

MCZA038

Prática Avançada de Programação A

Ordenação

Prof. Alexandre Donizeti Alves



Universidade Federal do ABC

**Bacharelado em Ciência da
Computação**

Terceiro Quadrimestre - 2017

Introdução

- Os **algoritmos de ordenação** são fundamentais na área da computação e estão presentes em uma grande quantidade de problemas de programação
- Diversos algoritmos de ordenação têm sido desenvolvidos, sendo cada um aplicado a um **tipo específico de dados**

Aplicações

- **Como testar se todos os elementos de um conjunto S são únicos?**
 1. Ordene o conjunto – $O(n \log n)$
 2. Percorra o conjunto uma única vez buscando por repetições entre os elementos adjacentes $O(n)$

Aplicações

■ Remover duplicações de um elemento

1. Ordene o conjunto – $O(n \log n)$
2. Encontre o elemento no vetor – $O(\log n)$
3. Para cada repetição adjacente, troque o elemento com o último do vetor (utilize um índice para indicar o final)

Aplicações

■ Remover todas as duplicações de um vetor

1. Ordene o conjunto – $O(n \log n)$
2. Teste cada item i com $i+1$. Em caso de repetição, remova-o. – $O(n)$
3. Uma estrutura auxiliar pode facilitar as operações

Aplicações

■ Priorizar elementos

- ❑ Suponha que seja dado um conjunto de tarefas, cada uma com um deadline específico
- ❑ Ordenamos as tarefas de acordo com sua prioridade
- ❑ Ordenar é uma forma de implementar uma fila de prioridade

Aplicações

■ Encontrar o k-ésimo maior item de um conjunto.

1. Ordene o conjunto – $O(n \log n)$
2. Retorne o k-ésimo elemento vetor – $O(1)$

■ Encontrar a mediana de um conjuntos

1. Se o tamanho n for ímpar, basta encontrar o $(n/2)$ -ésimo maior elemento do vetor
2. Se n for par, basta retorna a média do $(n/2)$ -ésimo e do $(n/2+1)$ -ésimo maiores elementos do vetor

Aplicações

■ Contar frequências

Por exemplo: qual o item que mais ocorre em um vetor?

- ❑ Ordenamos o vetor – $O(n \log n)$
- ❑ Em seguida, basta percorrer o vetor e contar as ocorrências adjacentes – $O(n)$

Aplicações

■ Reconstruir a ordem original de um vetor

Como restaurar a ordem original de um vetor após uma série de permutação

1. Adicione um item extra no registro (indicando a posição original) – $O(n)$
2. Ordene de acordo com este item – $O(n \log n)$

Aplicações

- **Interseção e união de conjuntos**
- Se ambos conjuntos estiverem ordenados, podemos fazer em $O(n)$
- Basta percorrer os vetores pegando sempre o menor item de cada vetor e colocá-lo no conjunto

Aplicações

■ Busca eficiente

- Vetores ordenados podem ser pesquisados em tempo $O(\log n)$ através de busca binária

■ Para um dado número z , **encontrar dois elementos x e y em que $x + y = z$**

- Se os vetores estiverem ordenados, basta testar extremos (dois índices i e j), incrementando i e reduzindo j de acordo com os resultados
- Custo: $O(n)$

Complexidade assintótica

■ Algoritmos de ordenação por comparação

- ❑ **Quicksort** – $O(n^2)$; $O(n \log n)$ no caso médio
- ❑ Heapsort – $O(n \log n)$
- ❑ **MergeSort** – $O(n \log n)$
- ❑ Shellsort – $O(n^2)$ limite forte desconhecido :)
- ❑ InsertSort – $O(n^2)$
- ❑ SelectSort – $O(n^2)$
- ❑ Bubblesort – $O(n^2)$

Ordenação estável e não-estável

■ Ordenação estável

- A ordem dos elementos em que há empate é preservada

■ Ordenação não-estável

- Não há garantia de que a ordem os elementos de empate será preservada

Ordenação estável e não-estável

■ Algoritmos de ordenação estáveis

- ❑ **MergeSort** – $O(n \log n)$
- ❑ InsertSort – $O(n^2)$
- ❑ Bubblesort – $O(n^2)$

■ Algoritmos ordenação não-estáveis

- ❑ **Quicksort** – $O(n^2)$; $O(n \log n)$ no caso médio
- ❑ Heapsort – $O(n \log n)$
- ❑ Shellsort – $O(n^2)$
- ❑ SelectSort – $O(n^2)$

Maratona de Programação

Quais algoritmos utilizar?



MergeSort

- Método estável
- Custo $O(n \log n)$
- Disponível na STL: função **stable_sort**

MergeSort (STL)

```
#include <iostream>
#include <algorithm>
#include <vector>

using namespace std;

int main ()
{
    double mydoubles[] = {3.14, 1.41, 2.72, 4.67, 1.73};

    vector<double> myvector(mydoubles, mydoubles + 5);
    stable_sort ( myvector.begin(), myvector.end() );

    for ( vector<double>::iterator it = myvector.begin(); it != myvector.end(); ++it )
        cout << " " << *it;
    cout << endl;

    return 0;
}
```

QuickSort

- Algoritmo mais rápido para entradas maiores
- Método **não é estável**
- Método qsort da stdlib:

```
#include <stdlib.h>

int main() {
    ...
    qsort(vetor, n, sizeof(int), &comparador);
}

int comparador(const void *a, const void *b) {
    if (*(int*)a < *(int*)b) return -1;
    if (*(int*)a > *(int*)b) return 1;
    return 0;
}
```

QuickSort (STL)

```
#include <iostream>
#include <algorithm>
#include <vector>

using namespace std;

int main () {
    double mydoubles[] = {3.14, 1.41, 2.72, 4.67, 1.73};

    vector<double> myvector(mydoubles, mydoubles + 5);
    sort ( myvector.begin(), myvector.end() );

    for ( vector<double>::iterator it = myvector.begin(); it != myvector.end(); ++it)
        cout << " " << *it;
    cout << endl;

    return 0;
}
```

QuickSort (ordenação personalizada)

```
#include <iostream>
#include <algorithm>
#include <vector>

using namespace std;

bool compare_as_inteiro (double i, double j)
{
    return ( int(i) < int(j) );
}

int main ()
{
    double mydoubles[] = {3.14, 1.41, 2.72, 4.67, 1.73};
    vector<double> myvector(mydoubles, mydoubles + 5);
    sort (myvector.begin(), myvector.end(), compare_as_inteiro);

    for (vector<double>::iterator it=myvector.begin(); it != myvector.end(); ++it)
        cout << " " << *it;
    cout << endl;

    return 0;
}
```

QuickSort (ordenação multicritério)

```
typedef struct
{
    int nro_problemas;
    int penalty;
    char[50] equipe;
} Equipe;

bool equipe_compare(Equipe &a, Equipe &b)
{
    if (a.nro_problemas < b.nro_problemas)
        return true;
    else if (b.nro_problemas < a.nro_problemas)
        return false;
    else if (a.penalty < b.penalty)
        return true;
    return false;
}
```

Busca em vetores ordenados

■ Método **bsearch** da stdlib:

```
#include <stdlib.h>

int comparador(const void *a, const void *b) {
    if (*(int*)a < *(int*)b) return -1;
    if (*(int*)a > *(int*)b) return 1;
    return 0;
}

int main() {
    int vetor[] = { 1,3,15,50 };
    int n = 4;
    int k = 15;
    int *p = bsearch(&k, vetor, n, sizeof(int), &comparador);
    // (*p) será igual ao elemento
}
```

Busca em vetores ordenados

- Método **binary_search** da STL:

```
int main() {  
    int vetor[] = { 1, 3, 15, 50 };  
    vector<int> v(vetor, vetor + 4);  
  
    if (binary_search (v.begin(), v.end(), 3))  
        cout << "Encontrado!\n";  
    else  
        cout << "Nao encontrado.\n";  
}
```

- Retorna true ou false... Pouco útil em alguns casos...

Busca em vetores ordenados

- STL: `lower_bound`, `upper_bound` e `equal_range`
- Fazem busca binária e retornam iteradores. Uso varia em caso de haver mais de um elemento com a mesma chave:
- **`lower_bound`**: retorna o primeiro elemento
- **`equal_range`**: retorna um par contendo a primeira e a última ocorrência do elemento
- **`upper_bound`**: retorna o último elemento

Busca em vetores ordenados

```
#include <iostream>
#include <algorithm>
#include <vector>

using namespace std;

int main () {
    int myints[] = {10,20,30,30,20,10,10,20};
    vector<int> v(myints,myints+8);          // 10 20 30 30 20 10 10 20
    vector<int>::iterator low,up;

    sort (v.begin(), v.end());              // 10 10 10 20 20 20 30 30

    low=lower_bound (v.begin(), v.end(), 20);
    up= upper_bound (v.begin(), v.end(), 20);

    cout << "lower_bound at position " << int(low - v.begin()) << endl;
    cout << "upper_bound at position " << int(up - v.begin()) << endl;

    return 0;
}
```














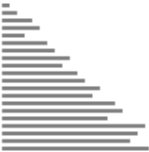
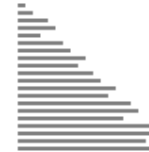

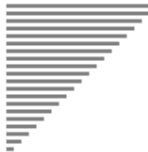

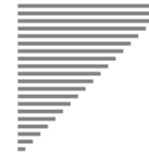

Sorting Algorithms Animations

The following animations illustrate how effectively data sets from different starting points can be sorted using different algorithms.

3.9K
SHARES



HOW TO USE: Press "Play all", or choose the ▶ button for the individual row/column to animate.

 Play All	 Insertion	 Selection	 Bubble	 Shell	 Merge	 Heap	 Quick	 Quick3
 Random								
 Nearly Sorted								
 Reversed								

<https://www.toptal.com/developers/sorting-algorithms/>

Exercício

- Polly tem uma academia de dança e vai participar de uma competição. Como todos seus alunos desejam ser seu parceiro, ela resolveu escolher quem tem as medidas ideais. Assim:
- – O primeiro critério será a altura: o mais próximo de 180 cm.
- – O segundo critério será o peso: o mais próximo de 75kg, sem ultrapassar.
- – O terceiro critério será a ordem crescente do sobrenome da pessoa.

Exercício

■ Entradas:

Nome (sempre primeiro e último) **Altura** (sempre cm) **Peso** (sempre kg)

■ Saídas:

- ❑ Lista com todos os classificados, ordenados pelos critérios de escolha

Exemplo de entrada:

George Bush	195	110
Harry Truman	180	75
Bill Clinton	180	75
John Kennedy	180	65
Ronald Reagan	165	110
Richard Nixon	170	70
Jimmy Carter	180	77

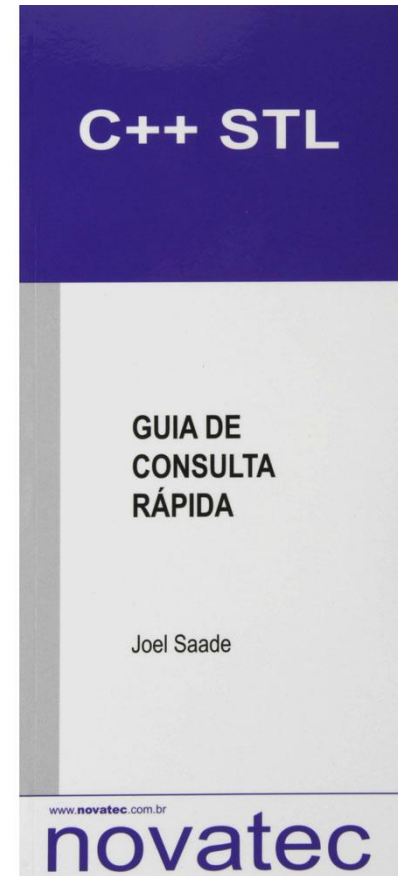
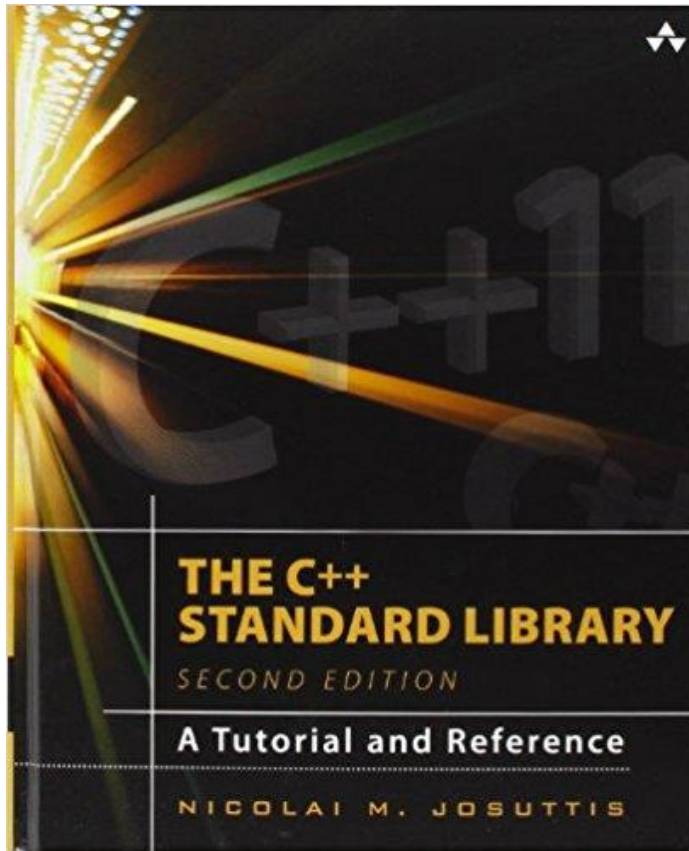
Exemplo de saída:

Clinton, Bill
Truman, Harry
Kennedy, John
Carter, Jimmy
Nixon, Richard
Bush, George
Reagan, Ronald

Agradecimentos

- Slides baseados nas aulas dos professores **Túlio A. M. Toffolo (UFOP)** e **Marco Antonio M. Carvalho (UFOP)**
-

Referências



Referências

