



INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
**PIAUÍ**  
CAMPUS PICOS

# ESTRUTURA DE DADOS

MAÍLA DE LIMA CLARO

claromaila@gmail.com

# BUSCA E ORDENAÇÃO EM VETORES

---

- **Busca:** Dada uma coleção de  $n$  elementos, pretende-se saber se um determinado elemento “valor” está presente nessa coleção. Para efeitos práticos, vamos supor que essa coleção é implementada como sendo um vetor de  $n$  elementos inteiros: `vetor[0]..vetor[n-1]`.
- **Pesquisa sequencial:** Uma solução possível é percorrer o vetor desde a primeira posição até a última. Para cada posição  $i$ , comparamos `vetor[i]` com `valor`.
- Se forem iguais dizemos que `valor` existe.
- Se chegarmos ao fim do vetor sem sucesso dizemos que `valor` não existe.

# BUSCA E ORDENAÇÃO EM VETORES

---

- **Pesquisa sequencial:**
- 1º passo — inicialização
  - $i = 0$ ;
  - `encontrado = 0; /*Falso*/`
- 2º passo — pesquisa
- `while (i < TAMANHO && !encontrado) {`
  - `if (vetor[i] == valor) {`
    - `encontrado = 1; /*Verdadeiro*/`
  - `} else {`
  - `i++;`
- `}`
- `}`

# BUSCA E ORDENAÇÃO EM VETORES

---

- **Pesquisa sequencial:**

- 3º passo — tratamento do resultado

```
if (encontrado) {  
    printf ("Valor %d está na posicao %d\n",  
        vetor[i], i);  
} else {  
    printf ("Valor %d não encontrado\n", valor);  
}
```

# PESQUISA SEQUENCIAL

---

- Quanto tempo este algoritmo demora a executar?
- Em outras palavras, quantas vezes a comparação `valor == vetor[i]` é executada?
- caso valor não esteja presente no vetor,  $n$  vezes.
- caso valor esteja presente no vetor,
  - 1 vez no melhor caso (valor está na primeira posição).
  - $n$  vezes no pior caso (valor está na última posição).
  - $2/n$  vezes no caso médio.

# PESQUISA SEQUENCIAL

---

- Dados a lista de convidados de uma festa e o nome de uma pessoa, determinar se essa pessoa é ou não convidada da festa. Codifique um programa completo para resolver esse problema. Crie um procedimento para fazer a entrada da lista de convidados e crie uma função `pertence()`, que retorna um valor `int`, para verificar se o nome consta ou não da lista.

# PESQUISA BINÁRIA

---

- Vamos supor agora que o vetor inicial estava ordenado por ordem crescente (se fosse por ordem decrescente o raciocínio era semelhante). Será que é possível resolver o problema de modo mais eficiente?
- Caso a lista esteja ordenada, sabemos que, para qualquer  $i$  e  $j$ ,  $i < j$ , se, e somente se,  $A[i] \leq A[j]$ .
- Portanto, comparando um determinado elemento com o elemento procurado, saberemos
  - se o elemento procurado é o elemento comparado,
  - se ele está antes do elemento comparado ou
  - se está depois.

# PESQUISA BINÁRIA

---

- Se fizermos isso sempre com o elemento do meio da lista, a cada comparação dividiremos a lista em duas, reduzindo nosso tempo de pesquisa.
- Se em um determinado momento o vetor, após sucessivas divisões, tiver tamanho zero, então o elemento não está no vetor.
- Quanto tempo o algoritmo de busca binária demora a executar?
- Por outras palavras, quantas vezes a comparação `valor == vetor[i]` é executada?



# PESQUISA BINÁRIA

---

- caso valor não exista no vetor,  $\log_2(n)$  vezes.
- caso valor exista no vetor,
  - 1 vez no melhor caso (valor é a mediana do vetor).
- $\log_2(n)$  vezes no caso médio.

## QUAL DOS DOIS ALGORITMOS É MELHOR?

---

- Para  $n = 1000$ , o algoritmo de pesquisa sequencial irá executar 1000 comparações no pior caso, 500 operações no caso médio.
- Por sua vez, o algoritmo de pesquisa binária irá executar 10 comparações no pior caso, para o mesmo  $n$ . O logaritmo de base 2 aparece porque estamos sempre a dividir o intervalo ao meio: 1000, 500, 250, 125, 63, 32, 16, 8, 4, 2, 1.
- O algoritmo de pesquisa binária assume que o vetor está ordenado. Ordenar um vetor também tem um custo, superior ao custo da pesquisa sequencial.
- Se for para fazer uma só pesquisa, não vale a pena ordenar o vetor. Por outro lado, se pretendermos fazer muitas pesquisas, o esforço da ordenação já poderá valer a pena.

# ORDENAÇÃO

---

- Dada uma coleção de  $n$  elementos, representada em um vetor de 0 a  $n-1$ , deseja-se obter uma outra coleção, cujos elementos estejam ordenados segundo algum critério de comparação entre os elementos.
- Iremos ver:
  - Ordenação por Inserção;
  - Ordenação por Seleção
  - Ordenação por Bolha.

# ORDENAÇÃO POR INSERÇÃO

---

- A proposta da ordenação por inserção é:
- Para cada elemento do vetor faça:
- insira-o na sua posição correspondente;
- Durante o processo de ordenação por inserção a lista fica dividida em duas sub-listas, uma com os elementos já ordenados e a outra com elementos ainda por ordenar.
- No início, a sub-lista ordenada é formada trivialmente apenas pelo primeiro elemento da lista.
- Nos métodos de ordenação por inserção, a cada etapa  $i$ , o  $i$ -ésimo elemento é inserido em seu lugar apropriado entre os  $i - 1$  elementos já ordenados. Os índices dos itens a serem inseridos variam 2 a *tamanho*.

# ORDENAÇÃO POR INSERÇÃO

---

Varredura	V[0]	V[1]	V[2]	V[3]	V[4]
Vetor Original	9	8	7	6	5
1	{8	9}	{7	6	5}
2	{7	8	9}	{6	5}
3	{6	7	8	9}	{5}
4	{5	6	7	8	9}

# ORDENAÇÃO POR INSERÇÃO

---

- Em cada etapa  $i$  são executadas no máximo,  $i$  comparações pois o loop interno é executado para  $j$  de  $i - 1$  até  $0$ . Como  $i$  varia de  $0$  até  $\text{tamanho}-1$ , o número total de comparações para ordenar a lista toda é da ordem de  $\text{tamanho}^2$

# ORDENAÇÃO POR SELEÇÃO

---

- A ordenação por seleção consiste, em cada etapa, em selecionar o maior (ou o menor) elemento e colocá-lo em sua posição correta dentro da futura lista ordenada.
- Durante a aplicação do método de seleção a lista com  $m$  registros fica decomposta em duas sub-listas, uma contendo os itens já ordenados e a outra com os restantes ainda não ordenados. No início a sub-lista ordenada é vazia e a outra contém todos os demais. No final do processo a sub-lista ordenada apresentará tamanho  $-1$  itens e a outra apenas 1.
- Cada etapa (ou varredura) como já descrito acima consiste em buscar o maior elemento da lista não ordenada e colocá-lo na lista ordenada.

# ORDENAÇÃO POR SELEÇÃO

---

Etapa	V[0]	V[1]	V[2]	V[3]	V[4]
Vetor Original	5	9	1	4	3
1	{5	3	1	4}	{9}
2	{4	3	1}	{5	9}
3	{1	3}	{4	5	9}
4	{1}	{3	4	5	9}



# ORDENAÇÃO POR SELEÇÃO

---

- A comparação é feita no loop mais interno. Para cada valor de  $i$  são feitas  $\text{tamanho} - i$  comparações dentro do loop mais interno. Como  $i$  varia de 0 até  $\text{tamanho} - 1$ , o número total de comparações para ordenar a lista toda é, novamente, da ordem de  $\text{tamanho}^2$ .

# ORDENAÇÃO POR BOLHA

---

- Um método simples de ordenação por troca é a estratégia conhecida como bolha que consiste, em cada etapa “borbulhar” o maior elemento para o fim da lista. Inicialmente percorre-se a lista dada da esquerda para a direita, comparando pares de elementos consecutivos, trocando de lugar os que estão fora da ordem. No exemplo abaixo, em cada troca, o maior elemento é deslocado uma posição para a direita.

# ORDENAÇÃO POR BOLHA

---

Varredura	V[0]	V[1]	V[2]	V[3]	Troca
I	10	9	7	6	V[0] e V[1]
	9	10	7	6	V[1] e V[2]
	9	7	10	6	V[2] e V[3]
	9	7	6	10	Fim da Varredura I

# ORDENAÇÃO POR BOLHA

- Após a primeira varredura o maior elemento encontra-se alocado em sua posição definitiva na lista ordenada. Podemos deixá-lo de lado e efetuar a segunda varredura na sub-lista  $V[0], V[1], V[2]$ . Veja a continuação do exemplo:

Varredura	V[0]	V[1]	V[2]	V[3]	Troca
2	9	7	6	10	V[0] e V[1]
	7	9	6	10	V[1] e V[2]
	7	6	9	10	Fim da Varredura I

# ORDENAÇÃO POR BOLHA

- Após a segunda varredura o maior elemento da sub-lista  $V[0], V[1], V[2]$  encontra-se alocado em sua posição definitiva. A próxima sub-lista a ser ordenada é  $V[0], V[1]$ . Veja a continuação do exemplo:

Varredura	V[0]	V[1]	V[2]	V[3]	Troca
3	7	6	9	10	V[0] e V[1]
	6	7	9	10	Fim da Varredura I

# ORDENAÇÃO POR BOLHA

---

- No algoritmo da bolha observamos que existem tamanho-1 varreduras (etapas) e que em cada varredura o número de comparações diminui de uma unidade, variando de tamanho-1 até 1. Portanto, para ordenar a lista toda novamente precisamos de aproximadamente  $\text{tamanho}^2$  comparações.



CONTATO:

**Maíla de Lima Claro**

(claromaila@gmail.com)