Algoritmos de Ordenação

Sumário

- Introdução aos algoritmos de ordenação
- Ordenação por inserção
- Ordenação por seleção
- Outros algoritmos de ordenação

Introdução aos Algoritmos de Ordenação

O que é ordenação?

É a tarefa de colocar um conjunto de dados em uma determinada ordem.

Qual a utilidade da ordenação?

Permite o acesso mais eficiente aos dados.

Ou seja, um algoritmo de ordenação coloca os dados de uma sequência fornecida em uma certa ordem.

Introdução aos Algoritmos de Ordenação

Os tipos de ordenação mais utilizados são:

- Numérica (1, 2, 3, 4, 5)
- Lexicográfica (ordem alfabética)

Podendo ambas serem crescente ou decrescente:

- Numérica crescente / decrescente = 1, 2, 3, 4, 5 / 5, 4, 3, 2, 1
- Lexicográfica crescente / decrescente = a, b, c, d / d, c, b, a

Introdução aos Algoritmos de Ordenação

Classificação dos métodos de ordenação

Ordenação interna:

- O arquivo a ser ordenado cabe todo na memória principal e qualquer registro pode ser imediatamente acessado.

Ordenação externa:

- O arquivo a ser ordenado não cabe na memória principal, ou seja, esse arquivo vai ser acessado por partes.

Introdução aos Algoritmos de Ordenação

Alguns algoritmos em questão de complexidade:

- **O(n log n):** Quicksort e Mergesort.

São algoritmos com grande eficiência para arrays de tamanho ***n*** grande.

- **O(n2):** Bubblesort , Insertion Sort e Selection Sort.

São algoritmos com menor complexidade, porém mais eficientes para arrays de tamanho ***n*** pequeno.

Ordenação por Inserção

– Primeiro, consideramos o vetor dividido em dois sub vetores (esquerdo e direito), com o da esquerda **ordenado** e o da direita **desordenado**.

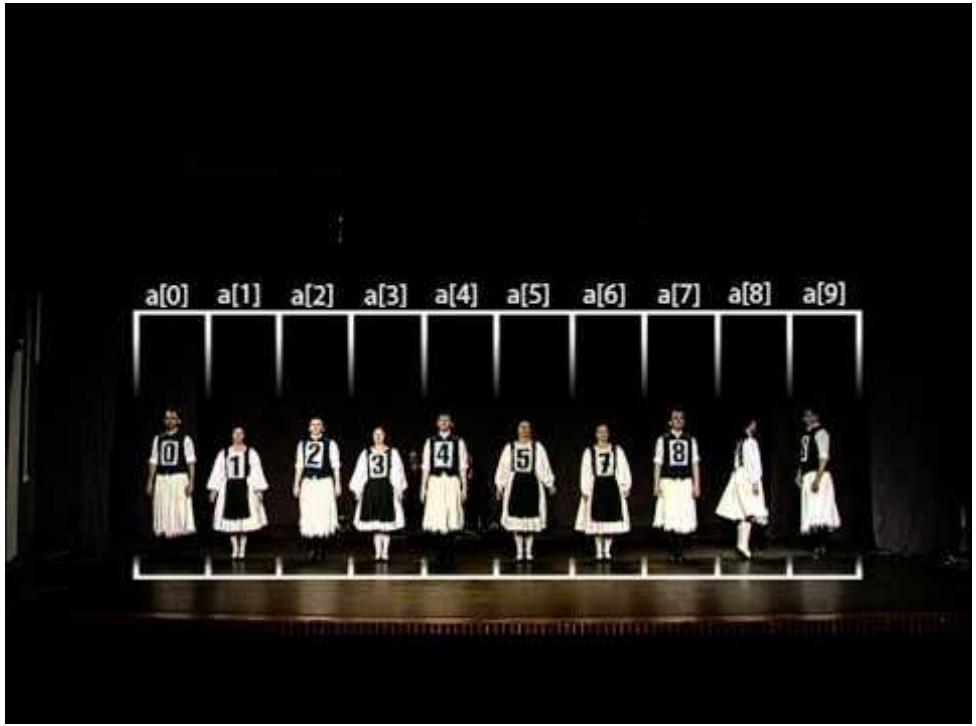
– Começa com um elemento apenas no sub vetor da esquerda, e os demais no da direita.

– Passa-se um elemento de cada vez do sub vetor da direita para o sub vetor da esquerda, inserindo-o na posição correta com o objetivo de manter o sub vetor da esquerda ordenado.

– Termina quando o sub vetor da direita (desordenado) fica vazio.

Ordenação por Inserção

Ordenação por Inserção



Ordenação por Inserção

```
#include <iostream>
using namespace std;

void InsertionSort(int a[],int tam){
    int i, j, aux;
    for(i=0;i<tam;i++){
        j=i;
        while(j>0 && a[j-1] > a[j]){
            aux = a[j-1];
            a[j-1] = a[j];
            a[j] = aux;
            j--;
        }
    }
}


int main() {
    int a[5],cont;
    for(cont=0;cont<5;cont++){
        cin >> a[cont];
    }
    InsertionSort(a,5);
    for(cont=0;cont<5;cont++){
        cout << a[cont] << " ";
    }
    return 0;
}
```

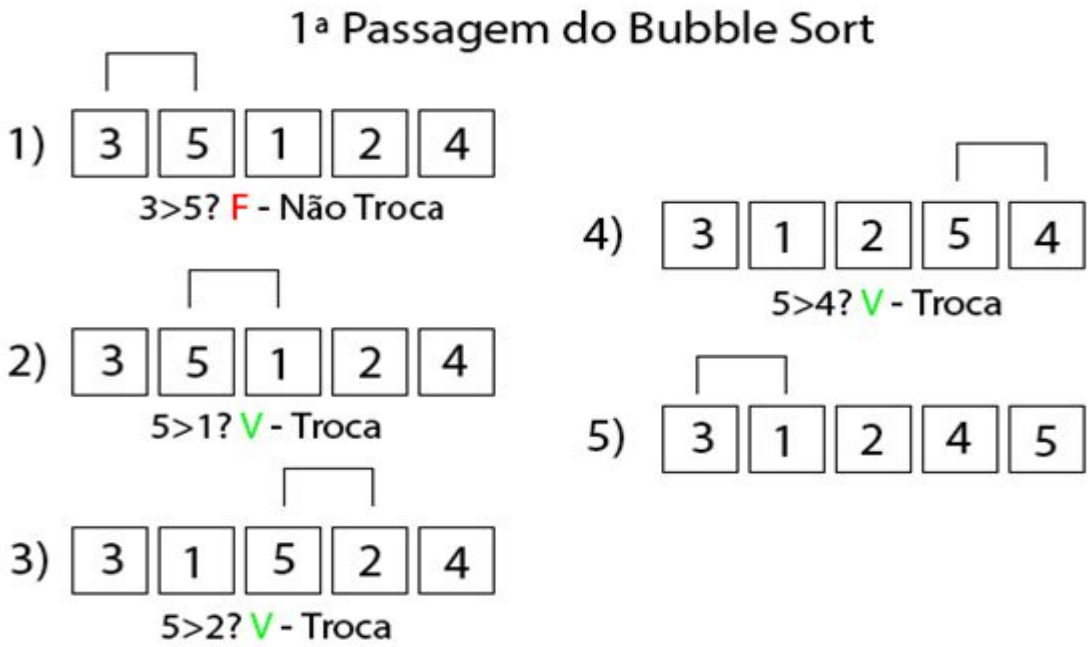
Melhor Caso : O (N)
Pior Caso: O (N ^2)

BubbleSort

Bubble sort é o algoritmo mais simples, mas o menos eficientes. Neste algoritmo cada elemento de uma posição i qualquer, será comparado com o elemento da posição i + 1, ou seja, um elemento da posição 2 será comparado com o elemento da posição 3. Caso o elemento da posição 2 for maior que o da posição 3, eles trocam de lugar e assim sucessivamente. Por causa desse método de ordenação o vetor será percorrido inúmeras vezes tornando assim o algoritmo não muito eficiente para listas grandes.

Melhor Caso: O (N)
Pior Caso: O (N^2)

BubbleSort


1ª Passagem do Bubble Sort
1) 3 5 1 2 4
3>5? F - Não Troca
2) 3 5 1 2 4
5>1? V - Troca
3) 3 1 5 2 4
5>2? V - Troca
4) 3 1 2 5 4
5>4? V - Troca
5) 3 1 2 4 5

BubbleSort

Algoritmo

```
void BubbleSort (int vetor[], int N) {
    int i, continua, aux, fim = N;
    do {
        continua = 0;
        for (i=0; i < fim - 1; i++) {
            if (vetor[i] > vetor[i + 1]) {
                aux = vetor[i];
                vetor[i] = vetor[i + 1];
                vetor[i + 1] = aux;
                continua = i;
            }
        }
    } while (continua != 0);
}
```

Ordenação por Seleção

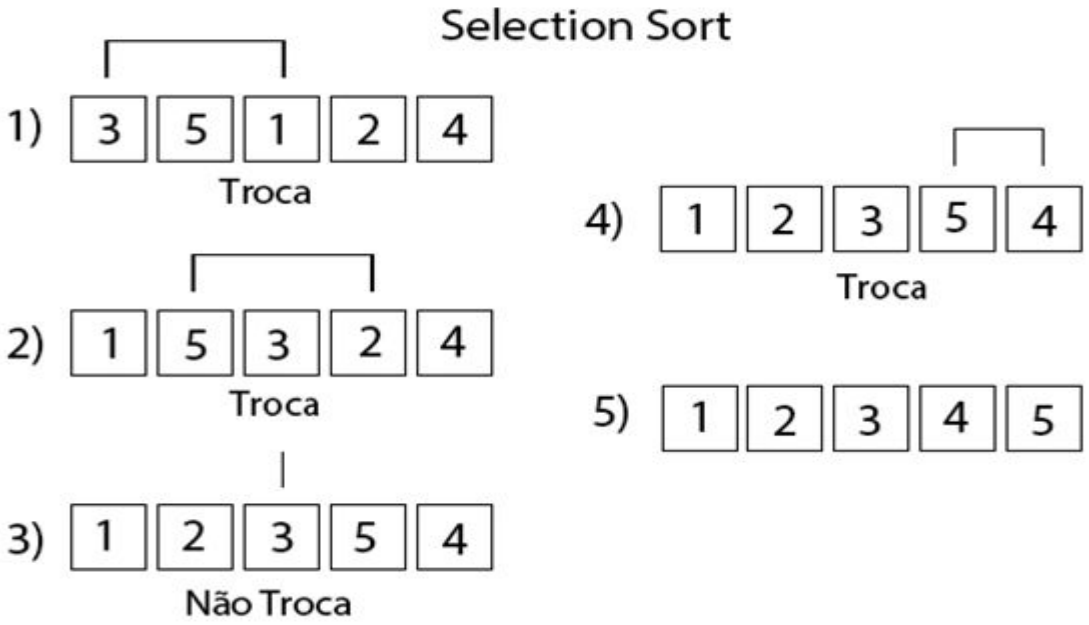
– Definir o mínimo do vetor, sendo ele o primeiro elemento no início do laço “for”.

– O termo mínimo inicial do laço (temporário) vai ser comparado com todos os elementos do array até encontrar algum que seja menor, se encontrar alguém menor, esse elemento vai se tornar o novo mínimo e vai continuar a ser comparado com os elementos seguintes a ele, até encontrar o menor valor no conjunto e realizar a troca com o termo mínimo inicial.

– Continuar para o resto do vetor (excluindo aqueles que já foram ordenados).

Melhor caso: O (N ^ 2)
Pior caso: O (N ^ 2)

Ordenação por Seleção



Ordenação por Seleção

9 25 10 18 5 7 15 3 troca(9,3).

3 25 10 18 5 7 15 9 troca(25,5).

3 5 10 18 25 7 15 9 troca(10,7).

3 5 7 18 25 10 15 9 troca(18,9).

3 5 7 9 25 10 15 18 troca(25,10).

3 5 7 9 10 25 15 18 troca(25,15).

3 5 7 9 10 15 25 18 troca(25,18).

3 5 7 9 10 15 18 25 FIM.

Ordenação por Seleção

Ordenação por Seleção

```cpp
#include <iostream>

using namespace std;

int main(){
    int vetor[5];
    int i, j, min, aux;

    for (i = 0; i < 5; i++){
        cin >> vetor[i];
    }

    for(i = 0; i < 5 - 1; i++){
        min = i;
        for (j = i+1; j < 5; j++){
            if(vetor[j] < vetor[min]){
                min = j;
            }
        }
        aux = vetor[i];
        vetor[i] = vetor[min];
        vetor[min] = aux;
    }

    for (i = 0; i < 5; i++){
        cout << vetor[i] << " ";
    }
    cout << endl;

    return 0;
}
```

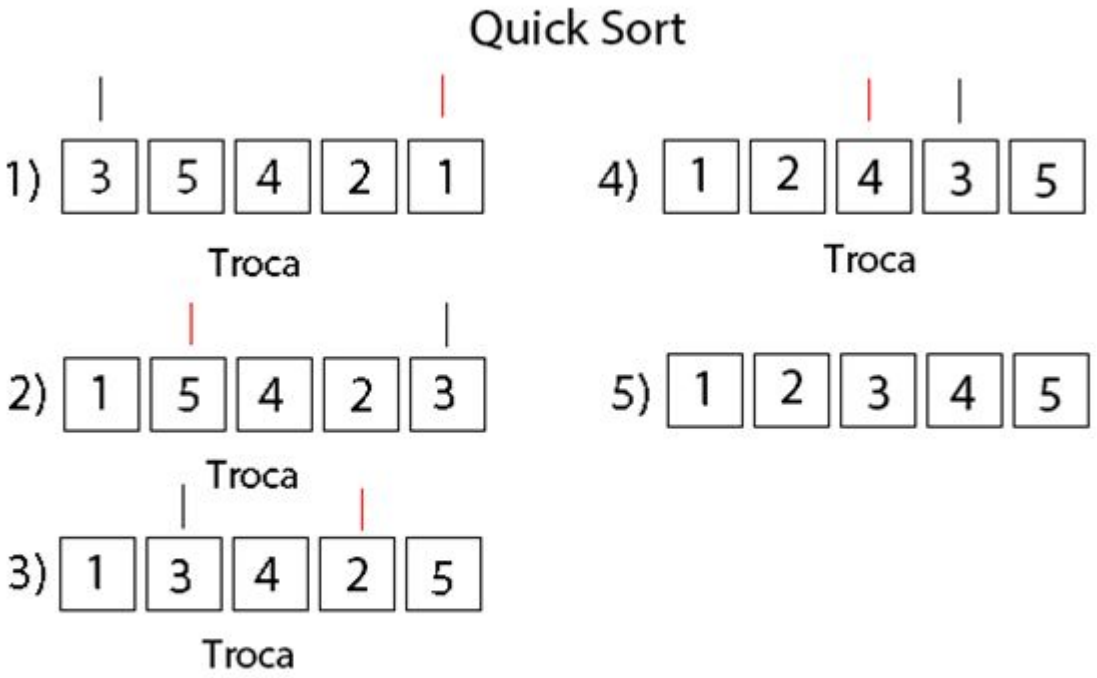
Outros Algoritmos de Ordenação

Quicksort

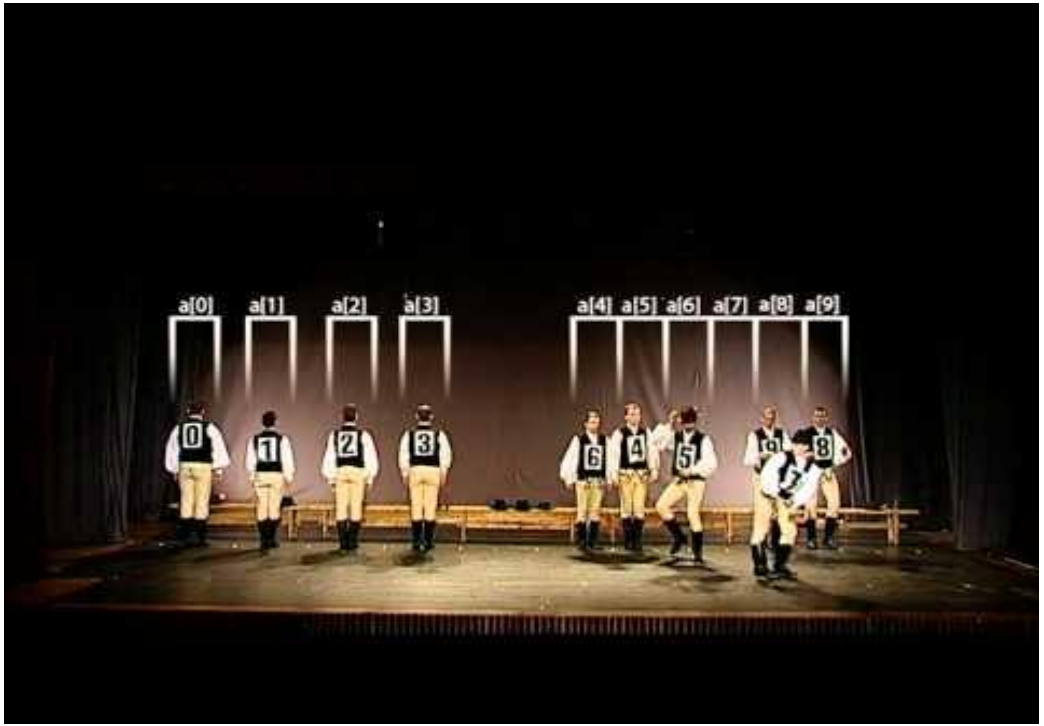
O Quicksort é o algoritmo mais eficiente na ordenação por comparação. Nele se escolhe um elemento chamado de pivô, a partir disto é organizada a lista para que todos os números anteriores a ele sejam menores que ele, e todos os números posteriores a ele sejam maiores que ele após isso os dois grupos desordenados sofrem o mesmo processo até que a lista esteja ordenada.

Melhor caso: O (N log N)
Pior caso: O (N ^ 2) (**RARO**)

Exemplo


Quick Sort
1) 3 5 4 2 1
Troca
2) 1 5 4 2 3
Troca
3) 1 3 4 2 5
Troca
4) 1 2 4 3 5
Troca
5) 1 2 3 4 5

Quicksort



Algoritmo

```
void Quick(int vetor[10], int inicio, int fim){

    int pivo, aux, i, j, meio;
    i = inicio;
    j = fim;
    meio = (int) ((i + j) / 2);
    pivo = vetor[meio];

    do{
        while (vetor[i] < pivo){
            i = i + 1;
          }
        while (vetor[j] > pivo){
            j = j - 1;
          }
        if(i <= j){
            aux = vetor[i];
            vetor[i] = vetor[j];
            vetor[j] = aux;
            i = i + 1;
            j = j - 1;
        }
    }while(j > i);

    if(inicio < j){
       Quick(vetor, inicio, j);
       }
    if(i < fim){
        Quick(vetor, i, fim);
       }
}
```

Mergesort

Esse método usa três processos para realizar a ordenação:

1) **Dividir**

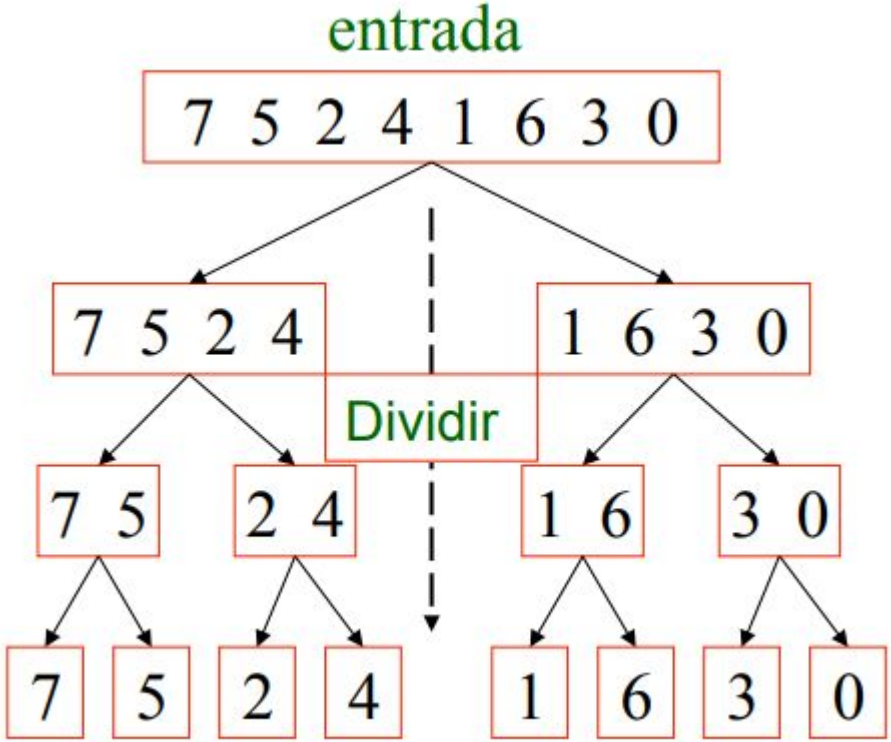
2) **Conquistar**

3) **Combinar**

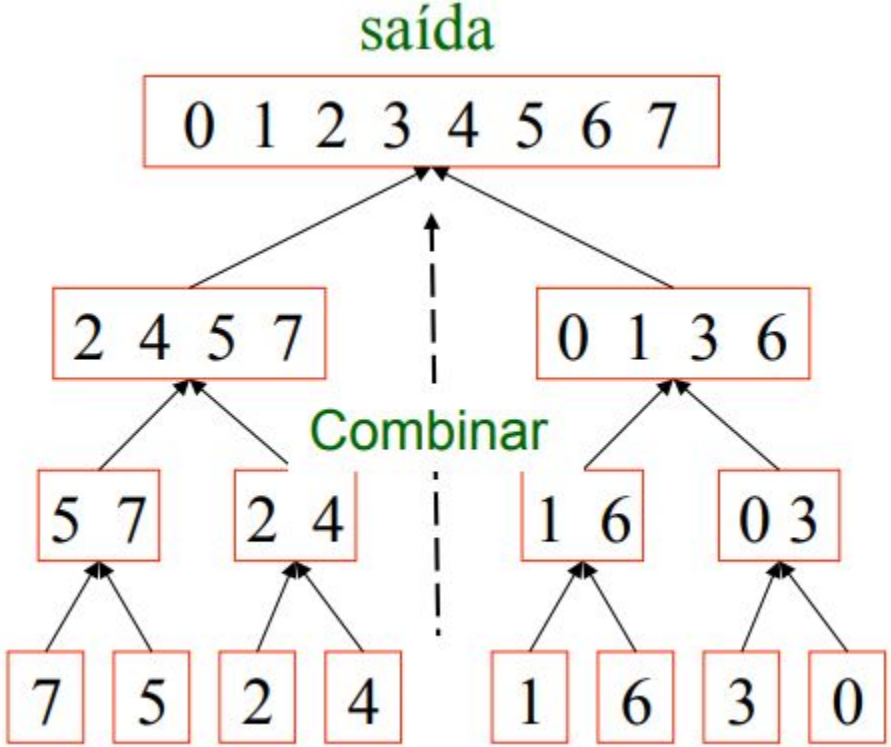
Melhor caso: O (N log N)

Pior caso : O (N log N)

Exemplo



Exemplo



Mergesort


a[0]
a[1]
a[2]
a[3]
a[4]
a[5]
a[6]
a[7]
a[8]
a[9]
2
4
0
6
8
5
1
7
3
9
b[0]
b[1]
b[2]
b[3]
b[4]
b[5]
b[6]
b[7]
b[8]
b[9]

Algoritmo

```
void mergeSort (int vetor[], int inicio, int fim) {
    int meio;
    if (inicio < fim) {
        meio = floor ((inicio + fim) / 2);
        mergeSort (vetor, inicio, meio);
        mergeSort (vetor, meio+1, fim);
        merge(vetor, inicio, meio, fim);
    }
}

void merge (int vetor[], int inicio, int meio, int fim) {
    int *temp, p1, p2, tamanho, i, j, k;
    int fim1 = 0, fim2 = 0;
    tamanho = fim - inicio + 1;
    p1 = inicio;
    p2 = meio + 1;
    temp = new int[tamanho];
    if (temp != NULL) {
        for(i=0; i < tamanho; i++) {
            if (!fim1 && !fim2) {
                if(vetor[p1] < vetor[p2])
                    temp[i] = vetor [p1++];
                else
                    temp[i] = vetor [p2++];

                if (p1 > meio) fim1 = 1;
                if (p2 > fim)  fim2 = 1;
            }else {
                if(!fim1)
                    temp[i] = vetor [p1++];
                else
                    temp[i] = vetor [p2++];
            }
        }
        for (j=0, k=incio; j < tamanho; j++, k++)
        vetor[k] = temp[j];
    }
    delete [] temp;
}
```

Eficiência de cada Algoritmo de ordenação

Tamanho do vetor: **100**

Algoritmo	**Tempo (ms)**	**N. de comp.**	**Movimentos**
BubbleSort	16, 6730	500 500	756 840
SelectionSort	5, 6664	499 500	2 997
InsertionSort	5, 7523	999	254 278
QuickSort	0, 3725	13 138	7 983

Eficiência de cada Algoritmo de ordenação

Tamanho do vetor: **10.000**

Algoritmo	**Tempo (ms)**	**N. de comp.**	**Movimentos**
BubbleSort	1455, 9734	50 005 000	74 237 889
SelectionSort	545, 1068	49 995 000	29 997
InsertionSort	539, 6891	9 999	24 765 961
QuickSort	4, 5072	176 065	103 635

Extra

http://www.cplusplus.com/

Site que contém a grande maioria das bibliotecas do C++.

Desafio Los Lakers

Los Angeles Lakers é o time da cidade na NBA. Já foi algumas vezes campeão de sua conferência e revelou vários excelentes jogadores.

Em um campeonato de basquete os times jogam todos entre si em turno único. A vitória vale dois pontos e a derrota vale um ponto (não há empates no basquete). Havendo empates na pontuação do campeonato fica na frente o time com melhor "**cesta average**" que é dado pela razão entre o número de pontos marcados pelo time dividido pelo número de pontos recebidos (na improvável hipótese de um time vencer todos os jogos do campeonato sem levar cestas seu cesta average é dado pelo número de pontos marcados). Persistindo o empate, leva vantagem quem marcou mais pontos.

Sua tarefa neste problema é fazer um programa que **recebe os resultados dos jogos de um campeonato e imprime a classificação final.**

Entrada:

São dadas várias sequências. Para cada sequência é dada o número $0 \leq \mathbf{n} \leq 100$ de times no campeonato. O valor **n** = 0 indica o fim dos dados. A seguir vêm **n (n−1) / 2** linhas indicando os resultados das partidas. Em cada linha são dados quatro inteiros **x**, **y**, **z** e **w**. Os inteiros **x** e **z** pertencem ao conjunto {1, 2, . . . , n} e representam os números de inscrição dos times na liga. Os inteiros **y** e **w** são, respectivamente, os números de pontos do time **x** e do time **z** na partida descrita.

Desafio Los Lakers

Entrada:

```
5
1 102 2 62
1 128 3 127
1 144 4 80
1 102 5 101
2 62 3 61
2 100 4 80
2 88 5 82
3 79 4 90
3 87 5 100
4 110 5 99
0
```

Saída:

```
Instancia 1
1 2 4 5 3
```