

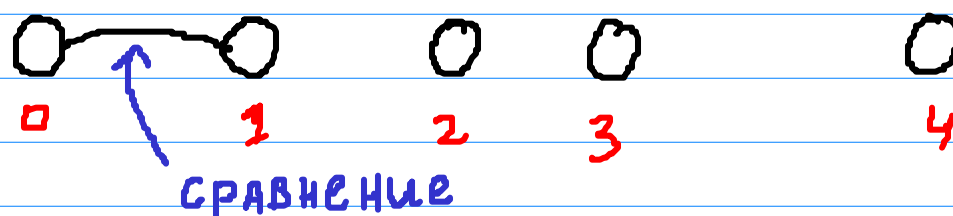
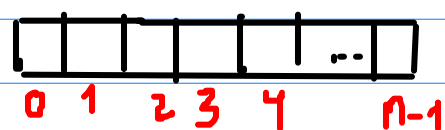
10

- Търсене в сортиран масив $\Omega(\log(n))$
- Търсене на конкретна джекца $\Omega(n \cdot \log(n))$
- Сортиране (дир. сравнения) $\Omega(n \cdot \log(n))$
- Element uniqueness $\Omega(n \cdot \log(n))$
- Търсене на min/max ел. в масив $\Omega(n)$
- Търсене на Moza $\Omega(n \cdot \log(n))$
- Closest Pair $\Omega(n \cdot \log(n))$

зау 1

С колко най-малко сравнения можем да намерим

а) Най-големият ел. в масив?



Графът трябва да е свързан!

$\Rightarrow n-1$ ребра

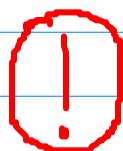
$\Rightarrow n-1$ сравнения

б) Втория най-голям?

Най-бавно: Намираме най-големия $n-1$
(премахваме го) +
Отново търсим най-големия $n-2$

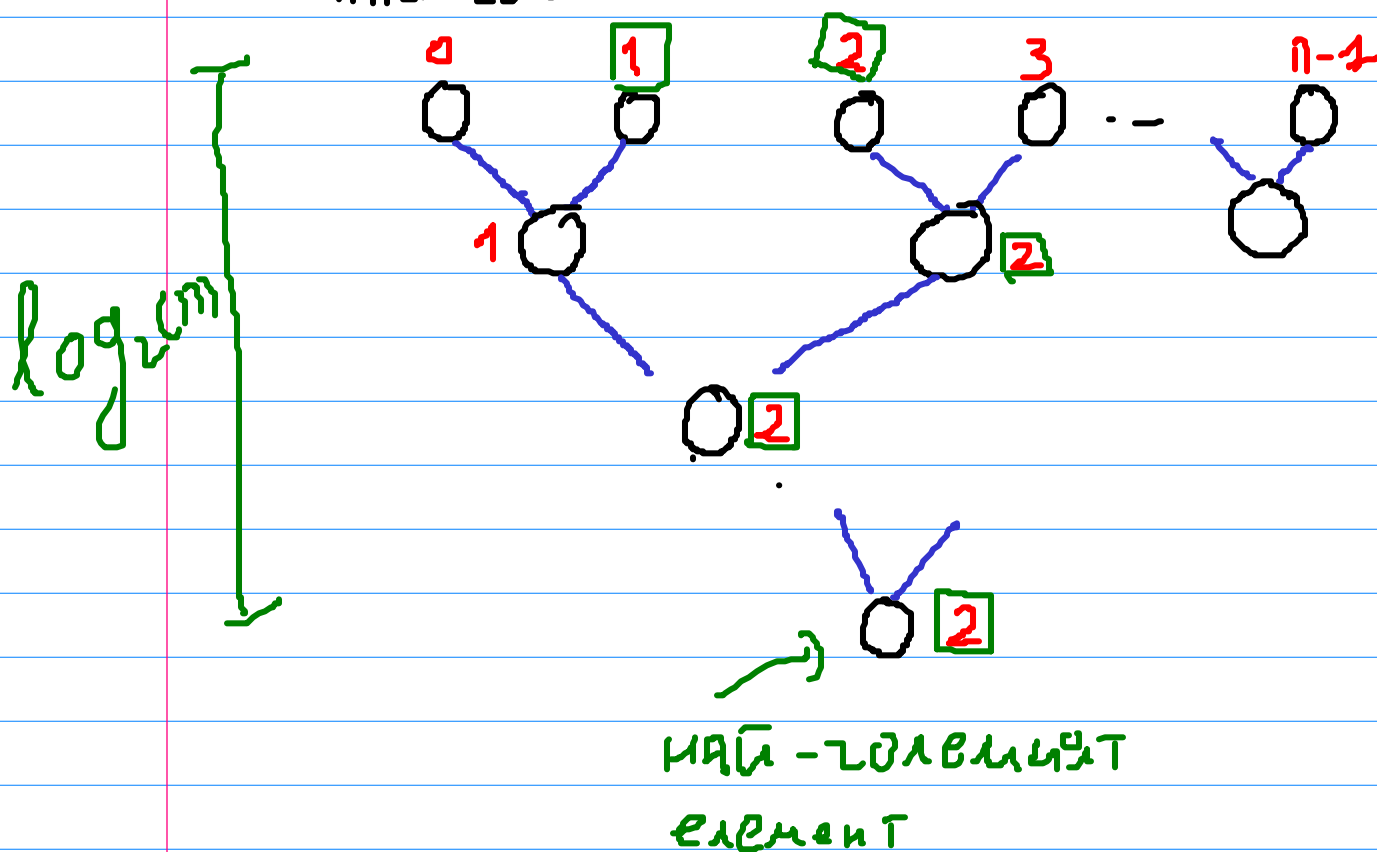
Нека мислим за **турнир**:

- Вторият по-големина е "загубил" само от най-големия



Провеждаме "**мини-турнир**" само от тези, които са "загубили"

от най-големия



$$\Rightarrow \boxed{n-1} + \boxed{\log_2 (n)-1}$$

заг² Дадена е двоична пирамида

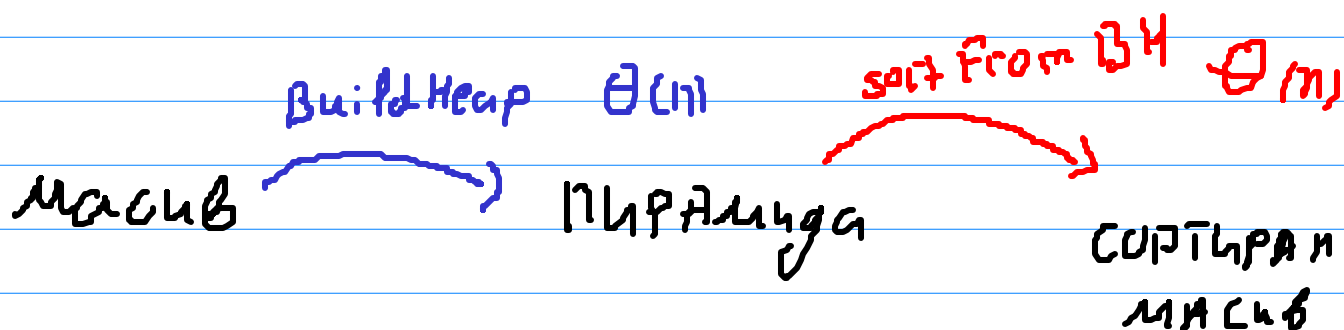
Точкните, че не съществуват алгоритъм, който по дадена двоична пирамида да връща сортиран масив с елементите от пирамидата за $\Theta(n)$

Редукция

- Строене на пирамида: $\Theta(n)$

Допускаме, че \exists такъв АЛ

Нека е **Sort From BH**.



=> Намерихме сортиращ алг. **$\Theta(n)$**

Тякъв показваме, че няма!

Сортиране $O(\text{Sortfrom BN})$

def: P_1, P_2 - задачи

$P_1 \leq P_2$ " P_1 се редуцира до P_2 "

- ① Ако P_1 е "трудна", то и P_2 е "трудна"!
- Ако P_2 е "лесна", то и P_1 е "лесна"

Зад. Докажете, че всеки алг, който връща броя на различните с-ти в масив, работи във време $\Omega(n \cdot \log n)$

Допускаме, че \exists алг **diff Values**, който работи по-бързо (например: $\Theta(n)$)

```
{ El. Uniqueness (arr[1...n])
```

```
    res ← diffValues(arr) //  $\Theta(n)$ 
```

```
    return res == n;    //  $\Theta(1)$ 
```

```
}
```

$\Rightarrow \Theta(n)$.

Но мне нужна граница

За Element uniqueness $\Theta(n \cdot \log(n))$



Element uniqueness. \mathcal{O} diffValues

3^o Докажете, че $\Omega(n)$ за строене на двоична пирамида

Поопускаме, че \exists алг **BuildHeapFast**, който работи по-бързо [например: $\log(n)$]

```
Max(arr[1...n])  
{  
    BuildHeapFast(arr); //  $\log(n)$   
    return arr[1]; //  $\Theta(1)$   
}
```

Сложност на Max: $\Theta(\log(n))$



Max \propto Строене на пирамида

3a) Произволен масив.

Докажете, че всеки алг, който
намира най-често срещаният
елемент работи във време $\Omega(n \log n)$

Допускаме, че същ. алг
mode, който връща най-често
срещания по-бързо (например: $\Theta(n)$)

```
Element uniqueness(арг[1..n]:  $\mathbb{N}$ )
```

```
{  
  x ← mode(арг) //  $\Theta(n)$ 
```

```
  count ← 0
```

```
  for i = 1 to n
```

```
    if (арг[i] == x)
```

```
      count++
```

```
//  $\Theta(n)$ 
```

```
  return count == 1  
}
```


$\Theta(n)$



$\Rightarrow \Omega(n \cdot \log(n))$

Element uniqueness of Mode

~~Заг~~

Масив от точки - $\langle x, y \rangle$

$x \in \mathbb{N}, y \in \mathbb{N}$.

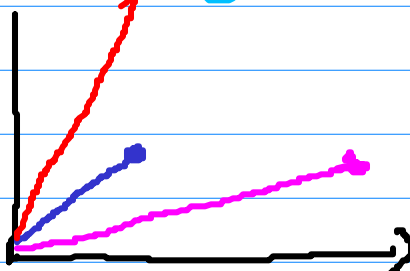
Докажете, че всеки алг., който
сортира точките по възла, който
сключват радиус-векторът с Ox ,
работи във време $\Omega(n \cdot \log(n))$

$[\langle 1, 1 \rangle, \langle 1, 100 \rangle, \langle 100, 1 \rangle]$

②

③

①



Допускаме, че \exists алг **sortPoints**,
която решава задачата по-бързо
(прим: $\Theta(n)$)

Sort [arr[1...n]]

{

arrPoints[1...n]: $\langle x, y \rangle$

for $i \leftarrow 1$ to n

arrPoints[i].x \leftarrow 5

arrPoints[i].y \leftarrow arr[i]

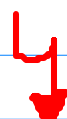
// $\Theta(n)$

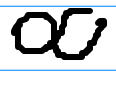
sortPoints (arrPoint); // $\Theta(n)$

for $i \leftarrow 1$ to n

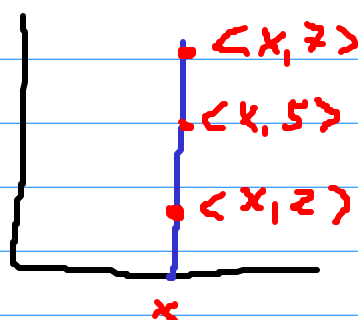
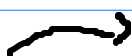
arr[i] \leftarrow arrPoint[i].y // $\Theta(n)$

}



Сортиране  Сортиране
на точки
по p-в.

arr = [5, 2, 7]



SortPoints