Анализ на рекурсивни алгоритми

За да можем да анализираме сложността на рекурсивен алгоритъм е нужно да **съставим** и **решим рекурентно уравнение**

- Какво е **рекурентно уравнение** (reccurence relation)

Определение: Recurrence relation is a function that calls itself.

Пример: Числата на Фибоначи

$$Fibb(n) = \begin{cases} Fibb(n-1) + Fibb(n-2), & n > 1 \\ 1, & 0 \le n \le 1 \end{cases}$$

- Какво означава да реши едно рекурентно уравнение?

Цел: Намиране на затворена формула (без рекурсия) за $T(n)$

- $Fibb(n) = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n$ (Формула на Бине)

Пример:

Да разгледаме следното рекурентно уравнение:

$$T(n) = \begin{cases} T(n-1) + 3, & n \ge 2 \\ 1, & n = 1 \end{cases}$$

при n:	1	2	3	4	5	...
T(n):	1	4	7	10	13	

Достигаме до затворена формула

$T(n) = 3.(n-1) + 1$

- Как решаваме рекурентни уравнения?
  - чрез налучкване + доказателство (обикновено индукция)

- чрез развиване !
- чрез Master theorem или друг метод

- Анализ на рекурсивен linearSearch

$T(n)$: брой стъпки при вход n

$$T(n) = \underbrace{T(n-1)}_{\text{за рекурсивното извикване на linearSearch}} + \underbrace{\Theta(1)}_{\text{за проверките}}$$

интересуване се как расте $T(n)$

Решаваме рекурентното уравнение чрез развиване

$T(1) = \Theta(1)$

$T(n) = T(n-1) + \Theta(1)$

$T(n) = \underbrace{T(n-2) + \Theta(1)}_{T(n-1)} + \Theta(1)$

$T(n) = \underbrace{T(n-3) + \Theta(1)}_{T(n-2)} + \Theta(1) + \Theta(1)$

!Наблюдение: Имаме "n-3" и $\underbrace{\Theta(1) + \Theta(1) + \Theta(1)}_{\text{3 пъти}}$

Продължавайки да развиваме уравнението, получаваме:

$$T(n) = T(1) + \underbrace{\Theta(1) + \ldots + \Theta(1)}_{n-1 \text{ пъти}}, \text{ но } T(1) = \Theta(1)$$

$n - (n-1)$

$$T(n) = \underbrace{\Theta(1) + \ldots + \Theta(1)}_{n \text{ пъти}} = \sum_{i=1}^{n} \Theta(1) = \Theta(n)$$

Нека разглеждаме функцията

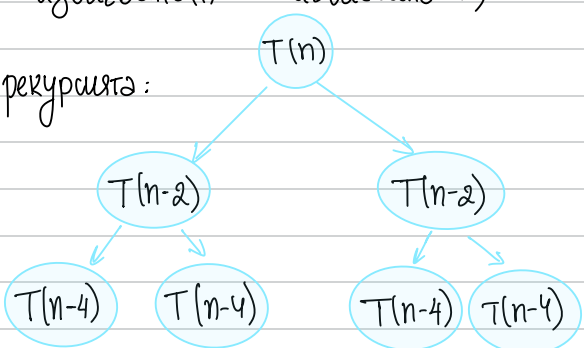
```
int f(int n){
  if (n<=0) return 1;
  return f(n-2)+f(n-2)+5;
}
```

(1) (2)

Дефинираме рекурентно уравнение:

$$T(n) = \underbrace{T(n-2)}_{\text{рекурсивно извикване (1)}} + \underbrace{T(n-2)}_{\text{рекурсивно извикване (2)}} + \Theta(1) = 2T(n-2) + \Theta(1)$$

Дърво на рекурсията:



Колко е дълбочината на дървото?

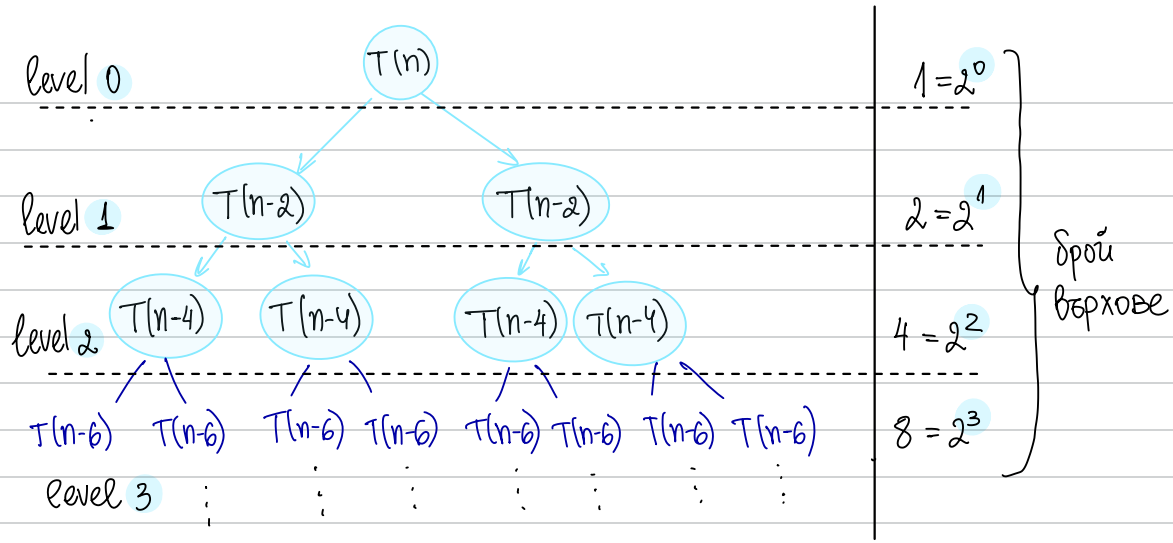
За да намерим колко е дълбочината на дървото дефинираме:

$n - 2 \cdot x \geq 0$

Колко пъти ще извадим 2 от първоначално подадения n

$-2x \geq -n \Rightarrow x \leq \frac{n}{2}$

Следователно дълбочината на дървото е най-много $\frac{n}{2}$



! Интересуваме се от броя върхове !

$$T(n) = 2.T(n-2) + \Theta(1) = 2.(\underbrace{2T(n-4) + \Theta(1)}_{T(n-2)}) + \Theta(1) =$$

$$= 2^2.T(n-4) + (2).\Theta(1) + \Theta(1)$$

← брой пъти за $T(n)$

↖ брой пъти за $T(n-2)$

↑ брой пъти, в които викаме $T(n-4)$

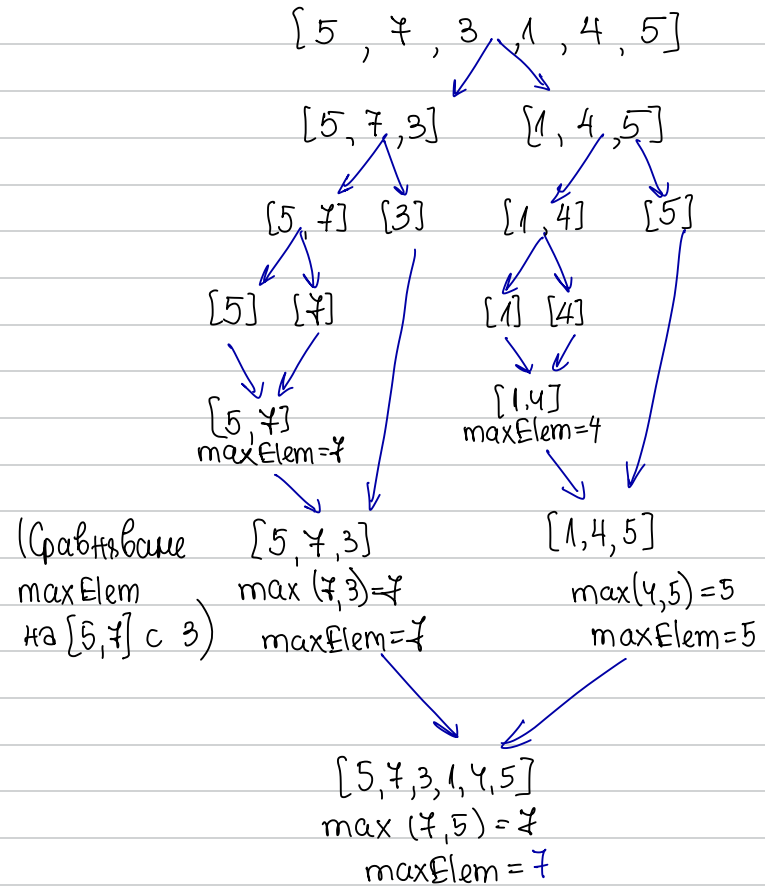
Броят върхове е: $2^0 + 2^1 + \ldots + 2^{\frac{n}{2}} = \sum_{i=0}^{n/2} 2^i = 2^{\frac{n}{2}+1} - 1 = 2^{\frac{n+2}{2}} - 1$

$2^{\frac{n+1}{2}} \in \Theta(2^{\frac{n}{2}})$

Алгоритми, изградени на схемата разделяй и владей

→ разделяме задачата на подзадачи
→ решаваме по-малките задачи
→ акумулираме резултатите

Пример: Търсим максимален елемент в масив. Целим през "средата".



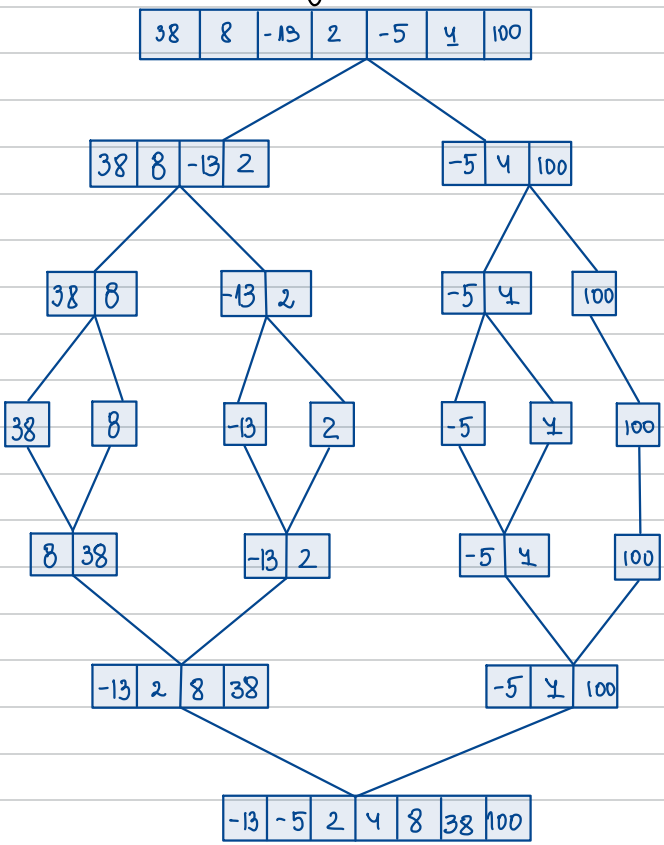
Недостатък: Рекурсивните решения изискват повече памет!

Бързи сортирания (за днес)

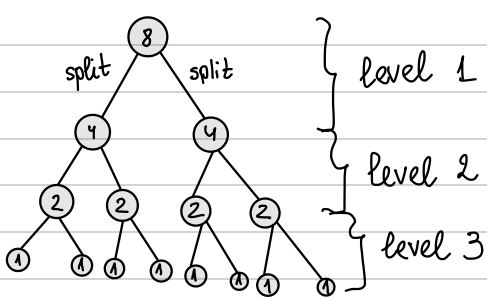
MergeSort QuickSort

Merge Sort

Пример.

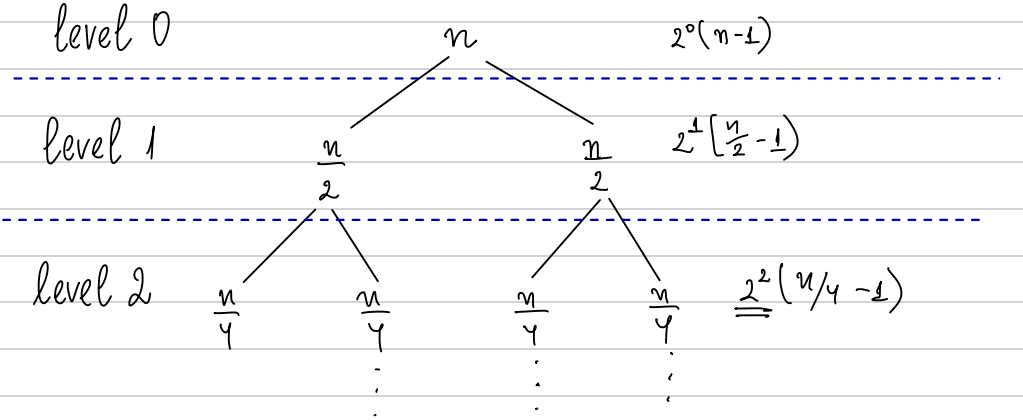


Интуиция за сложността:



Ако масивът има n елемента, то "времето", което ще отнеме да го сортираме е:

$$T(n) = \underbrace{T(\lfloor \tfrac{n}{2} \rfloor)}_{\text{split left}} + \underbrace{T(\lfloor \tfrac{n}{2} \rfloor)}_{\text{split right}} + (n-1) \rightarrow \text{сравнения}$$



$\lfloor \frac{n}{2} \rfloor$ - означа "закръглено надолу"

$\lfloor \frac{5}{2} \rfloor = 2$ (като целочислено деление в C++)

$$T(n) = 2 \cdot T(\lfloor \tfrac{n}{2} \rfloor) + (n-1)$$

$$T(\lfloor \tfrac{n}{2} \rfloor) = 2 \cdot T(\lfloor \tfrac{n}{4} \rfloor) + (\lfloor \tfrac{n}{2} \rfloor + 1)$$

$$2^i \left(\frac{n}{2^i} - 1 \right)$$ → за всеки от тях правим $\frac{n}{2^i} - 1$ сравнения

2^i - колко масива с дължина 2^i имаме

$$\sum_{i=0}^{\log n - 1} 2^i \left(\frac{n}{2^i} - 1 \right) = \sum_{i=0}^{\log n - 1} n - 2^i = \underbrace{\sum_{i=0}^{\log n - 1} n}_{(*)} - \underbrace{\sum_{i=0}^{\log n} 2^i}_{(**)}$$

$$(*)\ \sum_{i=0}^{\log n - 1} n = \underbrace{n + n \ldots + n}_{\log n \text{ пъти}} = n.\log n$$

$$(**)\ \sum_{i=0}^{\log n - 1} 2^i = \underbrace{2^0 + 2^1 + \ldots + 2^{\log_2 n - 1}}_{\text{геометрична прогресия}} = \frac{1.(1-2^{\log_2 n})}{1-2} = 2^{\log_2 n} - 1 = n-1$$

Формула за сума на геометрична прогресия: $\frac{a_1(1-q^n)}{1-q}$

От (*) и (**) получаваме $n.\log n - (n-1) \in \Theta(n.\log n)$