Universidade Federal de Roraima

Departamento de Ciência da Computação

Professor: Filipe Dwan.

Alunos: Felipe Derkian de Sousa Freitas Matrícula: 1201424418,  
 Philip Mahama Akpanyl Matrícula: 201514402

Trabalho de Ordenação e Busca.

Boa Vista, 10 de Fevereiro de 2017

# Bubble sort:

**O Bubble Sort**, ou ordenação por flutuação (literalmente "por bolha"), é um [algoritmo de ordenação](https://pt.wikipedia.org/wiki/Algoritmo_de_ordena%C3%A7%C3%A3o) dos mais simples. A ideia é percorrer o [vector](https://pt.wikipedia.org/wiki/Vector) 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](https://pt.wikipedia.org/wiki/Tanque_(reservat%C3%B3rio)) de água procuram seu próprio nível, e disso vem o nome do algoritmo.

No melhor caso, o algoritmo executa n operações relevantes, onde n representa o número de elementos do vector. No pior caso, são feitas n² operações. A [complexidade](https://pt.wikipedia.org/wiki/Complexidade) desse [algoritmo](https://pt.wikipedia.org/wiki/Algoritmo) é de [Ordem quadrática](https://pt.wikipedia.org/wiki/Ordem_quadr%C3%A1tica). Por isso, ele não é recomendado para programas que precisem de velocidade e operem com quantidade elevada de dados.

https://pt.wikipedia.org/wiki/Bubble\_sort

# Insertion sort:

**Insertion Sort**, ou ordenação por inserção, é o algoritmo de ordenação que, dado uma estrutura (array, lista) constrói uma matriz final com um elemento de cada vez, uma inserção por vez. Assim como algoritmos de ordenação quadrática, é bastante eficiente para problemas com pequenas entradas, sendo o mais eficiente entre os algoritmos desta ordem de classificação.

Podemos fazer uma comparação do Insertion Sort com o modo de como algumas pessoas organizam um baralho num jogo de cartas. Imagine que você está jogando cartas. Você está com as cartas na mão e elas estão ordenadas. Você recebe uma nova carta e deve colocá-la na posição correta da sua mão de cartas, de forma que as cartas obedeçam a ordenação.

A cada nova carta adicionada a sua mão de cartas, a nova carta pode ser menor que algumas das cartas que você já tem na mão ou maior, e assim, você começa a comparar a nova carta com todas as cartas na sua mão até encontrar sua posição correta. Você insere a nova carta na posição correta, e, novamente, sua mão é composta de cartas totalmente ordenadas. Então, você recebe outra carta e repete o mesmo procedimento. Então outra carta, e outra, e assim por diante, até você não receber mais cartas.

Esta é a ideia por trás da ordenação por inserção. Percorra as posições do array, começando com o índice 1 (um). Cada nova posição é como a nova carta que você recebeu, e você precisa inseri-la no lugar correto no subarray ordenado à esquerda daquela posição.

## Características:

Apesar de ser menos eficiente que algoritmos como Merge Sort e Quick Sort (de ordem O(nlog(n))), o Insertion Sort possui algumas características consideráveis:

É de simples implementação, leitura e manutenção;

In-place: Apenas requer uma quantidade constante de O(1) espaço de memória adicional;

Estável: Não muda a ordem relativa de elementos com valores iguais;

Útil para pequenas entradas;

Muitas trocas, e menos comparações;

Melhor caso: O(n), quando a matriz está ordenado;

Médio caso: O(n²/4), quando a matriz tem valores aleatórios sem ordem de classificação (crescente ou decrescente);

Pior caso: O(n²), quando a matriz está em ordem inversa, daquela que deseja ordenar.

## Vantagens e Desvantagens:

É o método a ser utilizado quando o arquivo está "quase" ordenado

É um bom método quando se desejar adicionar poucos elementos em um arquivo já ordenado, pois seu custo é linear.

O algoritmo de ordenação por inserção é estável.

https://pt.wikipedia.org/wiki/Insertion\_sort

# Selection sort:

A ordenação por seleção (do inglês, selection sort) é um [algoritmo de ordenação](https://pt.wikipedia.org/wiki/Algoritmo_de_ordena%C3%A7%C3%A3o" \o "Algoritmo de ordenação) baseado em se passar sempre o menor valor do vetor para a primeira posição (ou o maior dependendo da ordem requerida), depois o de segundo menor valor para a segunda posição, e assim é feito sucessivamente com os n − 1 elementos restantes, até os últimos dois elementos.

O Selection Sort compara a cada interação um elemento com os outros, visando encontrar o menor. Dessa forma, podemos entender que não existe um melhor caso mesmo que o vetor esteja ordenado ou em ordem inversa serão executados os dois laços do algoritmo, o externo e o interno. A complexidade deste algoritmo será sempre O ( n 2 ) enquanto que, por exemplo, os algoritmos [Heapsort](https://pt.wikipedia.org/wiki/Heapsort) e [Mergesort](https://pt.wikipedia.org/wiki/Mergesort) possuem complexidades.

## Vantagens:

Ele é um algoritmo simples de ser implementado em comparação aos demais.

Não necessita de uma vetor auxiliar (in-place).

Por não usar um vetor auxiliar para realizar a ordenação, ele ocupa menos memória.

Ele é uns dos mais velozes na ordenação de vetores de tamanhos pequenos.

Desvantagens:

Ele é um dos mais lentos para vetores de tamanhos grandes.

Ele não é estável.

Ele faz sempre n 2 comparações, independente do vetor está ordenado ou não.

# <https://pt.wikipedia.org/wiki/Selection_sort>

# Quick sort:

O [algoritmo](https://pt.wikipedia.org/wiki/Algoritmo) Quicksort é um método de ordenação muito rápido e eficiente, inventado por [C.A.R. Hoare](https://pt.wikipedia.org/wiki/C.A.R._Hoare) em 1960[[1]](https://pt.wikipedia.org/wiki/Quicksort#cite_note-1), quando visitou a [Universidade de Moscovo](https://pt.wikipedia.org/wiki/Universidade_de_Moscovo) como estudante. Naquela época, Hoare trabalhou em um projeto de [tradução de máquina](https://pt.wikipedia.org/w/index.php?title=Tradu%C3%A7%C3%A3o_de_m%C3%A1quina&action=edit&redlink=1) para o [National Physical Laboratory](https://pt.wikipedia.org/w/index.php?title=National_Physical_Laboratory,_UK&action=edit&redlink=1). Ele criou o Quicksort ao tentar traduzir um dicionário de inglês para russo, ordenando as palavras, tendo como objetivo reduzir o problema original em subproblemas que possam ser resolvidos mais fácil e rápido. Foi publicado em 1962 após uma série de refinamentos.

Quicksort é um algoritmo de [ordenação por comparação](https://pt.wikipedia.org/wiki/Ordena%C3%A7%C3%A3o_por_compara%C3%A7%C3%A3o) [não-estável](https://pt.wikipedia.org/wiki/Ordena%C3%A7%C3%A3o_est%C3%A1vel).

## O algoritmo:

O Quicksort adota a estratégia de [divisão e conquista](https://pt.wikipedia.org/wiki/Divis%C3%A3o_e_conquista). A estratégia consiste em rearranjar as chaves de modo que as chaves "menores" precedam as chaves "maiores". Em seguida o Quicksort ordena as duas sublistas de chaves menores e maiores recursivamente até que a lista completa se encontre ordenada. Os passos são:

Escolha um elemento da lista, denominado pivô;

Rearranje a lista de forma que todos os elementos anteriores ao pivô sejam menores que ele, e todos os elementos posteriores ao pivô sejam maiores que ele. Ao fim do processo o pivô estará em sua posição final e haverá duas sublistas não ordenadas. Essa operação é denominada partição;

[Recursivamente](https://pt.wikipedia.org/wiki/Recursividade_(ci%C3%AAncia_da_computa%C3%A7%C3%A3o)) ordene a sublista dos elementos menores e a sublista dos elementos maiores;

A base da recursão são as listas de tamanho zero ou um, que estão sempre ordenadas. O processo é finito, pois a cada iteração pelo menos um elemento é posto em sua posição final e não será mais manipulado na iteração seguinte.

## Complexidade:

Comportamento no pior caso

O pior caso de particionamento ocorre quando o elemento pivô divide a lista de forma desbalanceada, ou seja, divide a lista em duas sublistas: uma com tamanho 0 e outra com tamanho n - 1 (no qual n se refere ao tamanho da lista original). Isso pode ocorrer quando o elemento pivô é o maior ou menor elemento da lista, ou seja, quando a lista já está ordenada, ou inversamente ordenada.

Se isso acontece em todas as chamadas do método de particionamento, então cada etapa recursiva chamará listas de tamanho igual à lista anterior - 1. Teremos assim, a seguinte relação de recorrência:

T(n) = T(n - 1) + T(0) + θ(n)

= T(n - 1) + θ(n).

Se somarmos os custos em cada nível de recursão, teremos uma série aritmética que tem valor θ(n²), assim, o algoritmo terá tempo de execução igual à θ(n²).

Comportamento no melhor caso

O melhor caso de particionamento acontece quando ele produz duas listas de tamanho não maior que n/2, uma vez que uma lista terá tamanho [n/2] e outra tamanho [n/2] - 1. Nesse caso, o quicksort é executado com maior rapidez. A relação de recorrência é a seguinte:

T(n) ≤ 2T(n/2) + θ(n)

que, a partir do teorema mestre, terá solução T(n) = O(n log n).

Complexidade de espaço: θ(log 2 n) no melhor caso e no caso médio e θ(log 2 n) no pior caso. R. Sedgewick desenvolveu uma versão do Quicksort com partição recursão de cauda que tem complexidade θ(n2) no pior caso.

https://pt.wikipedia.org/wiki/Selection\_sort

# Heapsort:

O [algoritmo](https://pt.wikipedia.org/wiki/Algoritmo) Heap Sort é um [algoritmo de ordenação](https://pt.wikipedia.org/wiki/Algoritmo_de_ordena%C3%A7%C3%A3o) generalista, e faz parte da família de algoritmos de [ordenação por seleção](https://pt.wikipedia.org/wiki/Ordena%C3%A7%C3%A3o_por_sele%C3%A7%C3%A3o). Foi desenvolvido em 1964 por Robert W. Floyd e J.W.J Williams.

Tem um desempenho em tempo de execução muito bom em conjuntos ordenados aleatoriamente, tem um uso de memória bem comportado e o seu desempenho em [pior cenário](https://pt.wikipedia.org/wiki/Complexidade_Pior_caso" \o "Complexidade Pior caso) é praticamente igual ao desempenho em [cenário médio](https://pt.wikipedia.org/wiki/Complexidade_Caso_m%C3%A9dio" \o "Complexidade Caso médio). Alguns algoritmos de ordenação rápidos têm desempenhos espectacularmente ruins no pior cenário, quer em tempo de execução, quer no uso da memória. O Heapsort trabalha no lugar e o tempo de execução em pior cenário para ordenar n elementos é de [O](https://pt.wikipedia.org/wiki/Nota%C3%A7%C3%A3o_O_mai%C3%BAsculo" \o "Notação O maiúsculo) (n lg n). Lê-se logaritmo (ou log) de "n" na base 2. Para valores de n, razoavelmente grandes, o termo log n é quase constante, de modo que o tempo de ordenação é quase linear com o número de itens a ordenar.

## Complexidade:

Comparações no pior caso: 2n log2n + O(n) é o mesmo que 2n lgn + O(n)

Trocas no pior caso: n log2n + O(n) é o mesmo que n lgn + O(n)

Melhor e pior caso: O(n log2n) é o mesmo que O(n lgn)

## Caracteristica:

O Heapsort não é um algoritmo de [ordenação estável](https://pt.wikipedia.org/wiki/Ordena%C3%A7%C3%A3o_est%C3%A1vel" \o "Ordenação estável). Porém, é possível adaptar a estrutura a ser ordenada de forma a tornar a ordenação estável. Cada elemento da estrutura adaptada deve ficar no formato de um par (elemento original, índice original). Assim, caso dois elementos sejam iguais, o desempate ocorrerá pelo índice na estrutura original.

## Funcionamento:

O heapsort utiliza uma estrutura de dados chamada [heap](https://pt.wikipedia.org/wiki/Heap" \o "Heap), para ordenar os elementos à medida que os insere na estrutura. Assim, ao final das inserções, os elementos podem ser sucessivamente removidos da raiz da heap, na ordem desejada, lembrando-se sempre de manter a propriedade de max-heap.

A heap pode ser representada como uma árvore (uma árvore binária com propriedades especiais[[2]](https://pt.wikipedia.org/wiki/Heapsort" \l "cite_note-2)) ou como um vetor. Para uma ordenação decrescente, deve ser construída uma heap mínima (o menor elemento fica na raiz). Para uma ordenação crescente, deve ser construído uma heap máxima (o maior elemento fica na raiz).

https://pt.wikipedia.org/wiki/Heapsort

# Merge sort:

O merge sort, ou [ordenação](https://pt.wikipedia.org/wiki/Ordena%C3%A7%C3%A3o_(computa%C3%A7%C3%A3o)) por mistura, é um exemplo de [algoritmo de ordenação por comparação](https://pt.wikipedia.org/wiki/Ordena%C3%A7%C3%A3o_por_compara%C3%A7%C3%A3o) do tipo [dividir-para-conquistar](https://pt.wikipedia.org/wiki/Divis%C3%A3o_e_Conquista).

Sua ideia básica consiste em Dividir (o problema em vários subproblemas e resolver esses subproblemas através da recursividade) e Conquistar (após todos os subproblemas terem sido resolvidos ocorre a conquista que é a união das resoluções dos subproblemas). Como o algoritmo Merge Sort usa a recursividade, há um alto consumo de memória e tempo de execução, tornando esta técnica não muito eficiente em alguns problemas.

## Características

Os três passos úteis dos algoritmos de [divisão e conquista](https://pt.wikipedia.org/wiki/Divis%C3%A3o_e_conquista), ou divide and conquer, que se aplicam ao merge sort são:

Dividir: Calcula o ponto médio do sub-arranjo, o que demora um tempo constante {\displaystyle \Theta (1)}O(1);

Conquistar: [Recursivamente](https://pt.wikipedia.org/wiki/Recursividade_(ci%C3%AAncia_da_computa%C3%A7%C3%A3o)) resolve dois sub-problemas, cada um de tamanho n/2, o que contribui com 2T(n/2) para o tempo de execução;

Combinar: Unir os sub-arranjos em um único conjunto ordenado, que leva o tempo {\displaystyle \Theta (n)}O(n).

## Complexidade

Complexidade de tempo: O (n log n).{\displaystyle \Theta (nlogn)}O

Complexidade de espaço: {\displaystyle \Theta (n)}O (n).

https://pt.wikipedia.org/wiki/Merge\_sort

# Radix sort:

O **Radix sort** é um [algoritmo de ordenação](https://pt.wikipedia.org/wiki/Algoritmo_de_ordena%C3%A7%C3%A3o) rápido e [estável](https://pt.wikipedia.org/wiki/Ordena%C3%A7%C3%A3o_est%C3%A1vel) que pode ser usado para ordenar itens que estão identificados por [chaves](https://pt.wikipedia.org/wiki/Chave_(computa%C3%A7%C3%A3o)) únicas. Cada chave é uma [cadeia de caracteres](https://pt.wikipedia.org/wiki/Cadeia_de_caracteres) ou número, e o radix sort ordena estas chaves numa qualquer ordem relacionada com a lexicografia.

Na ciência da computação, radix sort é um algoritmo de ordenação que ordena inteiros processando dígitos individuais. Como os inteiros podem representar strings compostas de caracteres (como nomes ou datas) e pontos flutuantes especialmente formatados, radix sort não é limitado somente a inteiros.

Computadores, na sua maioria, representam internamente todos os tipo de dados como números binários, por isso processar os dígitos na forma de inteiros em grupos representados por dígitos binários se torna mais conveniente. Existem duas classificações do radix sort, que são:

- Least significant digit (LSD – Dígito menos significativo) radix sort; - Most significant digit (MSD – Dígito mais significativo) radix sort.

O radix sort LSD começa do dígito menos significativo até o mais significativo, ordenando tipicamente da seguinte forma: chaves curtas vem antes de chaves longas, e chaves de mesmo tamanho são ordenadas lexicograficamente. Isso coincide com a ordem normal de representação dos inteiros, como a seqüência "1, 2, 3, 4, 5, 6, 7, 8, 9, 10". Os valores processados pelo algoritmo de ordenação são frequentemente chamados de “chaves”, que podem existir por si próprias ou associadas a outros dados. As chaves podem ser strings de caracteres ou números.

Já o radix sort MSD trabalha no sentido contrário, usando sempre a ordem lexicográfica, que é adequada para ordenação de strings, como palavras, ou representações de inteiros com tamanho fixo. A seqüência "b, c, d, e, f, g, h, i, j, ba" será ordenada lexicograficamente como "b, ba, c, d, e, f, g, h, i, j". Se essa ordenação for usada para ordenar representações de inteiros com tamanho variável, então a representação dos números inteiros de 1 a 10 terá a saída "1, 10, 2, 3, 4, 5, 6, 7, 8, 9".

O radix sort LSD opera na notação Big O, em O(nk), onde "n" é o número de chaves, e "k" é o comprimento médio da chave. É possível garantir esse desempenho para chaves com comprimento variável agrupando todas as chaves que tem o mesmo comprimento e ordenando separadamente cada grupo de chaves, do mais curto para o mais comprido, de modo a evitar o processamento de uma lista inteira de chaves em cada passo da ordenação.

Em muitas aplicações em que é necessário velocidade, o radix sort melhora as ordenações por comparação, como heapsort e o mergesort, que necessitam de Ω(n · log n) comparações, onde "n" é o número de itens a serem ordenados. Em compensação, algoritmos de ordenação baseados em comparações oferecem flexibilidade por serem aptos a ordenar de outras formas que não a lexicográfica. No entanto, essa habilidade é de pouca importância em várias aplicações práticas.

O algoritmo de ordenação radix foi originalmente usado para ordenar cartões perfurados. Um algoritmo computacional para o radix sort foi inventado em 1954 no MIT por Harold H. Seward.

## Características

Complexidade de Tempo: Θ(nk).

Complexidade de espaço: Θ(n + s).

– n = número de elementos.

– k = tamanho string.

– s = tamanho do alfabeto.

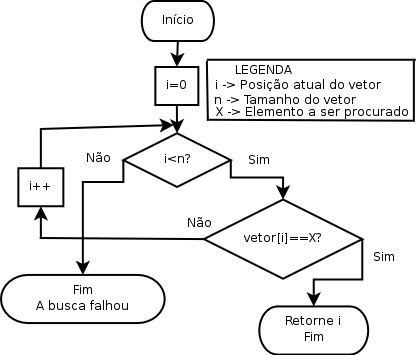
<https://pt.wikipedia.org/wiki/Radix_sort>

# Busca linear:

Na área de [informática](https://pt.wikipedia.org/wiki/Inform%C3%A1tica), ou [Ciência da Computação](https://pt.wikipedia.org/wiki/Ci%C3%AAncia_da_Computa%C3%A7%C3%A3o), costuma-se usar o termo **busca linear** (ou busca sequencial) para expressar um tipo de pesquisa em [vetores](https://pt.wikipedia.org/wiki/Vetor) ou [listas](https://pt.wikipedia.org/wiki/Lista) de modo sequencial, i. e., elemento por elemento, de modo que a função do tempo em relação ao número de elementos é linear, ou seja, cresce proporcionalmente. Num vetor ordenado, essa não é a pesquisa mais eficiente, a [pesquisa (ou busca) binária](https://pt.wikipedia.org/wiki/Pesquisa_bin%C3%A1ria), por exemplo, é um tipo de pesquisa com o gráfico de tempo logarítmo.

## Análise de Complexidade:

No melhor caso, o elemento a ser buscado é encontrado logo na primeira tentativa da busca. No pior caso, o elemento a ser buscado encontra-se na última posição e são feitas N comparações, sendo N o número total de elementos. No caso médio, o elemento é encontrado após (N+1)/2 comparações. O algoritmo de busca linear é um algoritmo O(n).

[](https://pt.wikipedia.org/wiki/Ficheiro:Busca_sequencial.png)

https://pt.wikipedia.org/wiki/Busca\_linear

# Pesquisa binária:

A pesquisa ou busca binária (em [inglês](https://pt.wikipedia.org/wiki/L%C3%ADngua_inglesa) binary search algorithm ou binary chop) é um [algoritmo de busca](https://pt.wikipedia.org/wiki/Algoritmo_de_busca) em [vetores](https://pt.wikipedia.org/wiki/Vetor) que segue o paradigma de [divisão e conquista](https://pt.wikipedia.org/wiki/Divis%C3%A3o_e_conquista). Ela parte do pressuposto de que o vetor está ordenado e realiza sucessivas divisões do espaço de busca comparando o elemento buscado (chave) com o elemento no meio do vetor. Se o elemento do meio do vetor for a chave, a busca termina com sucesso. Caso contrário, se o elemento do meio vier antes do elemento buscado, então a busca continua na metade posterior do vetor. E finalmente, se o elemento do meio vier depois da chave, a busca continua na metade anterior do vetor.

## Análise de Complexidade:

A [complexidade](https://pt.wikipedia.org/wiki/Complexidade_computacional) desse [algoritmo](https://pt.wikipedia.org/wiki/Algoritmo) é da ordem de Θ(log2 n), em que n é o tamanho do vetor de busca. Apresenta-se mais eficiente que a [Busca linear](https://pt.wikipedia.org/wiki/Busca_linear) cuja ordem é O(n).

https://pt.wikipedia.org/wiki/Pesquisa\_bin%C3%A1ria