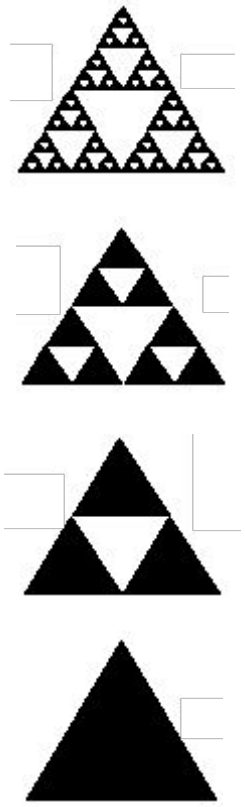
Divisão e Conquista

Prof. Martín Vigil

Recursão

- É definir um objeto em termos de outros objetos do mesmo tipo
- Os outros objetos são instâncias mais simples ou menores



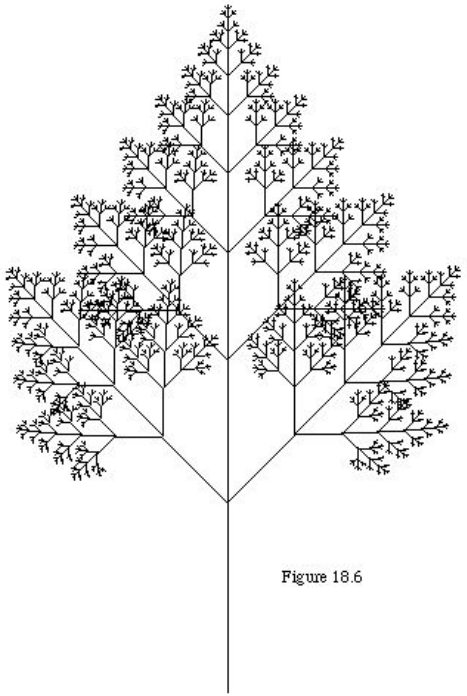


Figure 18.6

Algoritmos recursivos

- Algoritmos que fazem chamadas a si mesmos

```
1.  função Fib(n)
2.      se n == 1 OU n == 2
3.          retorne 1
4.      senão
5.          retorne Fib(n-1) + Fib(n-2)
```

Divisão e Conquista: Intuição

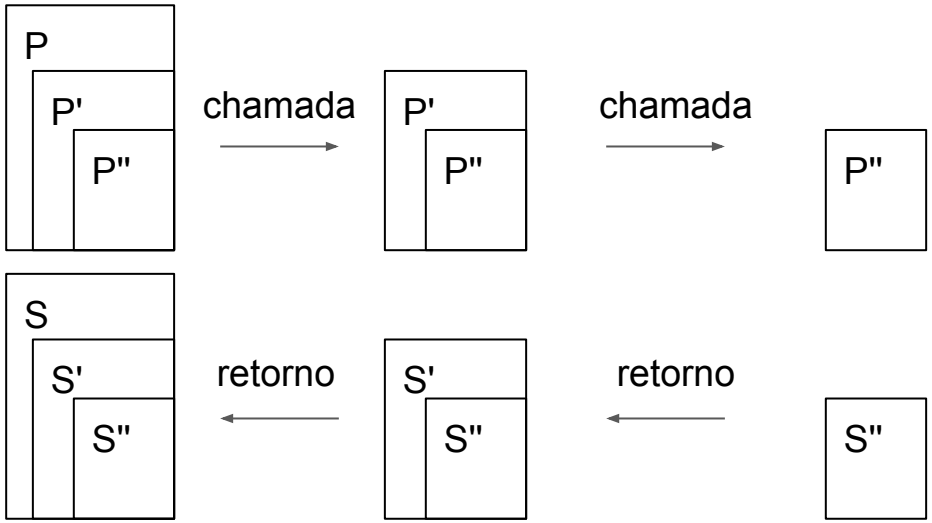
Dividir um problema difícil em partes menores e ***mais fáceis*** de serem conquistadas (solucionadas)

A solução das partes menores leva à solução do problema

Divisão e Conquista: Etapas

- Usa algoritmos recursivos
- Etapas da Técnica

1. **Dividir** o problema em subproblemas
2. **Conquistar** os subproblemas através de chamadas recursivas. Se o subproblema for trivial, resolvê-lo de maneira direta
3. **Combinar** as soluções dos subproblemas obtendo a solução do problema original



Divisão e Conquista: Complexidade

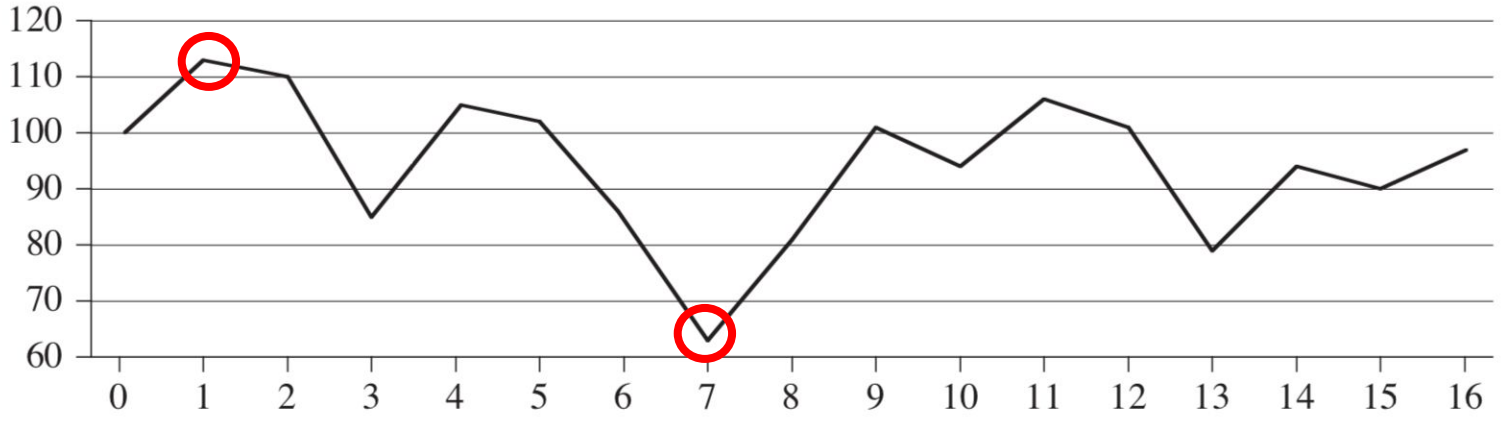
- Função de recorrência baseada nos três passos anteriores:
  - Seja T (n) o tempo de execução de um problema de tamanho n
  - Se o problema for pequeno o bastante (n ≤ c), então a complexidade para resolvê-lo é Θ(1) (solução direta)
  - Suponha que o problema foi dividido em a > 0 subproblemas, cada um com tamanho 1/b do tamanho original
  - Suponha que D(n) seja o tempo para dividir o problema em subproblemas
  - Suponha que C(n) seja o tempo para combinar as soluções

$$T(n) = \begin{cases} \Theta(1) & \text{se } n \leq c \\ aT(\frac{1}{b}) + D(n) + C(n) & \text{caso contrário} \end{cases}$$

Problema do máximo subarray

Buscando lucros no mercado acionário

- Você pode comprar/vender no fim do pregão
- Você pode ver o futuro
- Compre no mínimo e venda no máximo preço
- Nem sempre possível: máximo pode anteceder mínimo preço

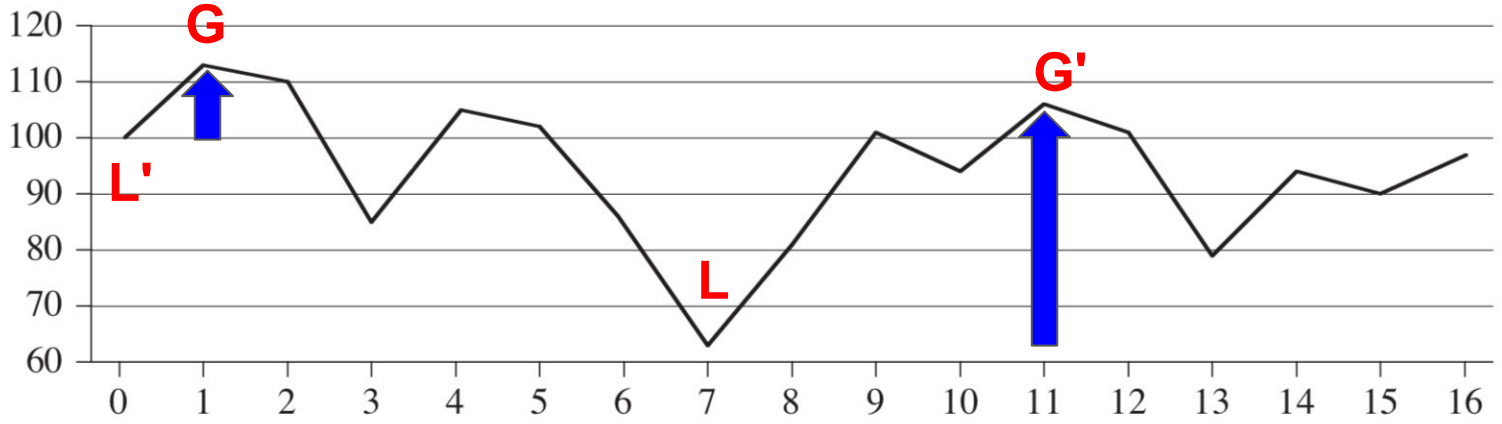


Day	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Price	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
Change		13	−3	−25	20	−3	−16	−23	18	20	−7	12	−5	−22	15	−4	7

Buscando lucros no mercado acionário

- *Nem sempre possível: máximo pode anteceder mínimo preço*

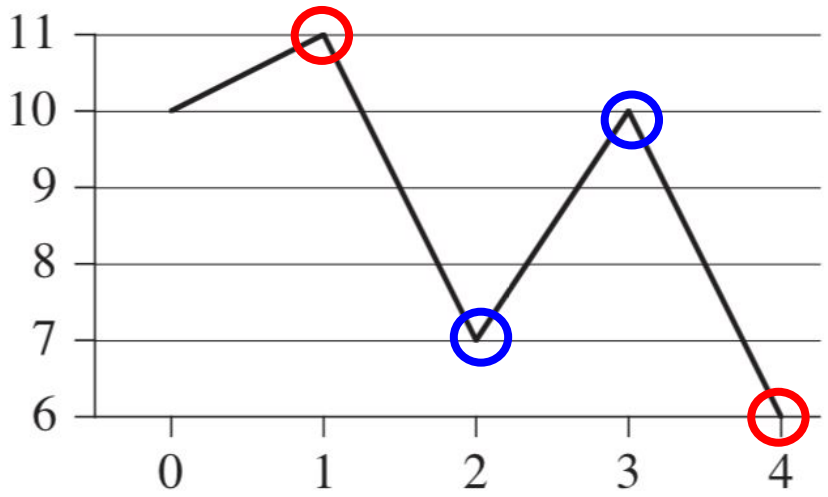
1. Encontre os máximo e mínimo preços G e L, respectivamente
2. Encontre o mínimo L' à esquerda do G
3. Encontre o máximo G' à direita de L
4. Escolha o max{G-L', G'-L}



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

Buscando lucros no mercado acionário

- Máximo lucro pode não estar entre mínimo e máximo preços



Day	0	1	2	3	4
Price	10	11	7	10	6
Change		1	−4	3	−4

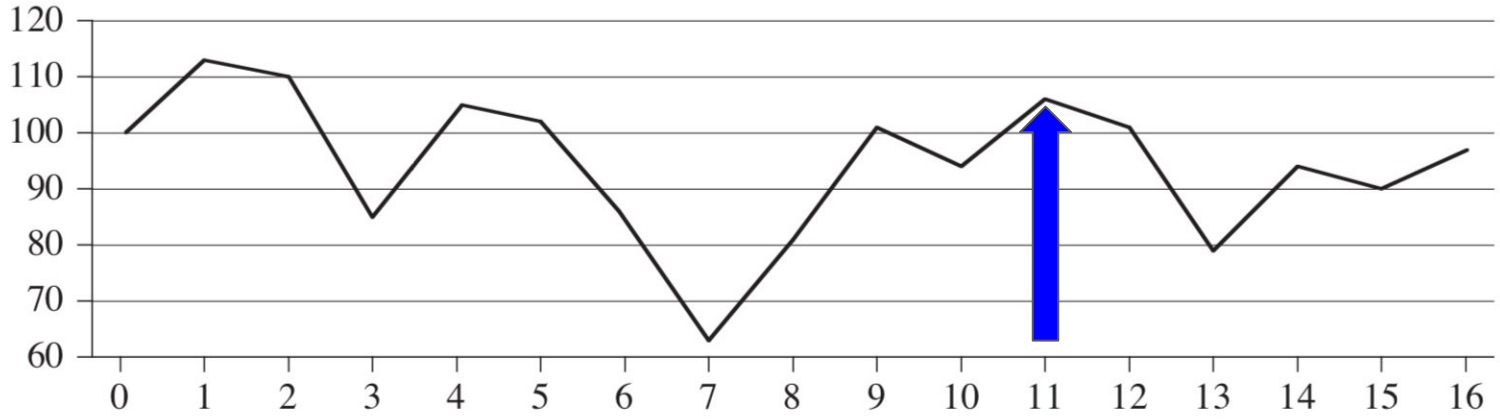
Solução por força bruta

- Calcule o lucro de todos os pares (preço compra, preço venda)
- Combinação (n,2)
- Complexidade assintótica?

```
1. maximo = 0
2. para i=1,2,..., n
3.   para j=1,2,..., n
4.     lucro = valores[i] - valores[j]
5.     maximo = max{maximo, lucro}
6. retorne maximo
```

(um) Máximo subarray

- Subarray contínuo cuja soma dos elementos é máxima
- Se não houver valores negativos, encontrá-lo é trivial



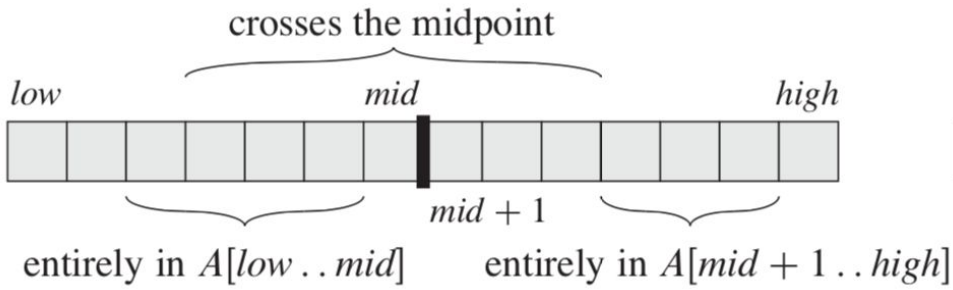
Day	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Price	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
Change		13	−3	−25	20	−3	−16	−23	18	20	−7	12	−5	−22	15	−4	7

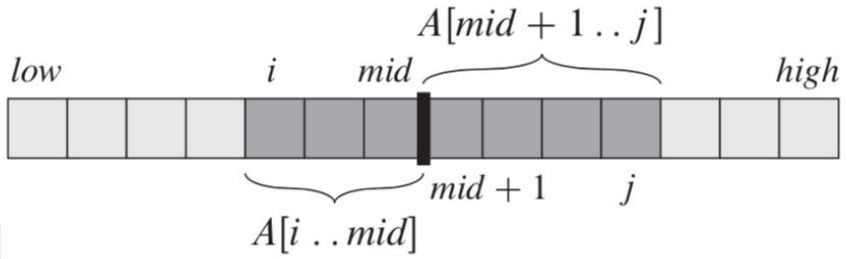
Máximo subarray: +43

Solução via Divisão e Conquista

- Seja A[*low*...*high*] um vetor de valores
- Divide-se A ao meio em A[*low*...*mid*] e A[*mid+1*...high]
- Um subarray de A pode estar em uma das opções:

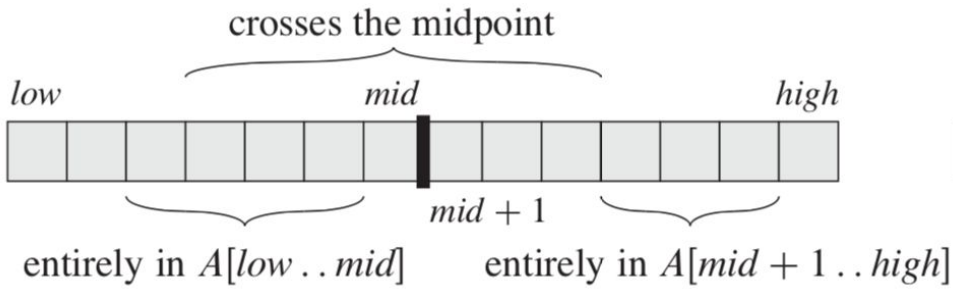
1. Na metade à esquerda A[low...mid] → min ≤ i ≤ j ≤ mid; ou
2. Na metade à direita A[mid+1...high] → mid < i ≤ j ≤ high ; ou
3. Cruzando o meio → low ≤ i ≤ mid < j ≤ high

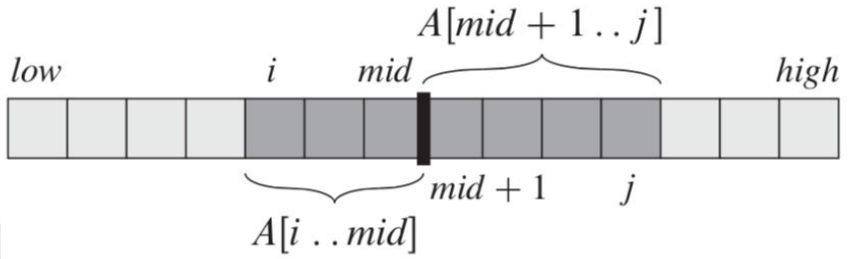




Solução via Divisão e Conquista

- O máximo subarray de A tem a máxima soma dos subarrays que estão nas metades de A e cruzando o meio de A
- Calcular os máximo subarrays nas metades é um subproblema resolvido recursivamente
- Calcular o máximo subarray não é um subproblema mas tem 2 partes





Máximo subarray cruzando o meio

```
FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
 1  left-sum = −∞
 2  sum = 0
 3  for i = mid downto low
 4       sum = sum + A[i]
 5       if sum > left-sum
 6            left-sum = sum
 7            max-left = i
 8  right-sum = −∞
 9  sum = 0
10  for j = mid + 1 to high
11       sum = sum + A[j]
12       if sum > right-sum
13            right-sum = sum
14            max-right = j
15  return (max-left, max-right, left-sum + right-sum)
```

Complexidade: $\Theta(n)$

Máximo subarray nas metades e cruzando

```
FIND-MAXIMUM-SUBARRAY(A, low, high)
 1  if high == low
 2       return (low, high, A[low])          // base case: only one element
 3  else mid = ⌊(low + high)/2⌋
 4       (left-low, left-high, left-sum) =
              FIND-MAXIMUM-SUBARRAY(A, low, mid)
 5       (right-low, right-high, right-sum) =
              FIND-MAXIMUM-SUBARRAY(A, mid + 1, high)
 6       (cross-low, cross-high, cross-sum) =
              FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
 7       if left-sum ≥ right-sum and left-sum ≥ cross-sum
 8            return (left-low, left-high, left-sum)
 9       elseif right-sum ≥ left-sum and right-sum ≥ cross-sum
10            return (right-low, right-high, right-sum)
11       else return (cross-low, cross-high, cross-sum)
```

Complexidade

```
FIND-MAXIMUM-SUBARRAY(A, low, high)
 1  if high == low
 2       return (low, high, A[low])          // base case: only one element
 3  else mid = ⌊(low + high)/2⌋
 4       (left-low, left-high, left-sum) =
              FIND-MAXIMUM-SUBARRAY(A, low, mid)
 5       (right-low, right-high, right-sum) =
              FIND-MAXIMUM-SUBARRAY(A, mid + 1, high)
 6       (cross-low, cross-high, cross-sum) =
              FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
 7       if left-sum ≥ right-sum and left-sum ≥ cross-sum
 8            return (left-low, left-high, left-sum)
 9       elseif right-sum ≥ left-sum and right-sum ≥ cross-sum
10            return (right-low, right-high, right-sum)
11       else return (cross-low, cross-high, cross-sum)
```

Θ(1)

Θ(n)

T(n/2)

Θ(1)

Complexidade do algoritmo

- T(n) = Θ(1) + 2T(n/2)+ Θ(n) + Θ(1)

$$T(n) = \begin{cases} \Theta(1) & \text{se } n = 1 \\ 2T(\frac{n}{2}) + \Theta(n) & n \geq 1 \end{cases}$$

$$T(n) \in O(n \log n)$$