MAUÁ

Análise de Complexidade e Ordenação

ECM404

Complexidade Computacional

- O mesmo problema, com frequência, pode ser resolvido com algoritmos que diferem em eficiência. Estas diferenças podem ser irrelevantes para processar um pequeno número de dados, mas crescem proporcionalmente com a quantidade de dados.
- Para comparar a eficiência de algoritmos, uma medida foi desenvolvida por Juris Hartmanis e Richard E. Stearns, chamada de complexidade computacional.

Complexidade Computacional

- Para avaliar a eficiência de um algoritmo, unidades de tempo real, como micro e nanossegundos, não são utilizadas. Ao invés disso, são utilizadas unidades lógicas que expressam uma relação entre a quantidade **n** de dados e a quantidade de tempo **t** exigida para processar estes dados.

Complexidade Computacional

- Exemplo:
  - Se houver uma relação linear entre **n** e **t**, isto é, $t_1 = c \cdot n_1$, então um aumento dos dados por um fator de 5 resulta num aumento do tempo de execução pelo mesmo fator. Se $n_2 = 5 \cdot n_1$ então $t_2 = 5 \cdot t_1$
  - Da mesma forma, se $t_1 = \log_2 n$, então duplicando-se n aumenta-se t somente por uma unidade $(t_2 = \log_2(2 \cdot n) \ \rightarrow \ t_2 = \ t_1 + 1)$.

Complexidade Assintótica

- Entretanto, uma função que expressa a relação entre **n** e **t** é muito mais complexa, e o cálculo desta função é importante somente em relação a grandes quantidades de dados. Assim, quaisquer termos que não modifiquem substancialmente a grandeza da função devem ser eliminados.
- Desta forma, não teremos uma medida precisa, mas uma aproximação suficientemente boa, chamada de **complexidade assintótica.**

Complexidade Assintótica

- Considere a função $f(n) = n^2 + 100n + \log_{10} n + 1000$

n	$f(n)$	n^2		$100n$		$\log_{10} n$		1000	
	Valor	Valor	%	Valor	%	Valor	%	Valor	%
1	1.101	1	0,10	100	9,10	0	0,00	1000	90,83
10	2.101	100	4,76	1.000	47,60	1	0,05	1000	47,60
100	21.002	10.000	47,60	10.000	47,60	2	0,99	1000	4,76
1.000	1.101.003	1.000.000	90,80	100.000	9,10	3	0,00	1000	0,09
10.000	101.001.004	100.000.000	99,00	1.000.000	0,99	4	0,00	1000	0,00
100.000	10.010.001.005	10.000.000.000	99,90	10.000.000	0,09	5	0,00	1000	0,00

Notação O-Grande

- A notação mais utilizada para representar a complexidade assintótica, isto é, a taxa de crescimento da função, é a O-Grande, introduzida em 1894 por Paul Bachmann.

Notação O-Grande

- **Definição:** $f(n)$ é $O(g(n))$ se existem números positivos **c** e N tais que $f(n) \leq c \cdot g(n)$ para todo $n \geq N$
  - Esta definição é lida: f é O-Grande de g se há um número positivo c tal que f não seja maior do que c vezes g para todos "enes" maiores que algum número N.
  - A relação entre f e g pode ser expressa estabelecendo-se tanto que $g(n)$ é um limite superior no valor de $f(n)$ ou que f cresce no máximo tão rápido quanto g.

Encontrando o O-Grande

- Na maioria dos casos estamos interessados na complexidade de tempo, que usualmente mede o número de atribuições e comparações realizadas durante a execução de um programa. Inicialmente, iremos considerar apenas o número de atribuições.

Encontrando o O-Grande

- Exemplo 1:
  - Um laço simples para calcular a soma dos números em um array.

```
for(i=0;i<n;i++){
    soma += vet[i];
}
```

  - O laço *for* itera *n* vezes e, durante cada iteração, executa duas atribuições, uma para atualizar o valor da variável *soma* e outra para *i*. Assim, existem *1+2n* atribuições para a execução completa do laço *for*, ou seja, sua complexidade é *O(n)*.

Encontrando o O-Grande

- Exemplo 2:
  - Um laço para calcular a soma dos números em uma matriz.

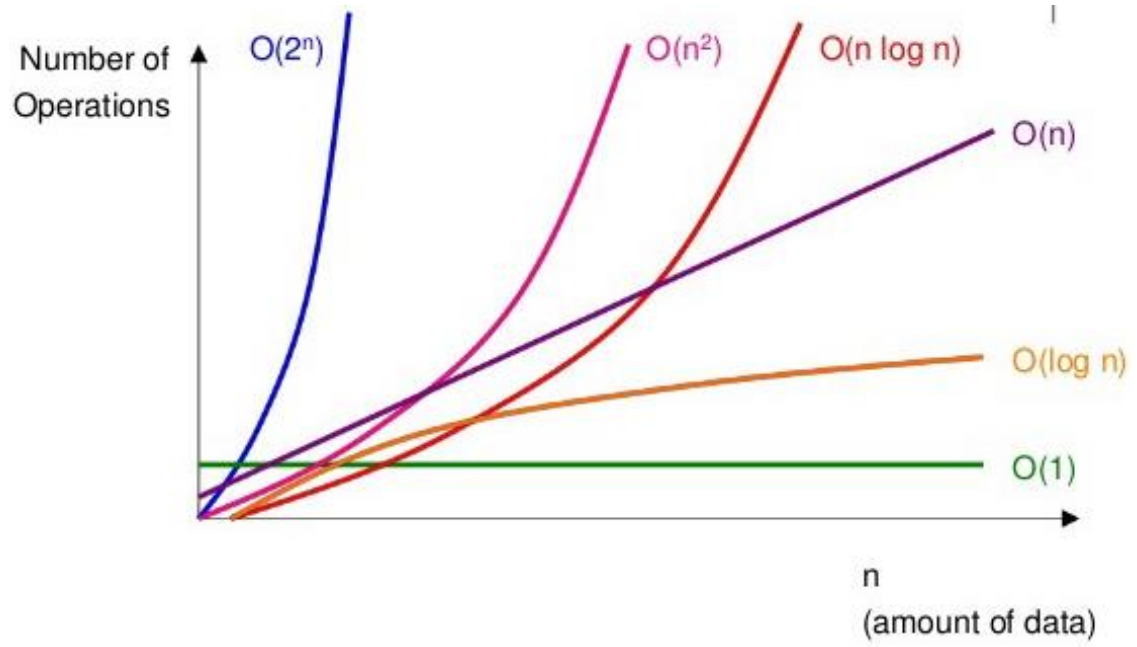
```
for(i=0;i<n;i++){
  for(j=0;j<n;j++){
    soma += mat[i][j];
  }
}
```

  - O laço *for* itera *n* vezes e, para cada iteração de *i*, itera *n* vezes *j*. Assim, existem *1+n(2n)* atribuições para a execução dos dois laços, ou seja, sua complexidade é $O(n^2)$.

Complexidade – Algoritmos de Busca

- Comparação da complexidade dos algoritmos de busca.

	Complexidade		
	Melhor Caso	Esperado	Pior Caso
Busca Linear	O(1)	O(n)	O(n)
Busca Binária	O(1)	O(log n)	O(log n)



Ordenação de Dados

- A eficiência do manuseio de dados muitas vezes pode ser substancialmente aumentada se os dados forem ordenados segundo algum critério. Por exemplo, seria praticamente impossível encontrar um nome na lista telefônica se os nomes não estivessem ordenados alfabeticamente.

Comparação entre algoritmos

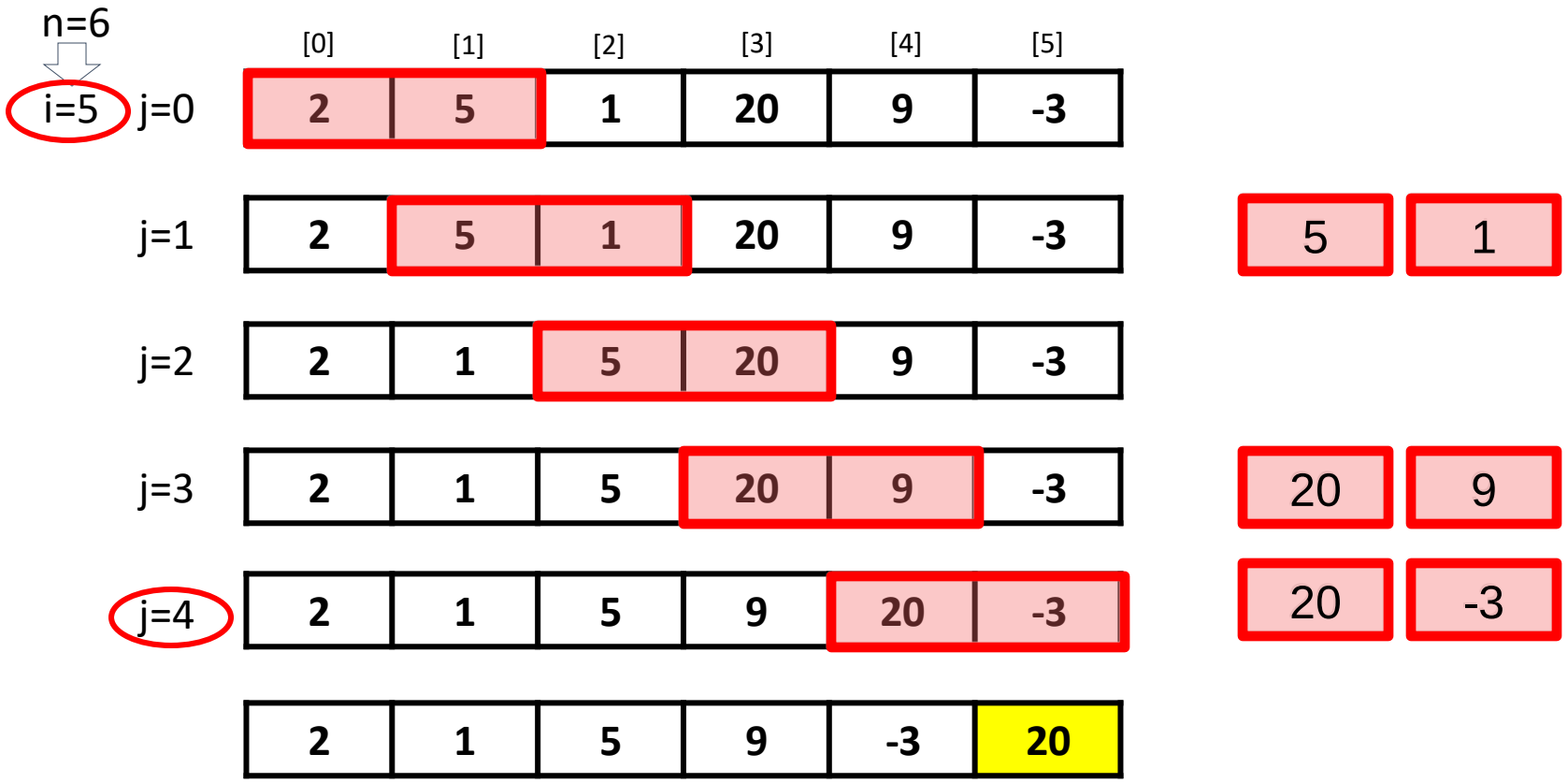
Ordenação de Dados

- Comparação da complexidade dos algoritmos de ordenação.

	Complexidade		
	Melhor Caso	Esperado	Pior Caso
Busca Linear	O(1)	O(n)	O(n)
Busca Binária	O(1)	O(log n)	O(log n)
Bubble Sort	O(n)	$O(n^2)$	$O(n^2)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Inserction Sort	O(n)	$O(n^2)$	$O(n^2)$
Merge Sort	O(n log n)	O(n log n)	O(n log n)
Quick Sort	O(n log n)	O(n log n)	$O(n^2)$
Heap Sort	O(n log n)	O(n log n)	O(n log n)
Bucket Sort	O(n + k)	O(n + k)	$O(n^2)$

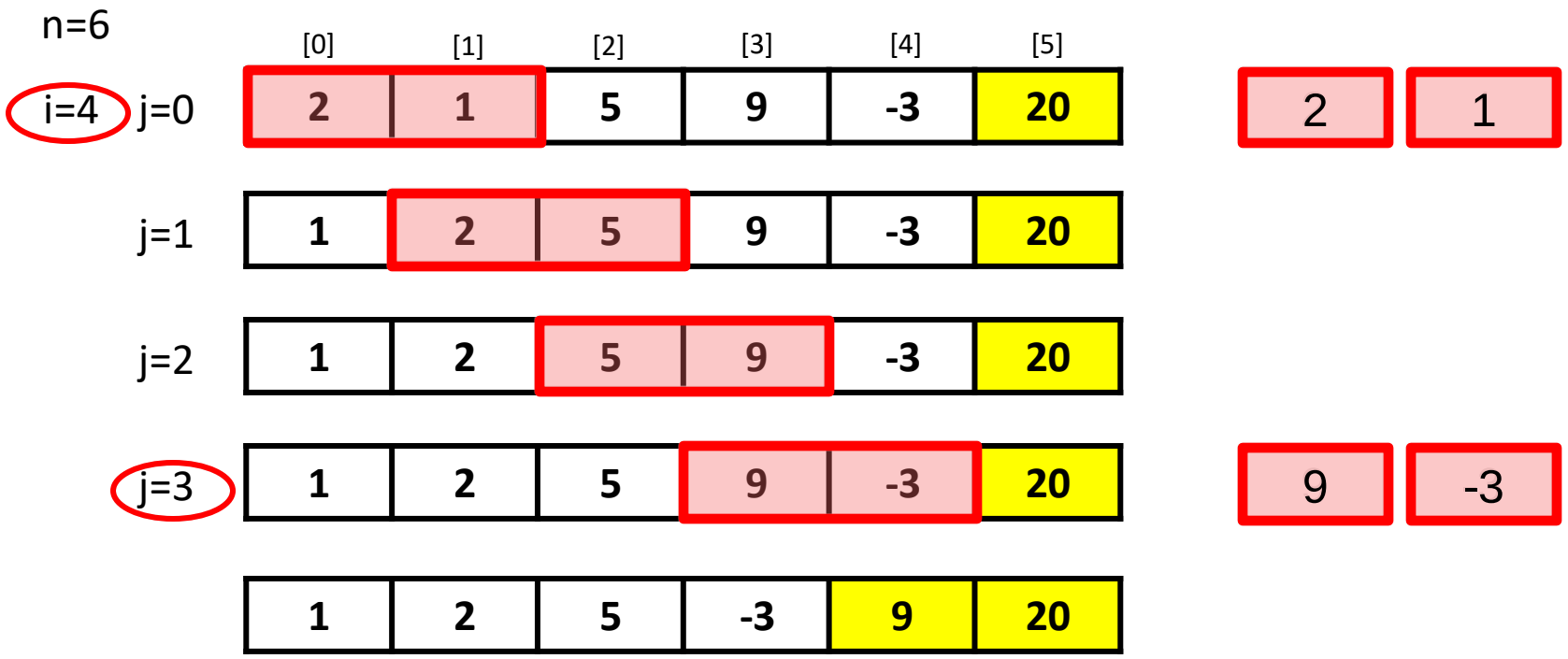
Bubble Sort

- Dado um vetor com n elementos a0, a1, ..., an-1, analisar dois elementos contíguos e fazer a troca caso aj > aj+1.



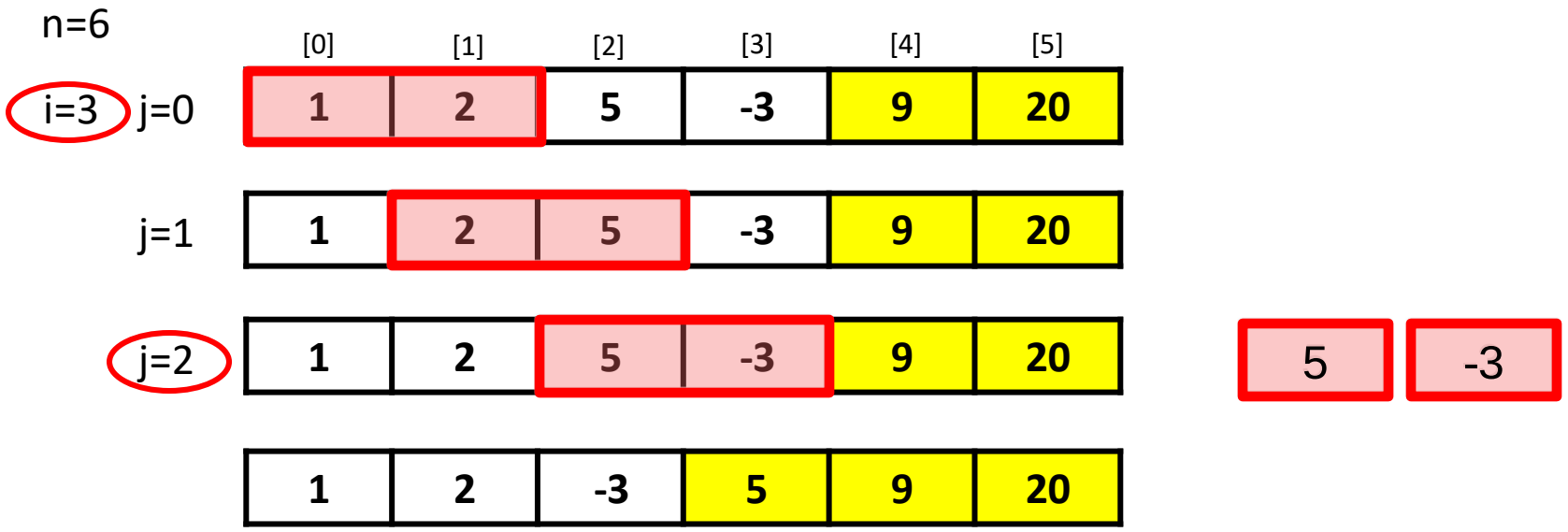
Bubble Sort

- Repetir o processo, levando os maiores para o final do vetor.



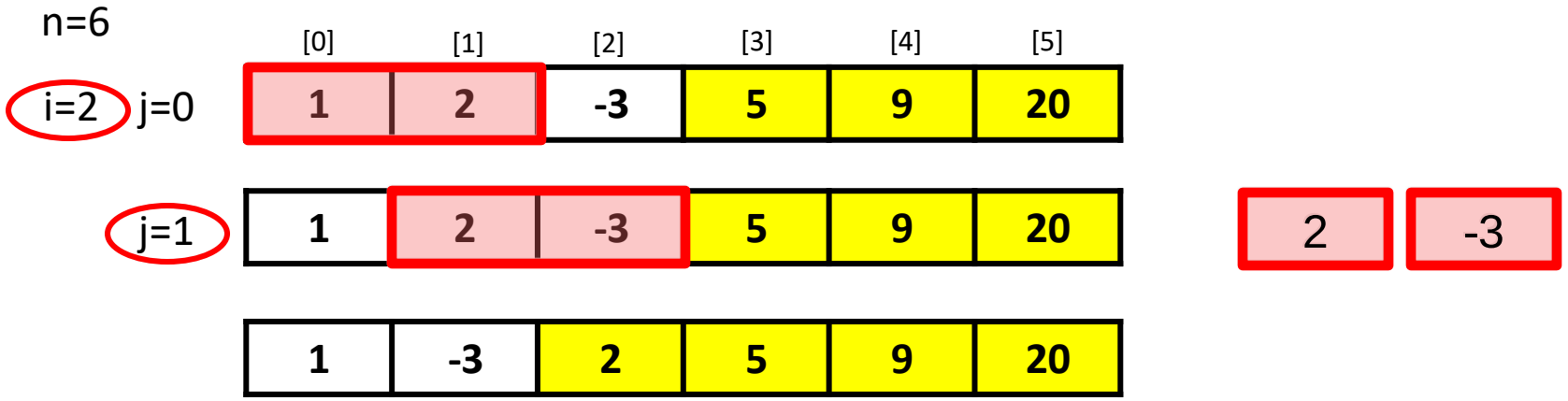
Bubble Sort

- Repetir o processo, levando os maiores para o final do vetor.



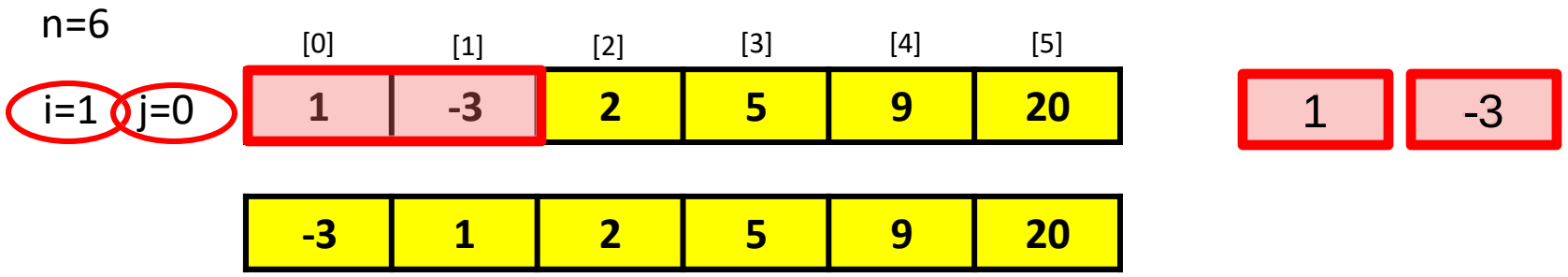
Bubble Sort

- Repetir o processo, levando os maiores para o final do vetor.



Bubble Sort

- Repetir o processo, levando os maiores para o final do vetor.



Bubble Sort

- O pseudocódigo do algoritmo bubble sort é:
  - Analisar do primeiro ao penúltimo elemento do vetor não ordenado;
    - Se o elemento analisado for maior do que o próximo valor, então troque-os de posição;
  - Repetir o processo para as demais posições não ordenadas do vetor;

Selection Sort

- O pseudocódigo do algoritmo selection sort é:
  - Analisar o vetor procurando pelo elemento de menor valor;
    - Se o elemento analisado for menor do que o menor valor já encontrado, atualize o índice que representa a posição com o menor valor;
    - Após analisar todo o vetor, troque a primeira posição do vetor com o elemento de menor valor (utilize o índice da varredura);
  - Repetir o processo para as demais posições não ordenadas do vetor;

Atividade

- Abra a pasta Downloads no VS Code
  - Utilize o comando git clone https://github.com/ECM404/atividade2.git --recursive
  - Entre na pasta utilizando o comando cd atividade2
  - Inicialize o diretório com o comando make install