# Laboratório de Algoritmos e Técnicas de Programação II

## Aula 07 - Ordenação interna: seleção, inserção, força bruta, shell sort, quick sort e merge sort

Álvaro Magri Nogueira da Cruz

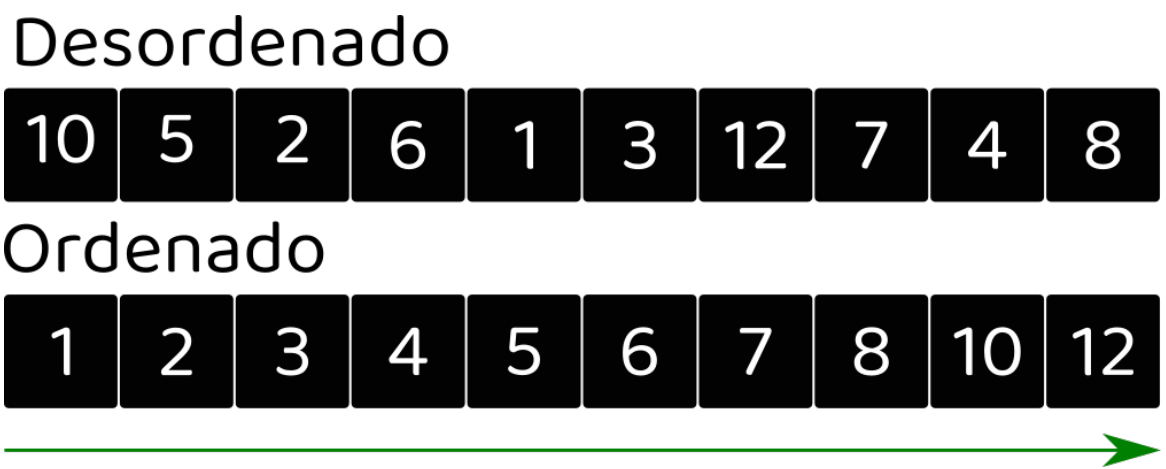
# Agenda

1. **Introdução**
2. **Métodos simples**
   - Bubble sort
   - Insertion sort
   - Selection sort
3. **Métodos sofisticados**
   - Merge sort
   - Quick sort
   - Shell Sort
4. **Exercícios**
5. **Referências**

# Introdução I

## Definição

- Um algoritmo de ordenação tem como objetivo estabelecer algum tipo de **ordem**, **crescente ou decrescente**, em um conjunto de dados.



# Introdução II

## Observações

- Vamos dividir nosso estudo em duas partes:
  - Métodos simples;
  - Métodos sofisticados.

# Métodos simples I

## Definição

- Os métodos simples de ordenação são aqueles que permitem uma movimentação dos elementos de maneira fácil e, além disso, são muito simples de serem implementados.

## Os métodos

- Bubble sort;
- Insertion sort;
- Selection sort.

# Bubble sort I

## Características

- É o algoritmo mais simples. A ideia é percorrer o vetor até que todos elementos estejam ordenados;
- **Recomendação de uso:** pequenos vetores/listas;
- **Complexidade pior caso:** $O(n^2)$
- **Complexidade melhor caso:** $O(n)$
- **Complexidade caso médio:** $O(n^2)$
- **Divirta-se:** `https://www.youtube.com/watch?v=lyZQPjUT5B4`

# Bubble sort II

```
1  void bubbleSort(int vetor[], int n){//Bubble Sort
2      int i, j, aux, trocou;
3      for (i=1; i<n; i++) {
4          trocou = 0;
5          for (j=0; j<n-1; j++){
6              if (vetor[j] > vetor[j+1]){
7                  trocou = 1;
8                  aux = vetor[j];
9                  vetor[j] = vetor[j+1];
10                 vetor[j+1] = aux;
11             }
12         }
13         if(trocou==0) break;
14     }
15 }
```

# Insertion sort I

## Características

- Método utilizado por jogadores de cartas. Procura-se o local ideal de um elemento e o **INSERE** ali;
- **Recomendação de uso:** vetores/listas quase ordenadas;
- **Complexidade pior caso:** $O(n^2)$
- **Complexidade melhor caso:** $O(n)$
- **Complexidade caso médio:** $O(n^2)$
- **Divirta-se:** `https://www.youtube.com/watch?v=ROalU37913U`

# Insertion sort II

```
1  void insertionSort(int vetor[], int n){//Insertion sort
2      int min, aux, j;
3      for(int i=1; i<n; i++){
4          aux = vetor[i];
5          j = i-1;
6          while(j>=0 && vetor[j]>aux){
7              vetor[j+1] = vetor[j];
8              j = j-1;
9          }
10         vetor[j+1] = aux;
11     }
12 }
```

# Selection sort I

## Características

- Selecione o menor elemento do vetor e troque-o com a primeira posição dele;
- **Recomendação de uso:** conjuntos pequenos de elementos;
- **Complexidade pior caso:** $O(n^2)$
- **Complexidade melhor caso:** $O(n^2)$
- **Complexidade caso médio:** $O(n^2)$
- **Divirta-se:** `https://www.youtube.com/watch?v=Ns4TPTC8whw`

# Selection sort II

```
1  void selectionSort(int vetor[], int n){//Selection sort
2      int min, aux;
3      for(int i=0; i<n; i++){
4          min = i;
5          for(int j=i+1; j<n; j++){
6              if(vetor[j]<vetor[min]) min = j;
7          }
8          aux = vetor[i];
9          vetor[i] = vetor[min];
10         vetor[min] = aux;
11     }
12 }
```

# Métodos sofisticados I

## Definição

- Os métodos sofisticados de ordenação são aqueles que utilizam-se de estruturas de dados adicionais para auxiliar na ordenação ou até mesmo estratégia de dividir para conquistar, **recursão**!

## Os métodos

- Merge sort;
- Quick sort;
- Shell sort.

# Merge sort I

## Características

- Sua ideia básica consiste em Dividir e Conquistar. Como o algoritmo Merge Sort usa a recursividade, há um alto consumo de memória. Dessa forma, dependendo do problema pode não ser a melhor saída.
- **Recomendação de uso:** memória maior que $n \ log \ n$;
- **Complexidade pior caso:** $O(n \ log \ n)$
- **Complexidade melhor caso:** $O(n \ log \ n)$
- **Complexidade caso médio:** $O(n \ log \ n)$
- **Divirta-se:** `https://www.youtube.com/watch?v=XaqR3G_NVoo`

# Merge sort II

```
1 void mergeSort(int vetor[], int p, int r){//Merge sort
2     if (p<r-1){
3         int q = (p + r)/2;
4         mergeSort(vetor, p, q);
5         mergeSort(vetor, q, r);
6         intercala(vetor, p, q, r);
7     }
8 }
```

# Merge sort III

```
1  void intercala(int vetor[], int p, int q, int r){//Intercala
       vetores ordenados
2     int *aux;
3     aux = malloc ((r-p) * sizeof (int));
4     int i = p, j = q;
5     int k = 0;
6     while (i<q && j<r) {
7        if (vetor[i]<=vetor[j])  aux[k++] = vetor[i++];
8        else  aux[k++] = vetor[j++];
9     }
10    while (i < q)  aux[k++] = vetor[i++];
11    while (j < r)  aux[k++] = vetor[j++];
12    for (i = p; i < r; i++)  vetor[i] = aux[i-p];
13    free (aux);
14 }
```

# Quick sort I

## Características

- Sua ideia básica consiste em Dividir e Conquistar. Como o algoritmo Merge Sort usa a recursividade, há um alto consumo de memória. Dessa forma, dependendo do problema pode não ser a melhor saída.
- **Recomendação de uso:** memória maior que $n\ log\ n$;
- **Complexidade pior caso:** $O(n^2)$
- **Complexidade melhor caso:** $O(n\ log\ n)$
- **Complexidade caso médio:** $O(n\ log\ n)$
- **Divirta-se:** `https://www.youtube.com/watch?v=ywWBy6J5gz8`

# Quick sort II

```
1 void quickSort(int vetor[], int esq, int dir){
2    if(esq < dir){
3       int j = separa (vetor, esq, dir);
4       quickSort(vetor, esq, j-1);
5       quickSort(vetor, j+1, dir);
6    }
7 }
```

# Quick sort III

```
1  int separa(int vetor[], int esq, int dir){
2      int pivo = vetor[dir];
3      int t, j = esq;
4      for(int k=esq; k<dir; k++){
5          if(vetor[k] <= pivo){
6              t = vetor[j];
7              vetor[j] = vetor[k];
8              vetor[k] = t;
9              j++;
10         }
11     }
12     t = vetor[j];
13     vetor[j] = vetor[dir];
14     vetor[dir] = t;
15     return j;
16 }
```

# Shell Sort I

## Características

- É um refinamento do método de inserção clássico. O algoritmo é diferente do método de inserção clássico pelo fato de no lugar de considerar o vetor a ser ordenado como um único segmento, ele considera vários segmentos sendo aplicado o método de inserção clássico em cada um deles.
- **Recomendação de uso:** memória maior que $n\ log\ n$;
- **Complexidade pior caso:** $O(n\ log\ n)$
- **Complexidade melhor caso:** $O(n\ log\ n)$
- **Complexidade caso médio:** depende da sequência do *gap*
- **Divirta-se:** `https://www.youtube.com/watch?v=CmPA7zE8mx0`

# Shell Sort II

**Tente você mesmo!**

- Pesquise sobre o Shell Sort e o implemente;
- Faça testes com variações de *gap* e observe o comportamento!

# Exercícios I

1. Adapte todos os algoritmos aqui citados para fazer ordenação em ordem decrescente;
2. Implemente o método *bubble sort* para fazer ordenação em listas encadeadas simples;
3. Crie um programa que gere um vetor inteiro de números aleatórios com um **milhão de elementos**. Depois execute alguns testes com todos algoritmos visto e discorra sobre o que foi observado.

# Referências I


1. **Deitel H. M., e Deitel P.J; "C: Como programar". 6.ed. Pearson Prentice Hall, 2011. 818p.**
2. Jean Paul Tremblay & Richard P. Bunt. "Ciência dos Computadores - Uma abordagem algorítmica". McGraw-Hill.
3. Jaime Evaristo. "Aprendendo a Programar / Programando em Turbo Pascal". Edufal - Editora da Univ. Federal de Alagoas. Maceió, 1996.
4. Harry Farrer et al. "Pascal Estruturado (da série "Programação Estruturada de Computadores")". Editora Guanabara Dois. Rio de Janeiro, 1985.
5. Stephen O'Brien. "Turbo Pascal 6 Completo e Total". Makron Books.

# Referências II


6. Celes, W., Cerqueira, R., Rangel, J.L. "Introdução a Estrutura de Dados". Elsevier, 2004.
7. Feofiloff, P. "Algoritmos em Linguagem C". Elsevier, 2009. 208p.
8. Schildt, H. "C Completo e Total". 3ª ed. Pearson. 1996. 852p.