

Estrutura de Dados II

Algoritmos de Ordenação:

- Bubble Sort (Bolha)
- Count Sort (Contagem)

```
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello world!" << endl;
    return 0;
}
```

Relembrando... Sessão Revisão



Revisando

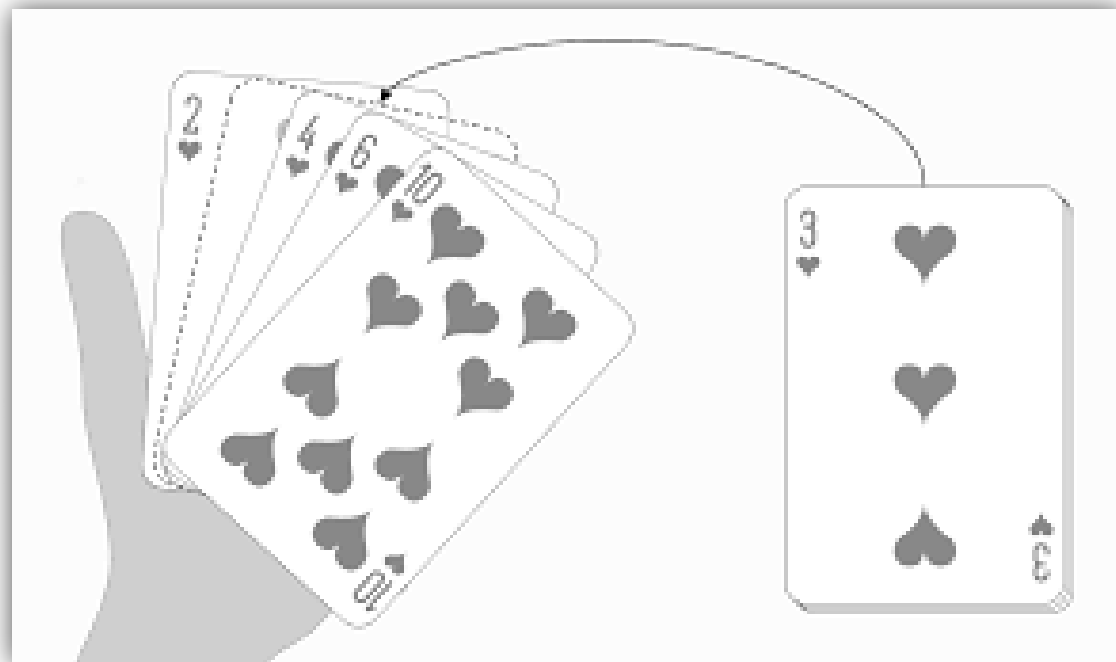
- **Pesquisa Sequencial** compara cada item de um conjunto de elementos.
- **Pesquisa binária** verifica se o elemento está no centro, se está na parte inferior ou na parte superior no conjunto de elementos.
- **Na pesquisa binária**, a cada execução é eliminado a metade do conjunto até finalizar o conjunto inteiro.

Revisando

- “**Pesquisar**” é (deve ser) uma operação rotineira em um conjunto de dados.
- **Pesquisas eficientes** exigem **Dados Ordenados** (ou estruturas ordenadas, índices por exemplo).
- **Ordenar Dados** exige **processamento extra**.

Ordenação de Dados

Algoritmos de Ordenação de Dados



Algoritmos de Ordenação

- Ordenação de Dados é um processo que coloca um conjunto de dados em uma certa ordem, ou seja, realiza a ordenação (classificação) dos dados.
- Durante a ordenação de dados podem ser usadas as o formato numérico ou alfabético (ou alfanumerico) como critério de ordenação. Pode também ser usado o formato Data (temporal) ou outros atributos definidos (ex. tamanho, etc)

Algoritmos de Ordenação

Vantagens:

- Um conjunto de dados ordenados possibilita o acesso dos seus dados de modo mais eficiente (velocidade).

Desvantagens:

- Requer **processamento extra para ordenação** e criação de mecanismos e algumas vezes mecanismos externos (índices) para manter a ordenação.

Algoritmos de Ordenação

O que diferencia um algoritmo do outro?

Basicamente e Eficiência: Tempo que leva para ordenar os dados, os recursos gastos (processamento) e o tipo de dado.

Os algoritmos são classificados pela sua complexidade:

- * $O(n)$ – n vezes
- * $O(n^2)$ – $n * n$ vezes (tempo quadrático)
- * $(\log n)$ logarítmica (diminuem a cada instância executada)



Algoritmos de Ordenação

Estáveis x Não Estáveis

Um algoritmo de ordenação é considerado estável quando consegue preservar a ordem de registro de chaves iguais.

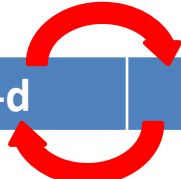
3-a	2-c	2-d	1-e	4-f
-----	-----	-----	-----	-----

Obrigatoriamente o resultado será:

1-e	2-c	2-d	3-a	4-f
-----	-----	-----	-----	-----

Os algoritmos não estáveis sujeitam os elementos associados aos objetos a serem ordenados:

1-e	2-d	2-c	3-a	4-b
-----	-----	-----	-----	-----



Algoritmos de Ordenação

Ordenação Interna x Ordenação Externa

Ordenação Interna:

Todos os elementos a serem ordenados cabem na memória. Os registros podem ser acessados diretamente.

Ordenação Externa:

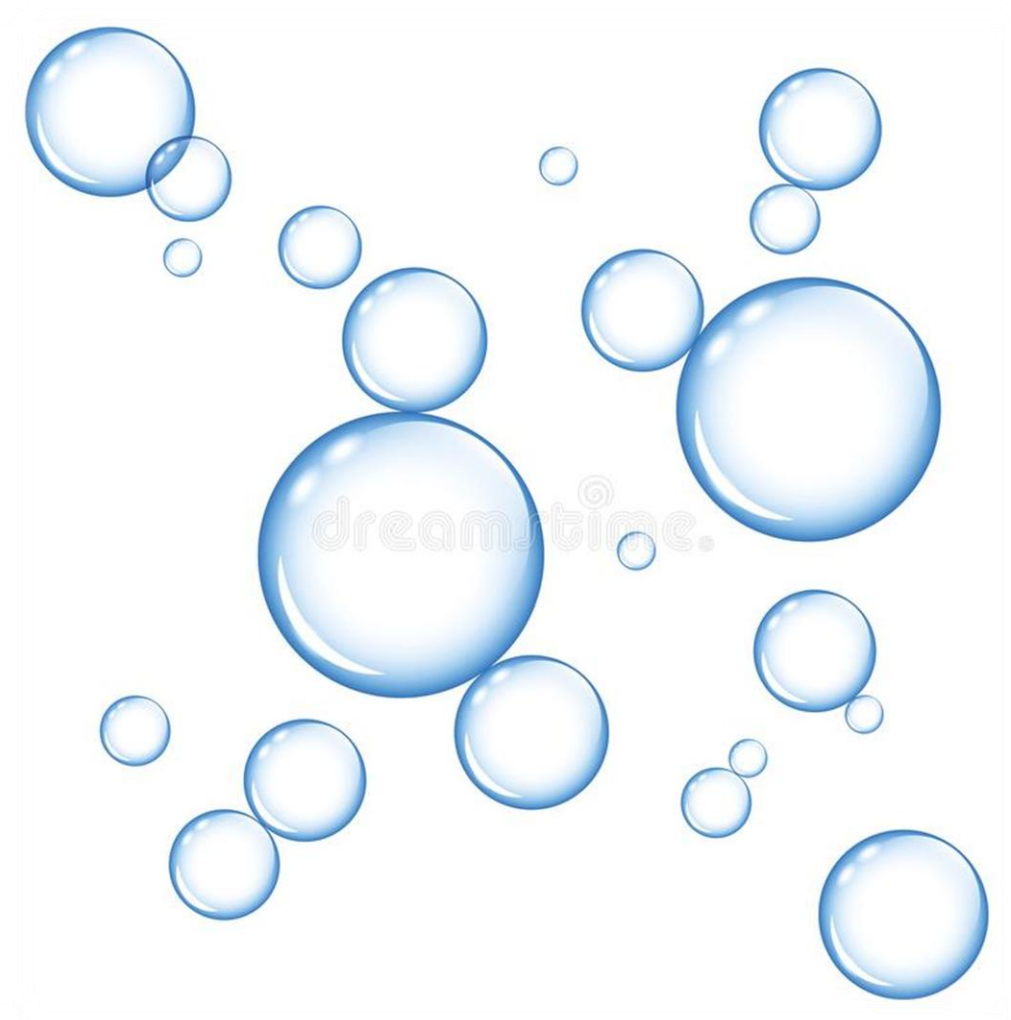
Os elementos não cabem na memória principal e os registros são acessados sequencialmente ou em blocos.

Algoritmos de Ordenação

Alguns Algoritmos de Ordenação:

- Bubblesort
- Ordenação por Contagem
- Ordenação por Inserção
- Mergesort
- Quicksort
- Ordenação por Seleção

ORDENAÇÃO POR BOLHAS



Algoritmo de Ordenação Bubblesort

Algoritmo O ***bubble sort***, ou ordenação por flutuação (literalmente "por bolha") é um dos algoritmos de ordenação mais simples.

Em uma estrutura de dados desordenada inicia-se o algoritmo pelo primeiro elemento, depois faz-se a comparação dele com todos os que estão depois dele na estrutura desordenada.

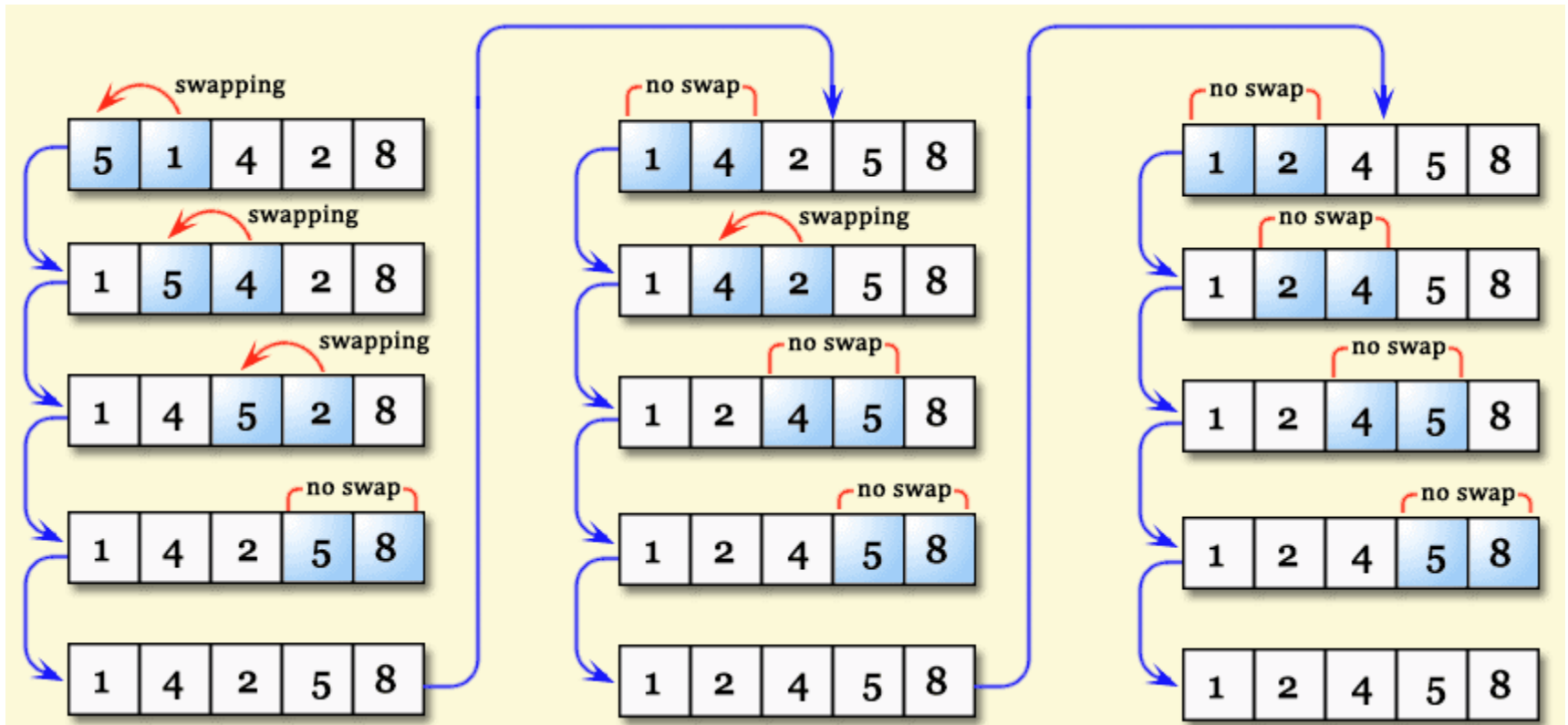
Algoritmo de Ordenação Bubblesort

A ideia é percorrer o vetor diversas vezes, a cada passagem fazendo flutuar para o topo o maior elemento da sequência.

Essa movimentação lembra a forma como as bolhas em um tanque de água procuram seu próprio nível, e disso vem o nome do algoritmo.



Bubblesort



Algoritmo de Ordenação Bubblesort

```
#include <stdio.h>
#include <stdlib.h>

void main ()
{
    //Inicialização do Vetor
    int v[5]= {1,7,4,3,5}, n=5;
    int i, j = 0, aux;

    //Algoritmo de Ordenação
    while (j < n)
    {
        for(i = 0; i < n-1; i++)
            if(v[i] > v[i + 1])
            {
                aux=v[i];
                v[i] = v[i+1];
                v[i+1] = aux;
            }
        j++;
    }

    //Laço de impressão do Vetor
    for (int q=0; q<5; q++)
    {
        printf("%d \n",v[q]);
    }
    system("pause"); }
```


Algoritmo de Ordenação Bubblesort

- Quantas **iterações** foram necessárias para ordenar um conjunto de 5 elementos?
- Em um conjunto de 50 elementos, quantas iterações serão necessárias?



ORDENAÇÃO POR CONTAGEM



Algoritmo de Ordenação por Contagem

Esse algoritmo pode ser o mais rápido em vários casos, pois ele ordena os dados em **tempo linear**.

Essa ordenação pressupõe que os dados serão sempre uma entrada de **1 a n**, para qualquer inteiro **n**, ou seja, se terão 6 dígitos, a entrada precisa ser qualquer combinação possível de **6 números** que variam entre si de um a 6.

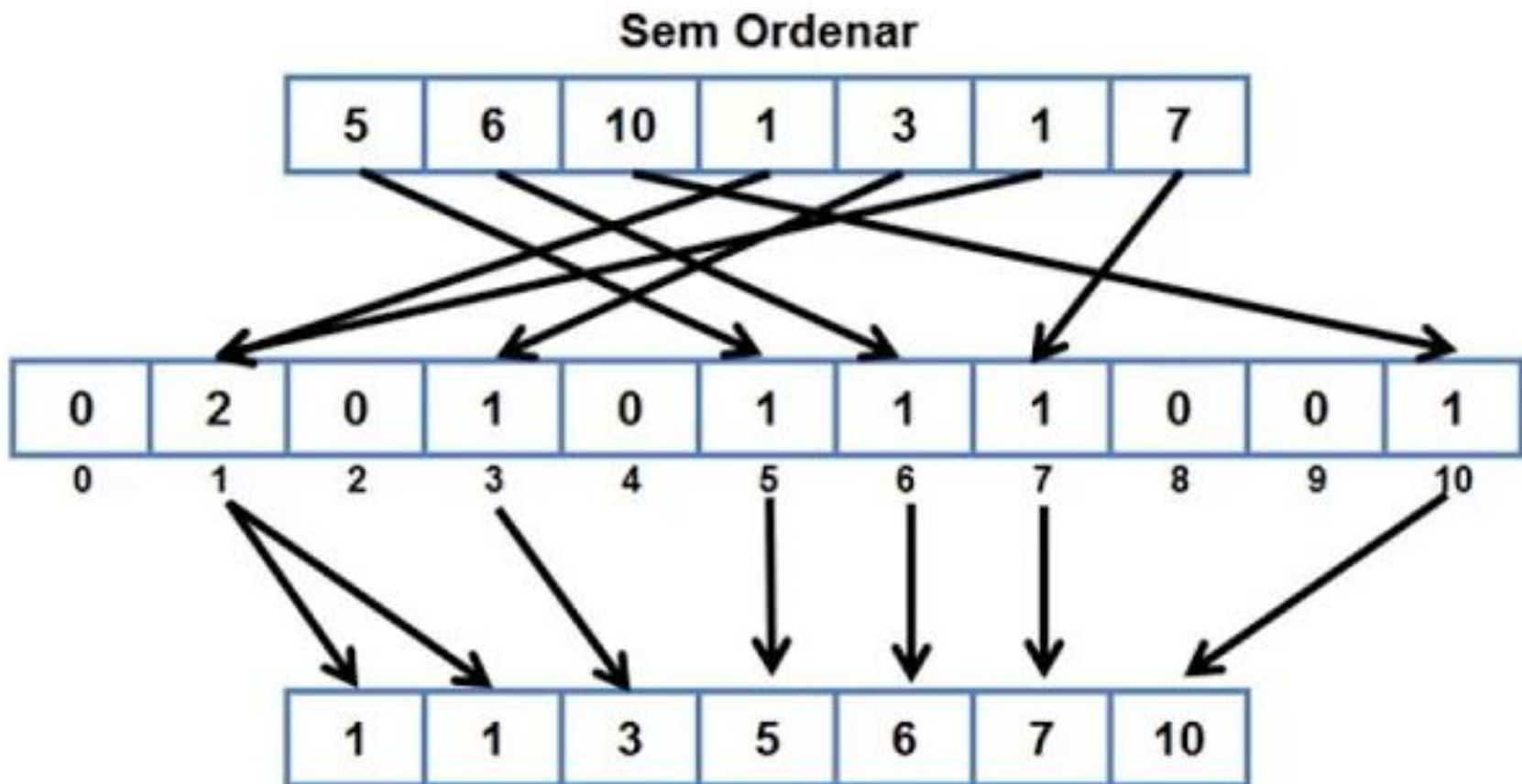
Algoritmo de Ordenação por Contagem

A base matemática do algoritmo está em determinar para cada elemento x de entrada o número de elementos menores que x , e depois inserir diretamente x na posição de saída na estrutura.

Exemplo: *O índice indica o elemento e o valor da posição o numero de vezes que ele aparece.*



Ordenação por Contagem



Algoritmo de Ordenação por Contagem

```
#include <stdio.h>
#include <stdlib.h>
```

```
main ()
{
```

```
//Inicialização do Vetor
int v[5]= {1,7,4,3,5};
int n=5;
int i, j = 0, aux;
```

```
//Algoritmo de Ordenação
counting_sort(v,n);
```

```
//Laço de impressão do Vetor
for (int q=0; q<5; q++)
{
    printf("%d\n",v[q]);
}
}
```

```
void counting_sort(int *array, int size)
{
    int i, min, max;
    min = max = array[0];

    //Identifica o Maior Elemento
    for(i = 1; i < size; i++)
    {
        if (array[i] < min)
            min = array[i];
        else if (array[i] > max)
            max = array[i]; }
    int range = max - min + 1;
    int *count = (int *) malloc(range * sizeof(int));

    //Marca Todas as posições com Zero
    for(i = 0; i < range; i++)
        count[i] = 0;

    //Marca as posições ocupadas
    for(i = 0; i < size; i++)
        count[array[i] - min]++;

    int j, indice = 0;
    //Array recebe as posicoes ocupadas
    for(i = min; i <= max; i++)
        for(j = 0; j < count[ i - min ]; j++)
            array[indice++] = i;
    free(count);
}
```

Algoritmo de Ordenação Contagem

- Quantas **iterações** foram necessárias para ordenar um conjunto de 5 elementos?
- Em um conjunto de 50 elementos, quantas iterações serão necessárias?



Laboratório

- Execute o Programa “Gera_RAND” para criar um arquivo com 10.000 número randômicos.
- Altere o Programa do Algoritmo BubbleSort para que leia o arquivo e gere um arquivo ordenado.
- Altere o Programa do Algoritmo por Contagem para que leia o arquivo e gere um arquivo ordenado.
- **Qual dos dois foi mais rápido?**
- **Quantas interações cada um teve?**




```
1 #include <time.h>
2 #include <unistd.h>
3 #include <stdlib.h>
4 #include <stdio.h>
5 main ()
6 {
7     //Inicializaçã do Vetor
8     FILE *txt;
9     txt = fopen("Arquivo.txt", "r");
10    float v[100000], n=100000;
11    float j = 0, aux;
12    long int i=0;
13
14    printf("\nCarregando o Arquivo no Vetor");
15    while (i< n)
16    {
17        fscanf(txt, "%f",&v[i]);
18        i++;
19    }
20    fclose(txt);
21
22    //Algoritmo de Ordenação
23    //aqui
24
25    printf("\nImprimindo Vetor Ordenado");
26    txt = fopen("Arquivo_ordenado.txt", "w");
27
28    //Laço de impressão do Vetor
29    for (i=0; i<n; i++)
30    {
31        fprintf(txt, "%.0f\n",v[i]);
32    }
33    fclose(txt);
34 }
35
36
37
```