:

$x = (\Delta) - (\square)$

Псевдокод:

```
Optimal(N, A[1..N-1])
1. sum ← N(N+1)/2 ; mySum ← 0
2. for i ← 1 to N-1
3.     mySum ← mySum + A[i]
4. return sum - mySum
```

→ $T(n) \asymp n$

$M(n) \asymp 1$

4н) Оптимално → малко по-добро откъм памет спрямо 3н)

С операцията XOR – за всеки два бита XOR връща 1, ако са различни и 0, ако съвпадат.

Изобщо, XOR връща 1, ако броят на 1ците е нечетен.

//Пример:

$$\begin{array}{rl} & 00000111 \quad // 7 \\ XOR & \underline{00000101} \quad // 5 \\ & 00000010 \quad // 2 \end{array}$$

Нека $a \in N$, тогава $a \wedge a$ (a xor a) $= 0$

Операцията XOR е асоциативна и дистрибутивна – няма значение в какъв ред и в каква последователност я прилагаме.

Тогава, нека k е липсващото число в A[1...N-1]. В сила е:

$$\begin{array}{l} 1\text{^}2\text{^}\ldots\text{^}k-1\text{^}k\text{^}k+1\text{^}\ldots\text{^}N \\ \text{^}\ 1\text{^}2\text{^}\ldots\text{^}k-1\text{^}\underbrace{\ }_{\text{липсва } k}k+1\text{^}\ldots\text{^}N \end{array} = \underbrace{(1\text{^}1)}_{0}\text{^}\underbrace{(2\text{^}2)}_{0}\text{^}\ldots\underbrace{(k-1\text{^}k-1)}_{0}\text{^}k\text{^}\underbrace{(k+1\text{^}k+1)}_{0}\text{^}\ldots\text{^}\underbrace{(N\text{^}N)}_{0} = k$$

Псевдокод:

```
Optimal1(N, A[1...N-1])
xor1 ← N
xor2 ← 0
for i ← 1 to N-1
    xor1 ← xor1 ^ i
    xor2 ← xor2 ^ A[i]
return xor1 ^ xor2
```

→ $T(n) \asymp N$

$M(n) \asymp 1$

//Ако в условието е казано, че A[1...N-1] е сортиран, може с двоично търсене – $T(n) \asymp \log n \wedge M(n) \asymp 1$

↳ по индекси

//Пример: $A = [1, 2, 3, 4, 5, 6, 7, 8, 10, 11]$, $N = 11$, $x = 9$

$$\begin{array}{cccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{array}$$

Елементът на позиция i ще е със стойност i+1 за всяко $i \geq x$, за всички останали ($j < x$) ще е в сила: $A[j] = j$

$$\begin{array}{cccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 1, & 2, & 3, & 4, & 5, & 6, & 7, & 8, & 10, & 11 \end{array}$$

↓

$$\begin{array}{cccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 1 & 2 & 3 & 4 & 5 & 6 & [7 & 8 & 10 & 11] \end{array}$$

↓

$$\begin{array}{cccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 1 & 2 & 3 & 4 & 5 & 6 & [7 & 8 & 10] & 11 \end{array}$$

↓

$$\begin{array}{cccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & [10] & 11 \end{array}$$

↳ $i = 9 = x$

зад ③ Брой инверсии в масив

По даден масив $A[1 \dots n]$ да се намери броят инверсии:

// инверсия е всяка двойка (i, j), такава, че $1 \le i < j \le n$ и $A[i] > A[j]$
или може вместо (i, j) да пишем $(A[i], A[j])$

// Пример: $A = [5\ 3\ 2\ 4\ 1]$

Инверсиите са: (5, 3), (5, 2), (5, 4), (5, 1)
(3, 2), (3, 1)
(2, 1)
(4, 1) и са 8 на брой

1н) Brute force

Линейно итериране през всеки елемент и търсим броя инверсии, които образува като ляв елемент

Псевдокод:

```
Brute(A[1...n])
1. count ← 0
2. for i ← 1 to n-1
3.     for j ← 1 to n
4.         if a[i] > a[j]
5.             count ← count + 1
6. return count
```

→ $T(n) \asymp n^2$

$M(n) \asymp 1$

2н) Оптимално - търсим решение около $n\log n$ и n.

// Идея: Нека имаме два сортирани масива // За по-ясно ще ги зададем конкретно:

A_1: [2, 3, 5, 6] A_2: [2, 2, 4, 4, 8]

Питаме се колко инверсии има от вида $(A_1[i], A_2[j])$

Нека дефинираме два указателя - left, right, които да сочат, съответно, към първите елементи на A_1, A_2:

A_1: [2, 3, 5, 6]
↑
left

A_2: [2, 2, 4, 4, 8]
↑
right

Проверяваме дали left > right, ако да, то всички елементи вдясно от left в A_1 ще са по-големи от right, тъй като A_1 е сортиран в етрпоз
Ако не, местим left надясно с една позиция.

A_1: [2, 3, 5, 6] — left, left, left

A_2: [2, 2, 4, 4, 8] — right, right, right

1) 2 ≯ 2. местим left

2) 3 > 2 => вс. елементи в [3, 5, 6] ще образуват инверсия с 2
count ← count + 3. местим right

3) 3 > 2 => ... count ← count + 3. местим right

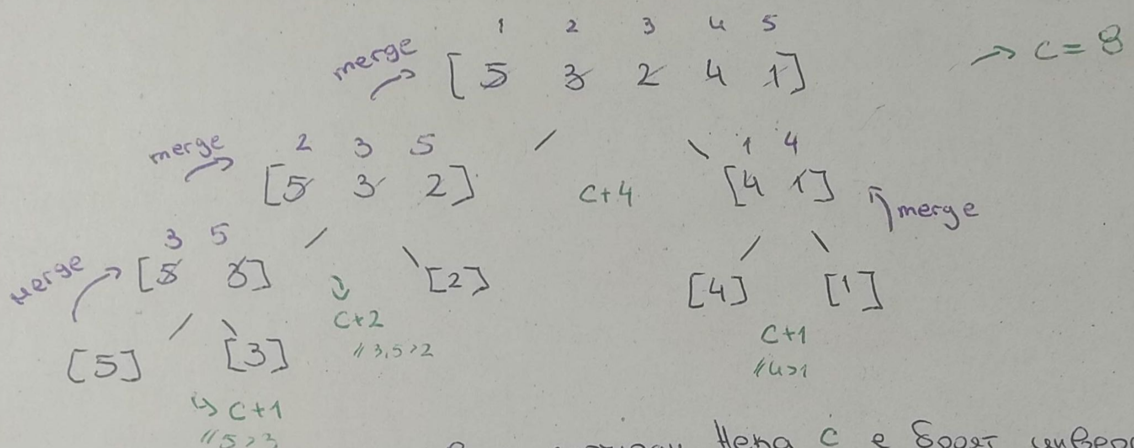
4) 3 ≯ 4. местим left

5) 5 > 2 ... count ← count + 2

Продължавайки така, получаваме count = 10

↓ $T(n) \asymp n$

↳ Ще приложим горната идея за оригиналната задача. Ще използваме рекурсивните викания на Merge Sort



Всеки едноелементен масив е сортиран. Нека c е броят инверсии

Псевдокод:

→ $T(n) \asymp n\lg n$

$M(n) \asymp n$

```
NumOfInversions (A[1...n])
1. count ← 0
2. mergeSort(A[1...n], count)
3. return count
```

Където mergeSort е модифициран, така че merge частта му поддържа counter:

```
merge(arr, low, mid, high, &count)
    ...
while left <= mid && right <= high
    if  arr[left] <= arr[right]
            ...
    else
        ...
        count += mid - left + 1    ->  A[ . ] A[ ... ]
                                      left mid right high
...
```

A[left] > A[right] ∧ A-сорт. => вс. ел-т в A[left...mid] е > A[right], тоест образува инверсия с него, броятиме mid-left+1

↳ Ако НЕ искаме да променяме масива, създаваме копие