Problema do Subarranjo Máximo

Paulo E. R. Araujo

Problema

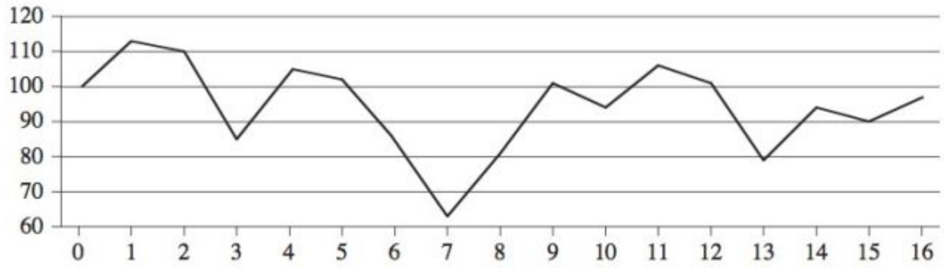
Problema

- Dada uma lista com números negativos e positivos, encontre a sublista contígua com maior soma.
- O problema foi originalmente proposto por Ulf Grenander, da Brown University.

10	5	-17	**20**	**50**	**-1**	**3**	-30	10

Exemplo de uso

- Investimento em Ações
  - Comprar na baixa e vender na alta.



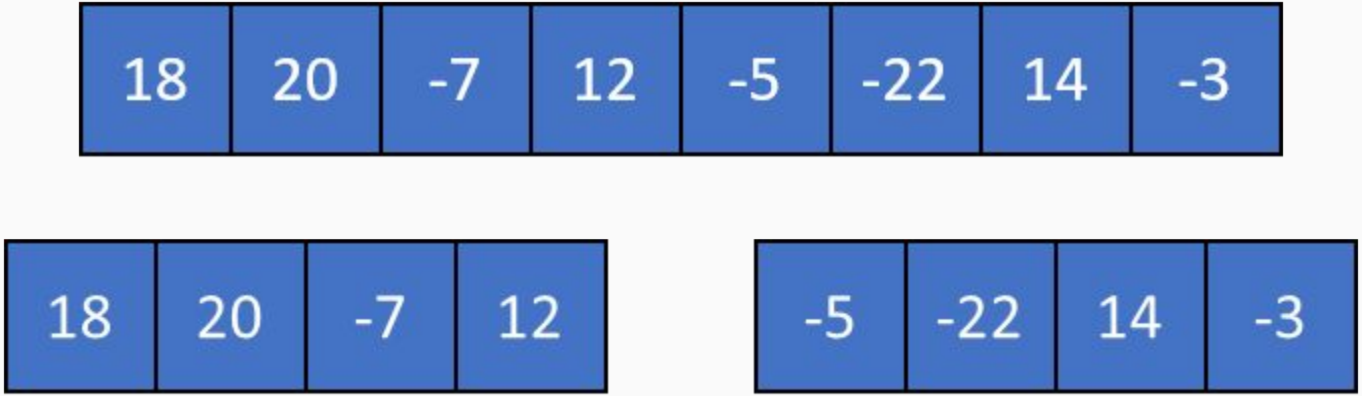
Day	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Price	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
Change		13	−3	−25	20	−3	−16	−23	18	20	−7	12	−5	−22	15	−4	7

Algoritmo de Divisão e Conquista

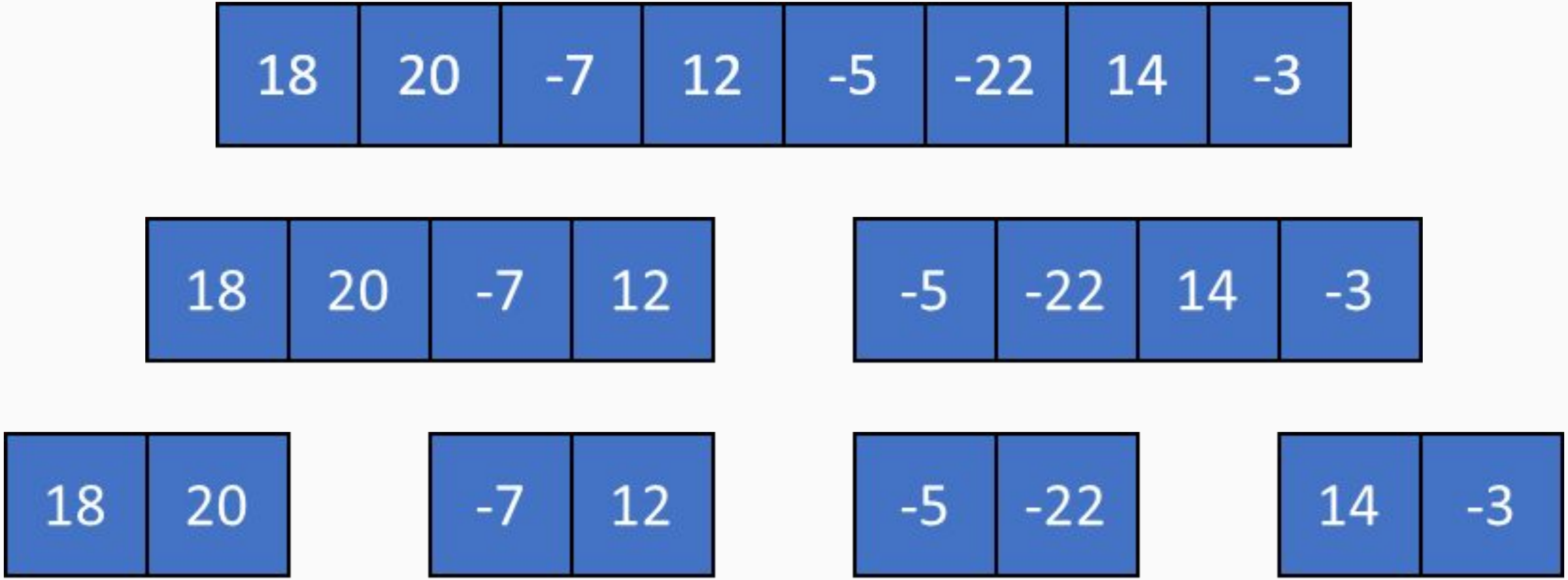
Encontrar a soma do subarranjo máximo

18	20	-7	12	-5	-22	14	-3

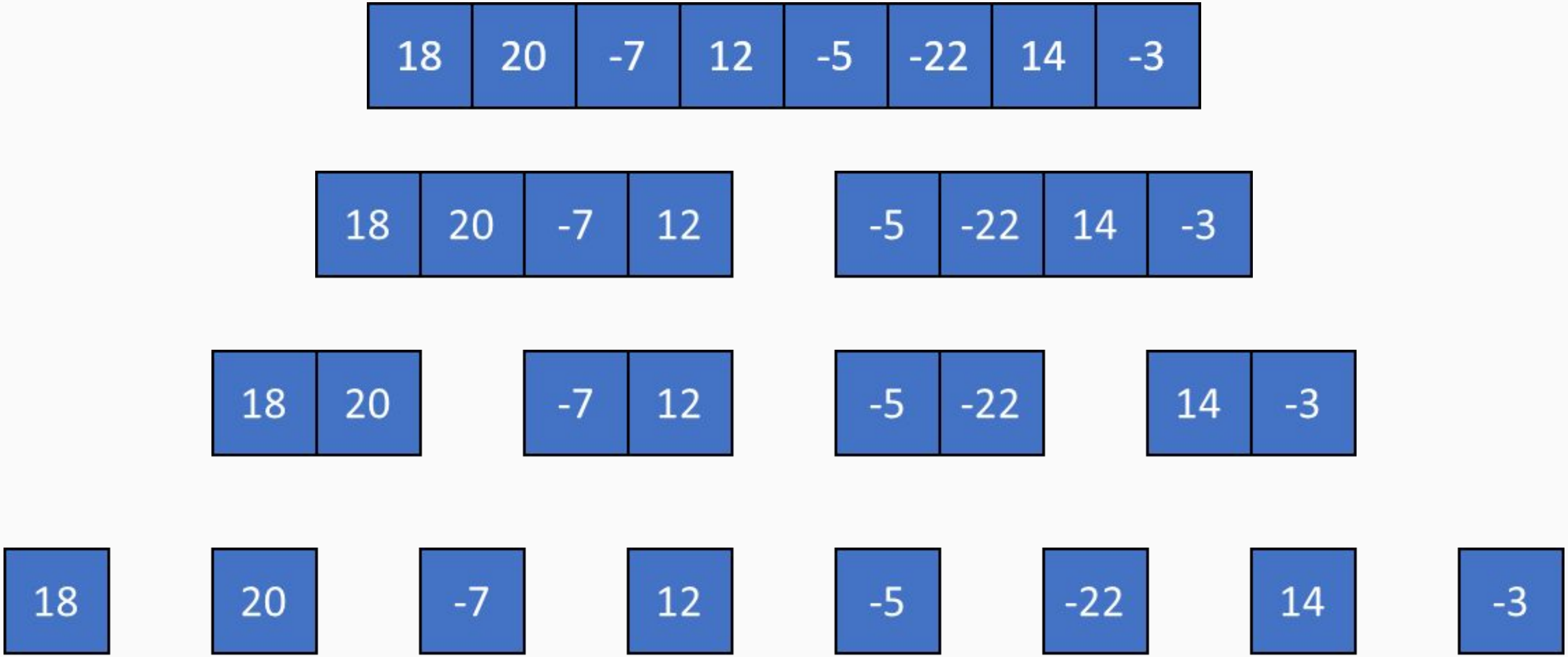
Divisão do problema



01 camada da recursividade.



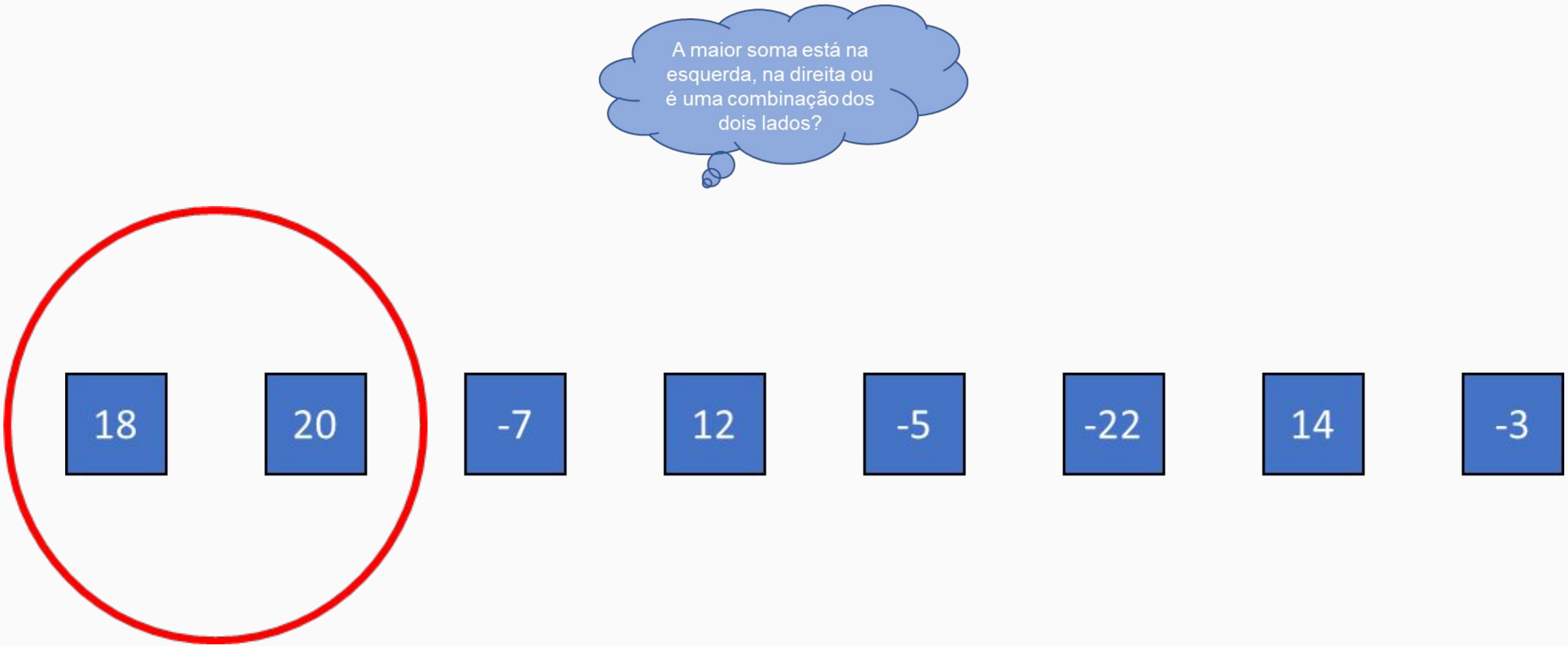
02 camada da recursividade.



03 camada da recursividade.

18	20	-7	12	-5	-22	14	-3

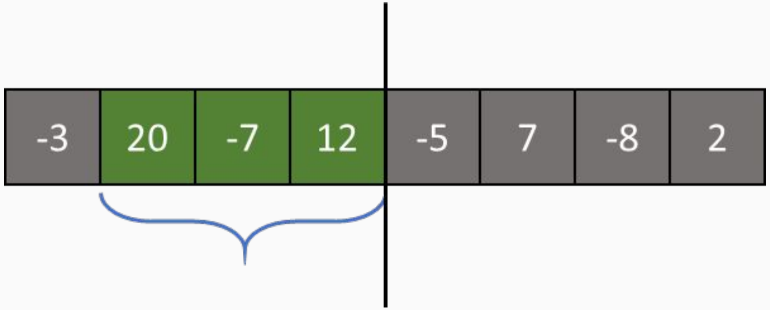
Conquista dos sub problemas



Conquista dos sub problemas

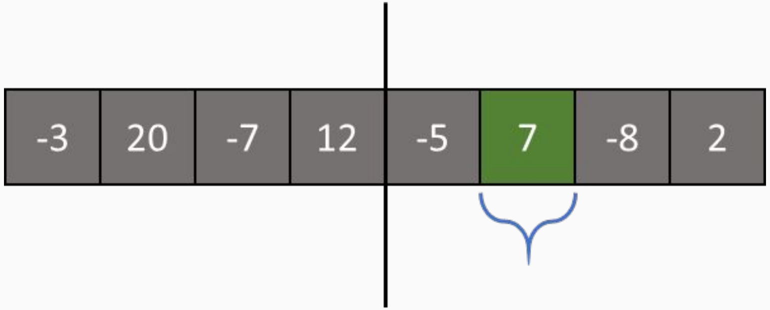
Existem 3 possibilidades

- O subarranjo máximo está na esquerda.



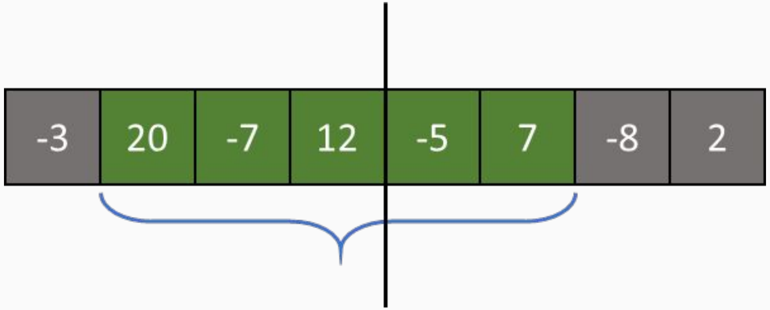
Existem 3 possibilidades

- O subarranjo máximo está na direita.



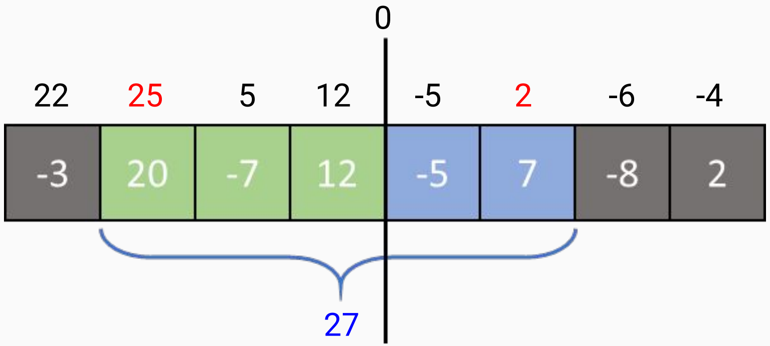
Existem 3 possibilidades

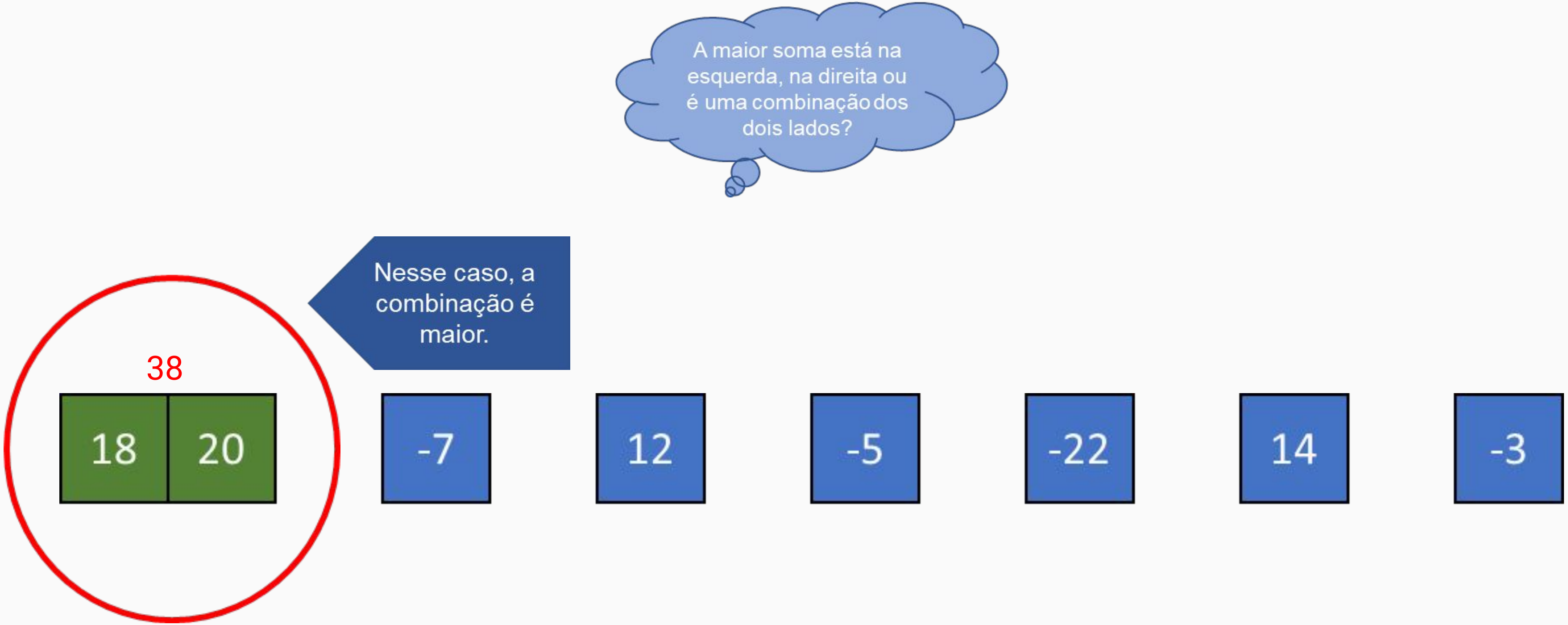
- O subarranjo máximo é uma combinação entre a esquerda e a direita.



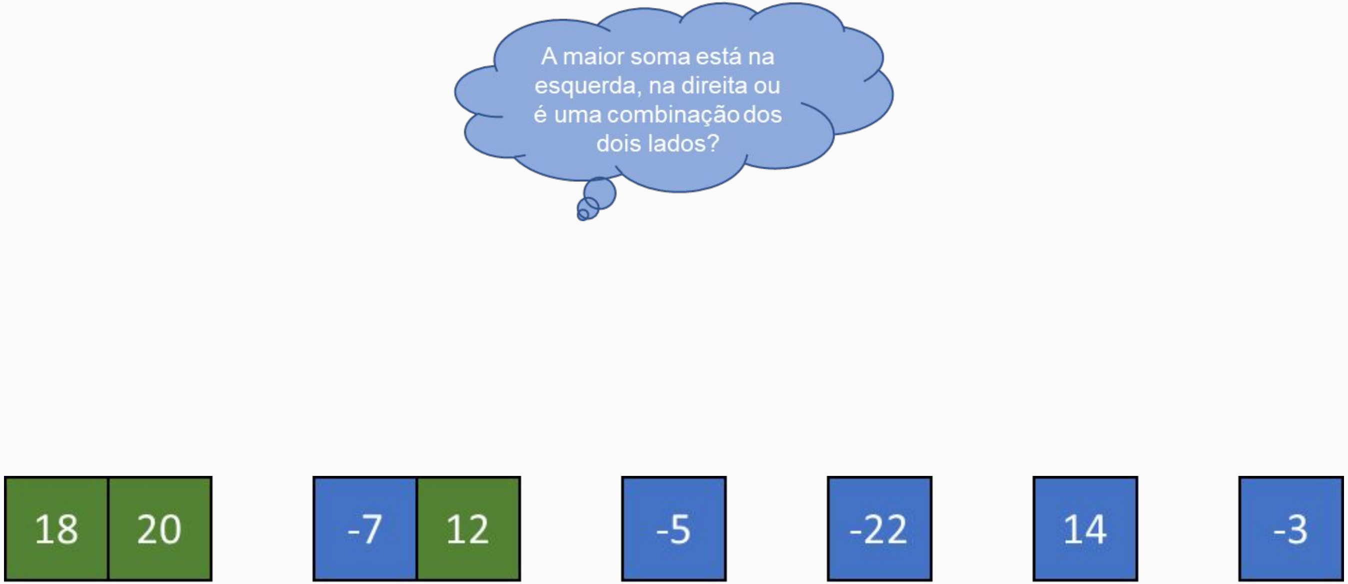
Combinação

- Como se faz a combinação?
  - Nada mais é do que o maior subarranjo começando do meio até o início + o maior subarranjo começando do meio até o final.
  - Como se calcula o maior subarranjo nesse caso?
    - Soma-se os elementos até atingir o maior valor possível.

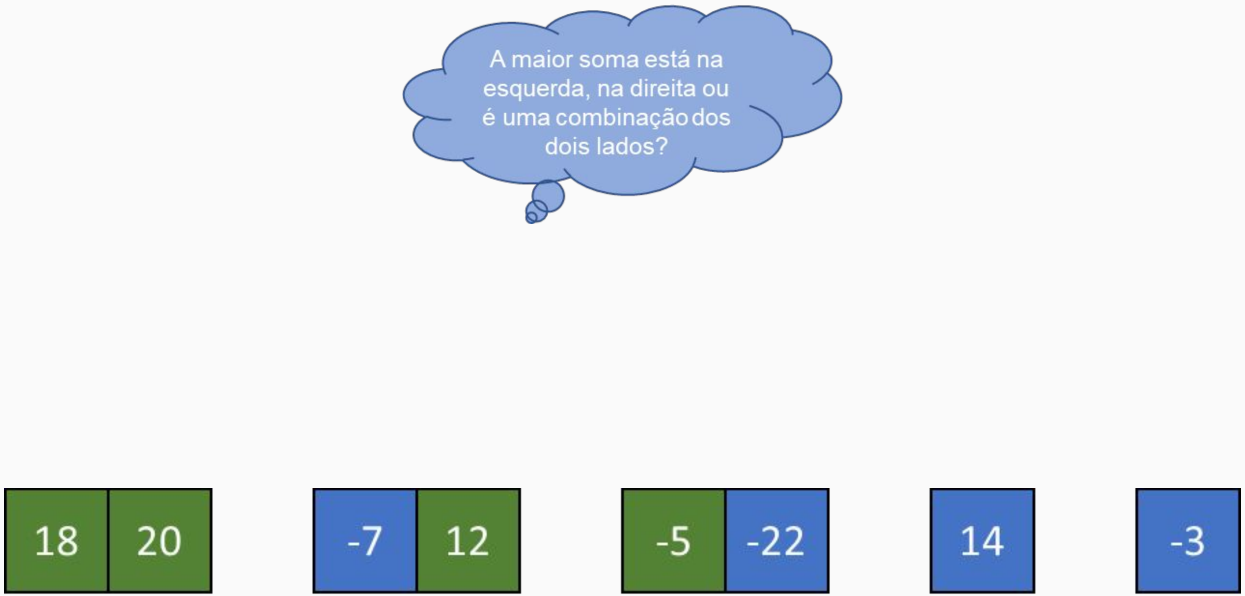




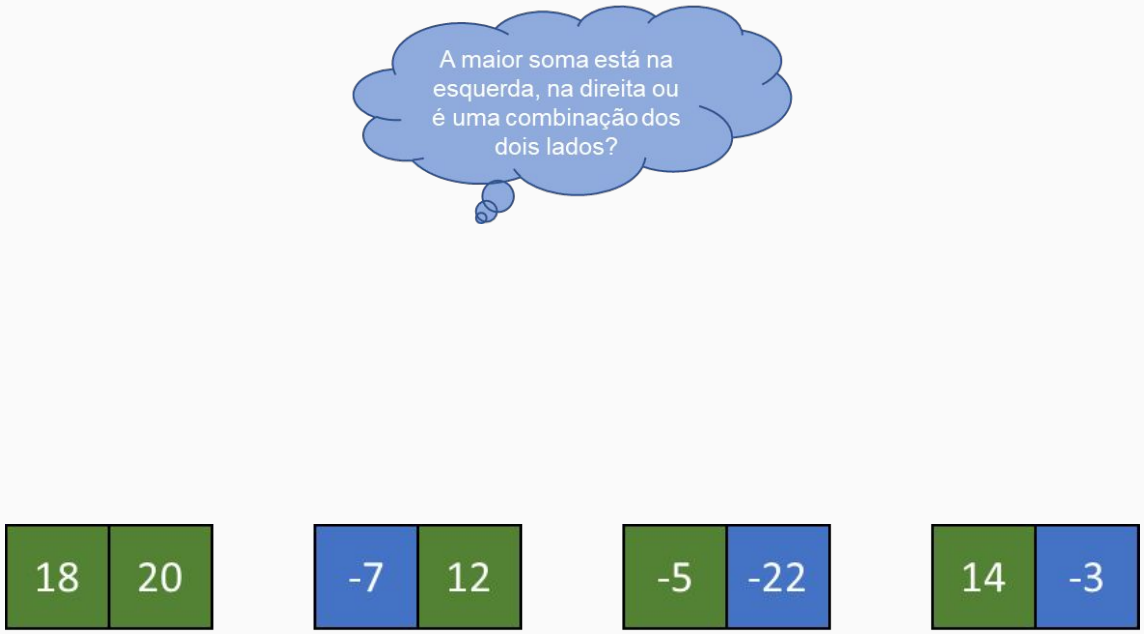
Conquista dos sub problemas



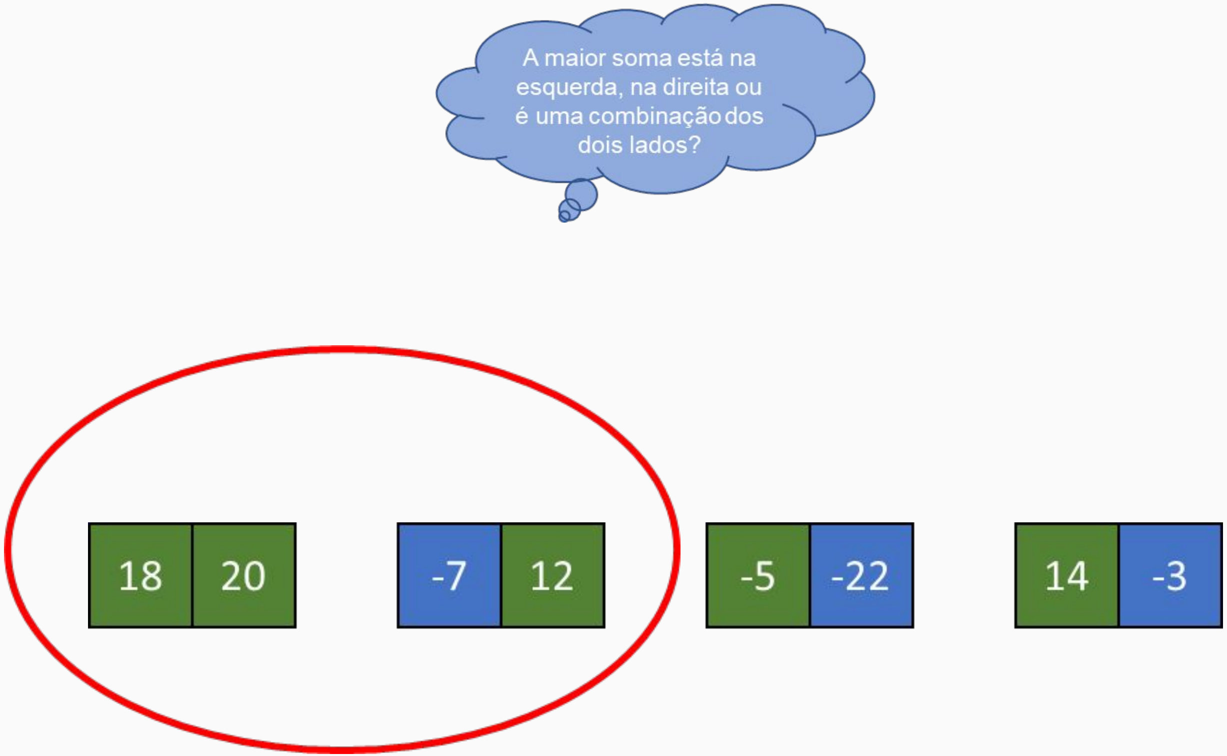
Continuando com os outros subproblemas ..



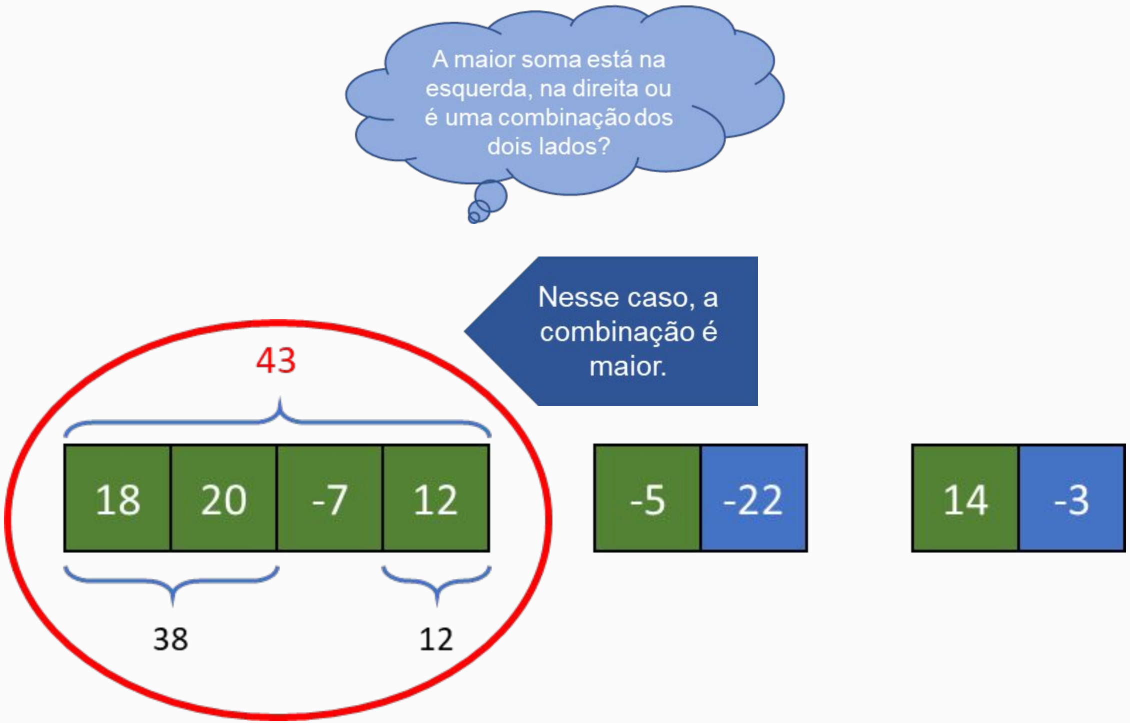
Continuando com os outros subproblemas ..



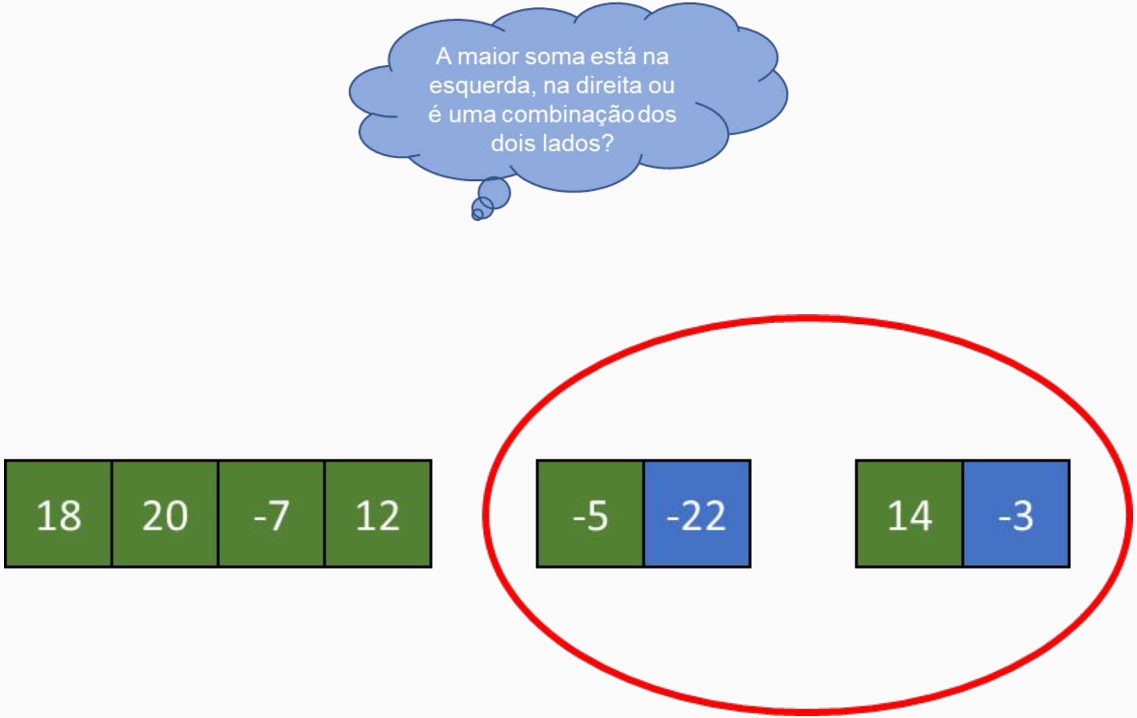
Continuando com os outros subproblemas ..



Subindo uma camada na conquista.



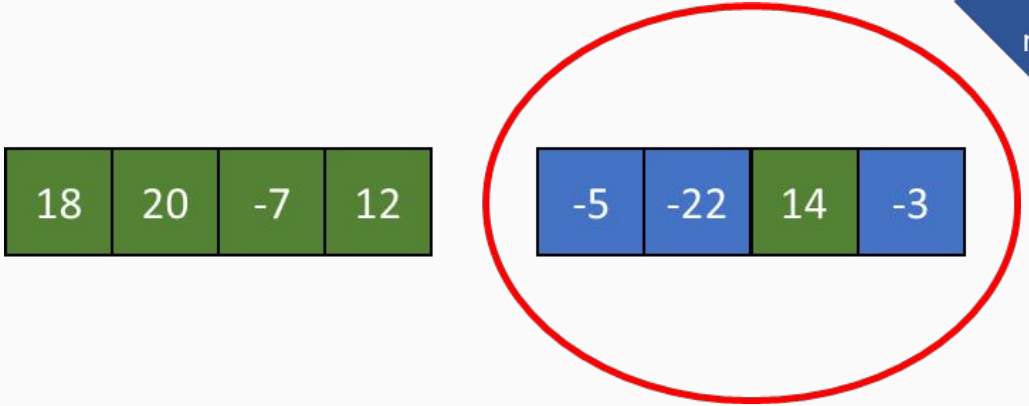
Subindo uma camada na conquista.



Continuando a conquista dos outros subproblemas.

A maior soma está na esquerda, na direita ou é uma combinação dos dois lados?

Nesse caso, a direita tem o maior subarranjo



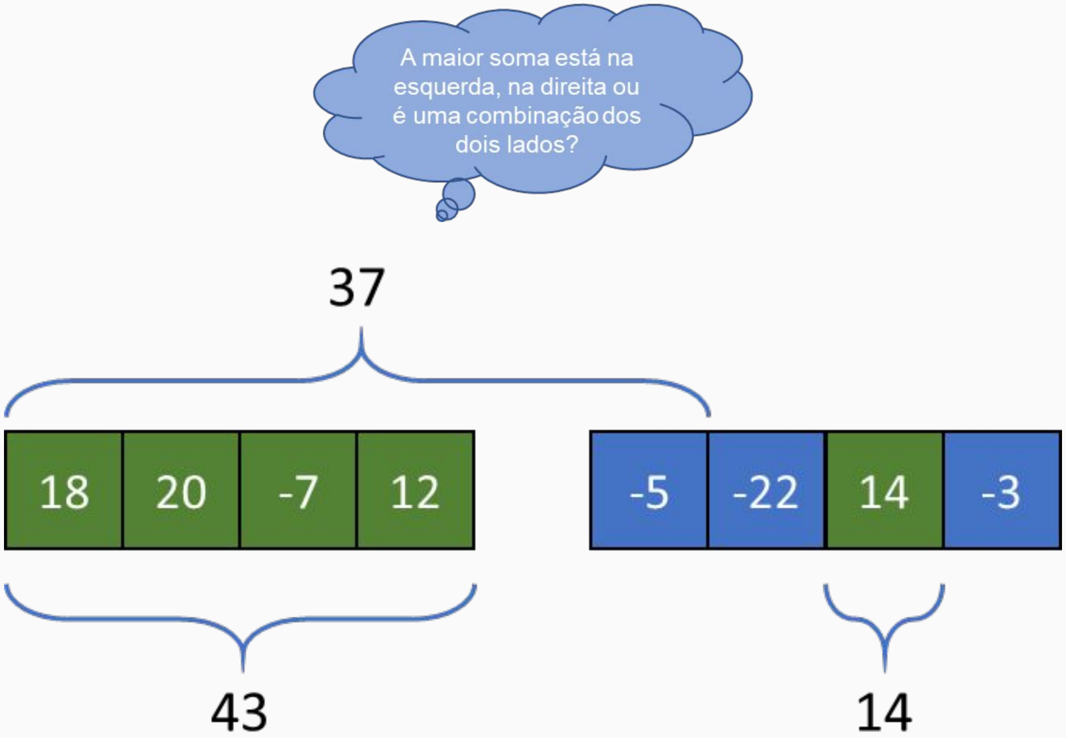
Continuando a conquista dos outros subproblemas.

A maior soma está na esquerda, na direita ou é uma combinação dos dois lados?

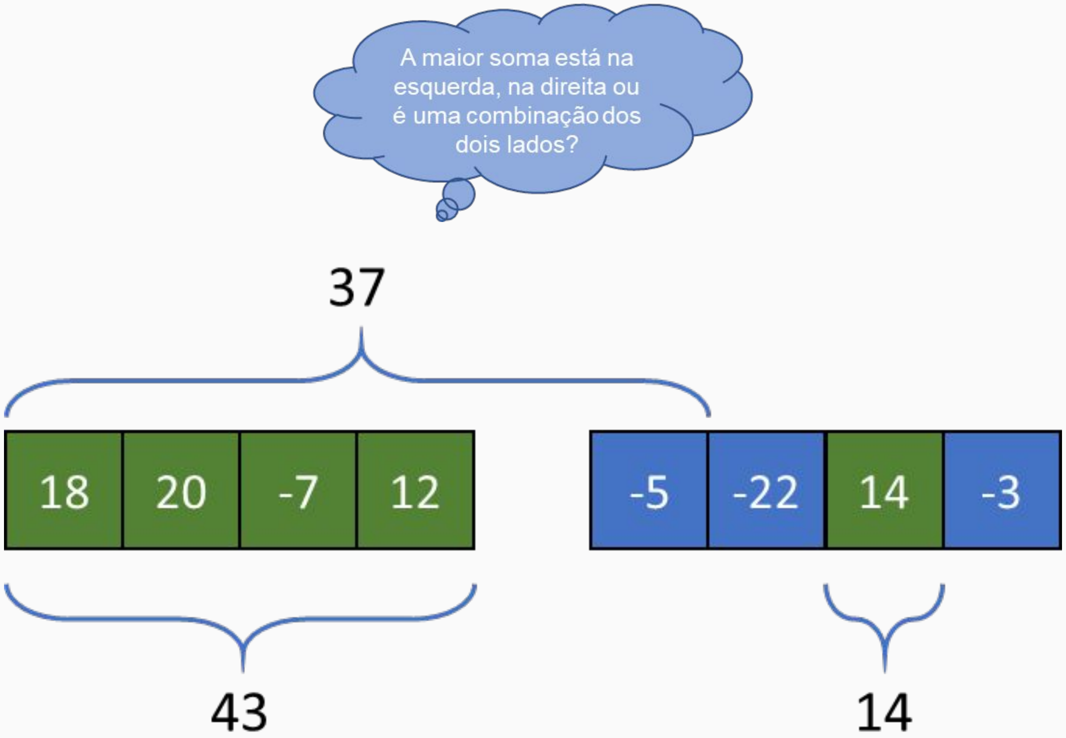
18	20	-7	12

-5	-22	14	-3

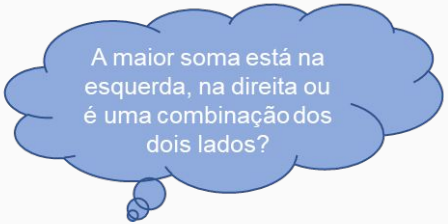
Subindo para a última camada da conquista.



Subindo para a última camada da conquista.



Subindo para a última camada da conquista.



18	20	-7	12	-5	-22	14	-3

No nosso caso, o subarranjo máximo vai do primeiro elemento até o quarto, com uma soma de 43.

Algoritmo

O algoritmo consiste na condição de parada que é quando o elemento do início é igual ao elemento do fim, ou seja, estamos tratando somente 1 elemento. Após isso, são feitas chamadas recursivas do algoritmo para esquerda, para a direita e na combinação do centro. E por fim um teste para verificar qual é a maior soma e retorná-la.

subarranjoMaximo(A, início, fim)

```
Se início = fim, então
    retorna início
Se não, então
    meio ← (inicio + fim) / 2

    somaEsq ← subarranjoMaximo(A, inicio, meio)
    somaDir   ← subarranjoMaximo(A, meio+1, fim)
    somaMeio ← combinarMeio(A, inicio, meio, fim)

    Se somaEsq > somaDir && somaEsq > somaMeio, então
        retorna somaEsq
    Se somaDir > soma Esq && somaDir > somaMeio, então
        retorna somaDir
    Se somaMeio > somaEsq && somaMeio > SomaDir, então
        retorna somaMeio
```

Combinação

A combinação consiste em pegar o maior subarranjo do meio ao início e o maior subarranjo do meio ao fim e somar seus valores. Para resgatar o subarranjo itera-se somando os elementos até encontrar a maior soma possível.

combinarMeio(A, início, meio, fim)

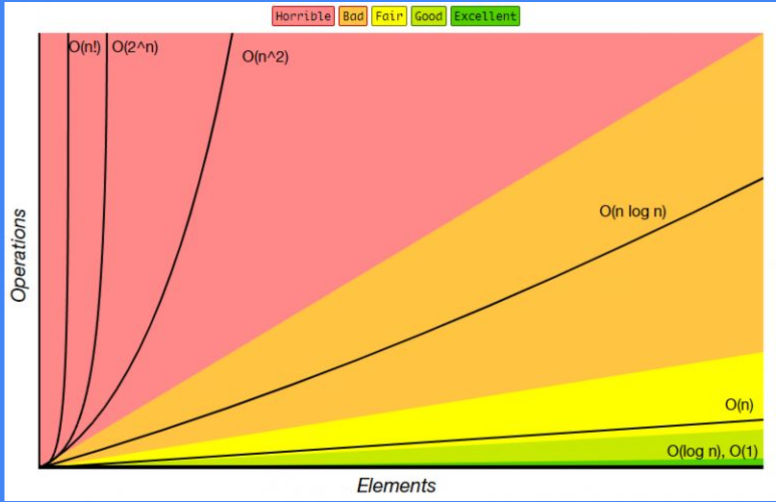
```
somaEsq ← infinito negativo
soma ← 0
para i ← meio até início faça
    soma ← soma + A[i]
    Se soma > somaEsq, então
        somaEsq ← soma
fim

somaDir ← infinito negativo
soma ← 0
para j ← meio +1 até fim faça
    soma ← soma + A[j]
    Se soma > somaDir, então
        somaDir ← soma
fim

retorna somaEsq + somaDir
```

Complexidade

Como calcular?



Algoritmo de combinação do meio.

combinarMeio(A, início, meio, fim)

```
somaEsq ← infinito negativo                O(1)
soma ← 0                                   O(1)
para i ← meio até início faça              O(n/2)
    soma ← soma + A[i]                         O(1)
    Se soma > somaEsq, então                   O(1)
        somaEsq ← soma                         O(1)
fim

somaDir ← infinito negativo                O(1)
soma ← 0                                   O(1)
para j ← meio +1 até fim faça              O(n/2)
    soma ← soma + A[j]                         O(1)
    Se soma > somaDir, então                   O(1)
        somaDir ← soma                         O(1)
fim

retorna somaEsq + somaDir                  O(1)
```

Temos que :

$2 * O(n/2) + 11 * O(1) = O(n/2) = O(n)$

Algoritmo do subarranjo máximo.

subarranjoMaximo(A, início, fim)

```
Se início = fim, então                                        O(1)
    retorna início                                            O(1)
Se não, então                                                 O(1)
    meio ← (inicio + fim) / 2                                 O(1)

    somaEsq ← subarranjoMaximo(A, inicio, meio)               T(n/2)
    somaDir  ← subarranjoMaximo(A, meio+1, fim)               T(n/2)
    somaMeio ← combinarMeio(A, inicio, meio, fim)             O(n)

    Se somaEsq > somaDir && somaEsq > somaMeio, então         O(1)
        retorna somaEsq                                       O(1)
    Se somaDir > soma Esq && somaDir > somaMeio, então        O(1)
        retorna somaDir                                       O(1)
    Se somaMeio > somaEsq && somaMeio > SomaDir, então        O(1)
        retorna somaMeio                                      O(1)
```

Temos que :

$2 * T(n/2) + O(n) + 10 * O(1) = 2T(n/2) + O(n)$

Problema

- Para simplificar a análise, consideremos que o tamanho do problema é uma potência de 2, ou seja, o tamanho do array é sempre par.
- Seja $T(n)$ o tempo de execução do algoritmo.
  - $O(1) = T(1) = 1 \rightarrow$ caso base para operação de tempo constante.
  - Sabendo que o problema da combinação possui complexidade de $O(n)$.
  - No passo recursivo, dividimos o array em duas partes iguais, então necessita-se de $T(n/2)$ de tempo para resolver cada um.
    - Como resolve-se 2 subproblemas, tem-se $2T(n/2)$.
  - Logo, a complexidade do algoritmo do subarranjo máximo é de $2T(n/2) + O(n)$.

Problema

- Utilizando o teorema mestre para resolver a recorrência $2T(n/2) + O(n)$, tem-se que:
  - $a = 2$, $b = 2$ e $f(n) = O(n)$ respeitam as condições do teorema.
  - $T(n) = O(n \lg n)$

Obrigado!

Contato:

Paulo E. R. Araujo

linkedin.com/in/pumba-dev/

pumbadeveloper@gmail.com
www.pumbadev.com