**Métodos de Ordenação: Insertion sort, Selection sort, Bubble sort e Quick sort**

# **João Vitor de OLIVEIRA Camara(1), Lúcio RODRIGUES Silva Filho(2), Gabriel da SILVA(3)**

(1,2) Estudantes de Engenharia da Computação. Instituto Federal Minas Gerais (IFMG) *campus* Bambuí. Rod. Bambuí/Medeiros km 5. CEP: 38900-000. Bambuí-MG. (3) Professor Orientador – IFMG.

{joaovitoroliveiracamara, lucioraiser}@gmail.com

***Abstract****:This present article has focused on analyze the principal characteristics of the four (4) sorting methods presented, Insertion Sort, Selection Sort, Bubble Sort, QuickSort, approaching his contents of input, output, conditional structures and your main applications, looking for better understanding. And starting of this point, this concepts will be better informed.*

***Resumo:*** *O presente artigo teve como foco analisar as principais características dos quatro(4) métodos de ordenação apresentados, Insertion Sort, Selection Sort, Bubble Sort, QuickSort, abordando os conteúdos de entrada, saída, estruturas condicionais e suas principais aplicações, buscando assim melhor entendimento. E a partir deste ponto, estes conceitos serão adiante melhor esclarecidos.*

**Introdução**

A ordenação consiste em reagrupar elementos e posicioná-los em uma ordem específica, com o objetivo de facilitar a recuperação de dados de uma lista. Com o uso do algoritmo de ordenação certo é possível realizar essa tarefa em pouco tempo. Para esse artigo foram selecionados quatro (4) algoritmos de ordenação a serem estudados, são eles: *Insertion sort, Selection sort, Bubble sort e Quick sort*. Os mesmos foram usados para ordenar cinco(5) conjuntos pequenos e cinco (5) conjuntos grandes de números agrupados em ordens distintas.

*Insertion sort*:. O insertion sort também é recomendado para conjuntos ou listas pequenas, neste algoritmo os elementos são considerados um a um e inseridos em sua posição em meio os elementos já tratados de acordo com o critério de ordenação. Este algoritmo funciona da seguinte forma. Vamos tratar que o critério da ordenação seja do menor para o maior (crescente). É feita a comparação dos elementos nas posições um (1) e dois (2). A partir da posição dois (2), é selecionado o elemento a sua direita, este é guardado em uma posição auxiliar e comparado com os elementos a sua esquerda. Caso todos sejam maiores que o elemento na posição auxiliar, este toma a posição um (1) do vetor. Continuando, o elemento selecionado agora será o da posição três (3), assim o elemento a sua direita é guardado na posição auxiliar e comparado com os elementos a esquerda, todos elementos a esquerda maiores que se encontra na posição auxiliar andam uma posição à direita até que o elemento da posição auxiliar seja maior que o seu correspondente à sua esquerda. É repetido esse processo até a ordenação completa do vetor.

*Selection Sort*: Este algoritmo é recomendado para conjuntos pequenos, consiste basicamente em alocar o menor (ou maior, depende da ordem estabelecida) elemento do vetor mais a esquerda possível a cada interação, sendo assim, seleciona o elemento da posição um (1), este é comparado com os demais a direita, caso encontre algum elemento menor que o selecionado, o elemento selecionado é guardado em uma posição auxiliar para que o elemento menor possa assumir sua posição um (1) e o elemento guardado na posição auxiliar é alocado na antiga posição do menor elemento, fazendo assim a troca entre eles. Caso nenhum elemento nas demais posições seja menor que o selecionado, então será selecionado o próximo elemento a direita para comparação, sendo assim, o elemento na posição dois (2). Após a comparação é selecionado o elemento na posição três (3). Será repetido o processo até que o vetor esteja completamente ordenado.

*Bubble Sort*: É um algoritmo mais simples mas também o menos eficiente, isso porque ele compara o elemento selecionado sempre com o elemento a sua direita. Na primeira interação ele seleciona o elemento na posição um (1) e compara com o elemento na posição dois (2), caso este for menor será feita a troca. Na segunda interação é selecionado o elemento na posição dois (2) e ele será comparado com o elemento na posição três (3), ou seja, o primeiro a direita. Da mesma forma caso este for menor será feita a troca, mas se algum elemento não estiver na posição correta após o algoritmo percorrer todo o vetor ele só será alocado em outra posição quando chegar sua vez de ser comparado novamente. Isso porque cada elemento sofre interação uma vez por vez que for percorrido o vetor, então se não primeira o vetor não estiver ordenado, será repetida todas as interações das posições novamente até que o vetor esteja ordenado corretamente. Fazendo com que o algoritmo seja ineficiente para conjuntos grandes.

*Quick Sort*: O quicksort tem como característica principal adotar o método de divisão e conquista,uma técnica de projeto de algoritmos utilizada pela primeira vez por Anatolii Karatsuba em 1960 no algoritmo de Karatsuba.Essa técnica tem como objetivo dividir um problema maior recursivamente em problemas menores até que o problema possa ser resolvido diretamente, isso é feita utilizando os diversos fragmentos dos problemas menores (característica implícita nas técnicas de algoritmos recursivos).O algoritmo quicksort tem como características:a escolha de um elemento aleatório, inicialmente, da lista que é denominado *pivô*.O segundo passo é reordenar a lista garantindo que todos os elementos antecessores do *pivô* sejam menores que ele e todos os elementos posterior a ele sejam maiores que o mesmo,ao fim desse processo o *pivô* estará em sua posição correta e final e as duas sublistas (antecessora e posterior) geradas a partir da ordenação do *pivô* estarão desordenadas.Recursivamente as sublistas serão ordenadas.

É importante salientar o caso base do *quicksort*,que são listas de tamanho zero ou um que por si só estarão sempre ordenadas.

**Desenvolvimento**

Para a realização dos testes, foi implementado um código em C++ caracterizado por dois menus. Para realizar a construção deste código foi utilizado o software Code: Blocks. O computador onde foram realizados os testes é um Dell Inspiron N4050,com um processador Intel(R) Core(TM) i3-2370M CPU @ 2.40GHz ,memória RAM 4GB e os testes foram realizados no Sistema Operacional Linux na distribuição Fedora 25.

É importante deixar definido os nomes que foram atribuídos aos vetores para deixar claro o entendimento do resto do artigo:

Vetor 1:Um vetor de 100 números inteiros distintos aleatórios;

Vetor 2:Um vetor de 100 números inteiros aleatórios com algumas repetições;

Vetor 3:Um vetor de 100 números inteiros aleatórios distintos ordenados;

Vetor 4:Um vetor de 100 números inteiros aleatórios distintos quase ordenados;

Vetor 5:Um vetor de 100 números inteiros aleatórios distintos invertidos;

Vetor 6:Um vetor de 5000 números inteiros distintos aleatórios;

Vetor 7:Um vetor de 5000 números inteiros aleatórios com algumas repetições;

Vetor 8:Um vetor de 5000 números inteiros aleatórios distintos ordenados;

Vetor 9:Um vetor de 5000 números inteiros aleatórios distintos quase ordenados;

Vetor 10:Um vetor de 5000 números inteiros aleatórios distintos invertidos;

Vetor 11:Um vetor de 30000 números inteiros aleatórios e distintos.

Foram usados métodos simples e um método eficiente para ordenar os conjuntos propostos. Os métodos simples (*Selection Sort, Bubble Sort e Insertion Sort*) são apropriados para pequenos vetores, sendo assim possuem complexidade C(n) = O (n²), quer dizer que requer O(n²) comparações.

Onde:

-C(n) = Número de Comparações entre chaves;

- n = Número de Registros.

Em métodos eficientes, são feitas mais comparações do que nos métodos simples. São feitos especialmente para trabalhar com conjuntos maiores de dados e sua complexidade se dá por C(n) = O(n log n). O método eficiente usado para os testes neste artigo foi o *Quick Sort.*

Logo abaixo estão os métodos de ordenação em código.

***Selection Sort***

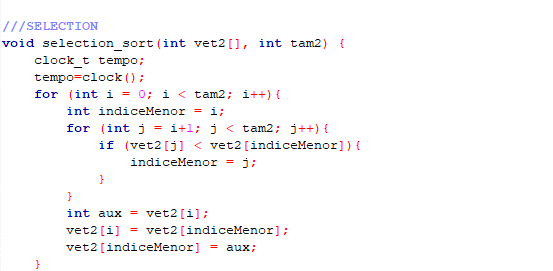
****

Imagem 1.

Como mostrado, o Selection possui dois laços. O laço interno é encarregado de percorrer todo o vetor e o laço externo é o controlador de índice. Na primeira interação o índice começa em 0, assim a cada interação é somado um (1) até o último índice do vetor, e o laço interno percorre o vetor começando do índice externo mais um (+1). Em termos de complexidade de tempo utilizando a notação *big* O se denotam o pior caso sendo como O(n²),caso médio O(n²) e melhor caso O(n²).

## 

## ***Bubble Sort***

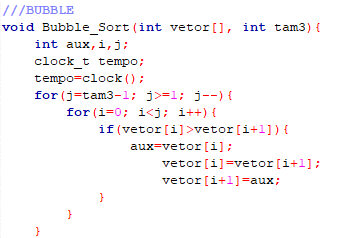


Imagem 1.1

## Já no *Bubble Sort* são comparados os elementos dois a dois, o vetor é percorrido até que nenhum elemento precise ser trocado na interação anterior. No melhor caso serão executadas (n) operações relevantes, ou seja, um vetor quase ordenado, sendo (n) o número a quantidade de elementos. O pior caso se dá (n²) operações, onde o vetor estará invertido.

## 

## ***Insertion Sort***

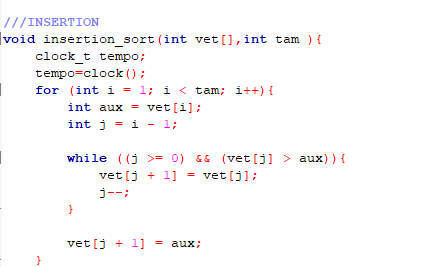


Imagem 1.1.1

O *Insertion Sort* ou Ordenação por Inserção é o mais eficiente dos métodos simples. Ele inicia a interação com o índice um (1) do vetor, a cada vez que ele passar no laço “for” a variável aux armazena o valor contido na posição i atual do vetor, criando uma lacuna no índice onde o elemento estava armazenado. Se existir outros elementos maiores que o elemento em aux nos índices anteriores, estes podem ser armazenados um índice à direita a assim o elemento em aux será armazenado a esquerda, o processo será repetido com todos os índices do vetor até que ele esteja ordenado. Em termos de complexidade de tempo utilizando a notação *big* O se denotam o pior caso sendo como O(n²), caso médio O(n²) e melhor caso O(n).

***Quick Sort***

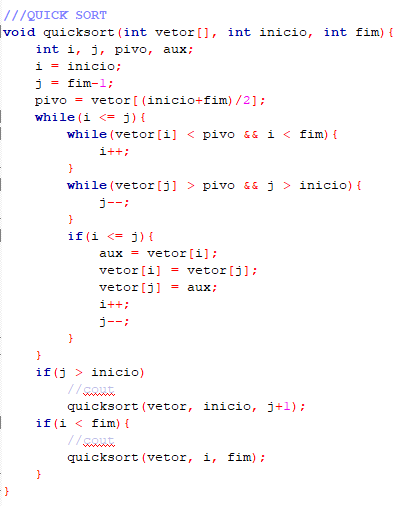
****

Imagem 1.1.1.1

A ideia do *Quick Sort* é dividir e conquistar, basicamente um elemento é escolhido como pivô. Os dados serão rearranjados e os elementos menores que o pivô serão armazenados a esquerda e os maiores a direita. Recursivamente essas duas partições são ordenadas. Em termos de complexidade de tempo utilizando a notação *big* O se denotam o pior caso sendo como O(n²),caso médio O (n log n),melhor caso O(n log n).

Para cada método de ordenação foi utilizada uma função denominada *clock(),*pertencente a biblioteca *time.h,* que iniciada nos locais onde começa a ordenação e finalizada no fim da ordenação resulta em um tempo de *clocks per second.*Feito isso,se utilizada do recurso da divisão de CLOCKS\_PER\_SEC para se retornar apenas o tempo em segundos.

A execução do código após compilado se apresenta da seguinte forma:

No primeiro menu foram distribuídos os 11 conjuntos trabalhados, enumerados de 1 até 11. Dos quais, de 1 até 5 os conjuntos pequenos, de 6 até 11 os conjuntos grandes. Os conjuntos estão apresentados na imagem abaixo:

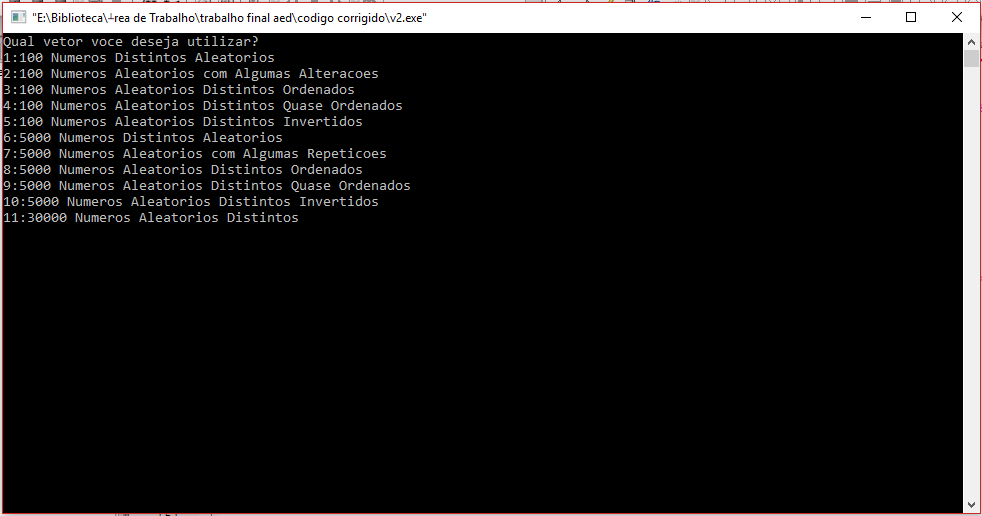
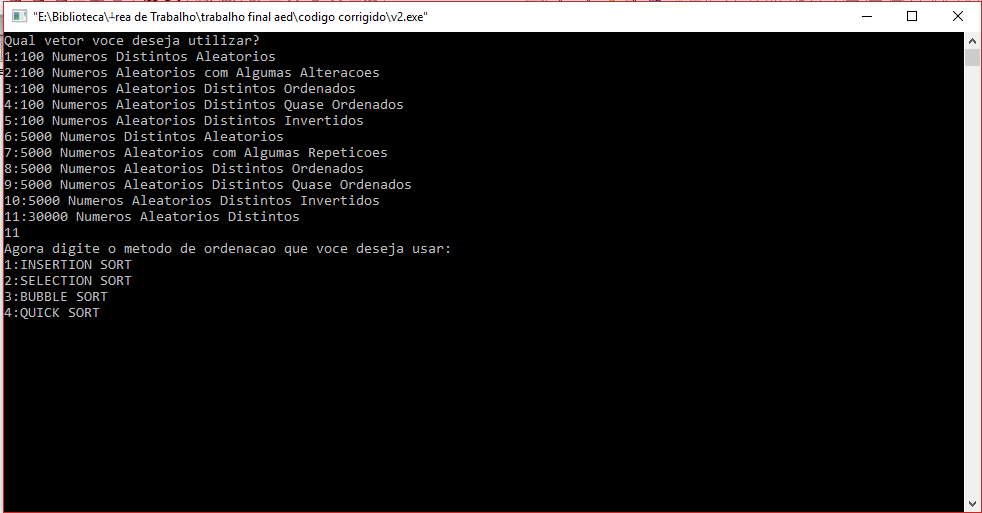


Imagem 2

Neste menu, é selecionado o tipo de conjunto que será usado no teste. Tendo selecionado o conjunto, logo após, será mostrado o segundo menu com os métodos de ordenação. Enumerados de 1 até 4.

Imagem 2.1

Após a seleção do vetor que o usuário deseja utilizar na ordenação é aberto um segundo menu que serve de apoio para o mesmo escolher qual método de ordenação ele deseja testar.Esse método retorna um valor em segundos do tempo que foi gasto para ordenar o vetor.

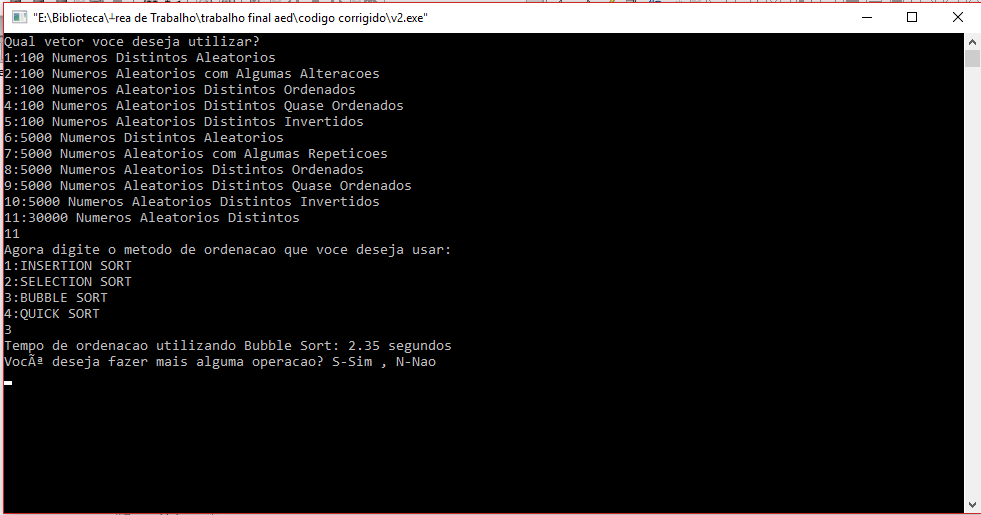


Imagem 2.1.1

Ao fim desse processo de mostra do tempo gasto na ordenação é exibida uma mensagem que pergunta ao usuário se ele deseja fazer uma nova ordenação,caso ele deseje,o programa pede novamente para ele selecionar um vetor e o método de ordenação em seguida e esse ciclo se repete.

**Discussão sobre os resultados observados:**

Foram executados testes usando todos os métodos de ordenação para cada grupo de elementos apresentados. Os resultados obtidos em cada método de ordenação estão nos gráficos abaixo.

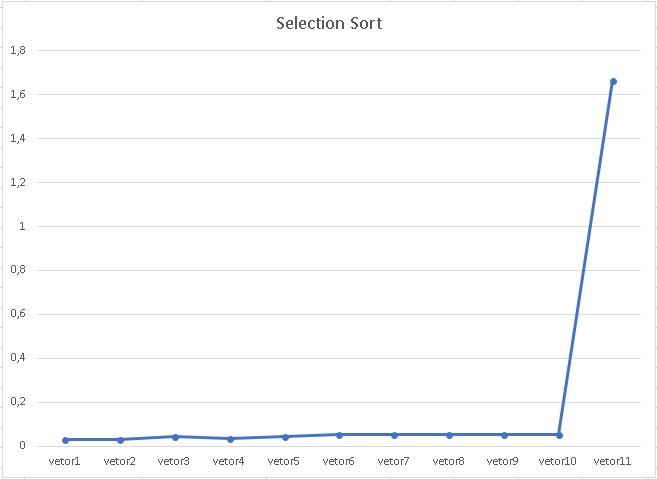


Imagem 3

O *Selection Sort*  foi o terceiro mais eficaz de acordo com os testes realizados, pois, foi o segundo mais rápido a ordenar os vetores 2,8,10. e o terceiro mais rápido a ordenar os vetores 1,3,4,5,6,7,9,11.

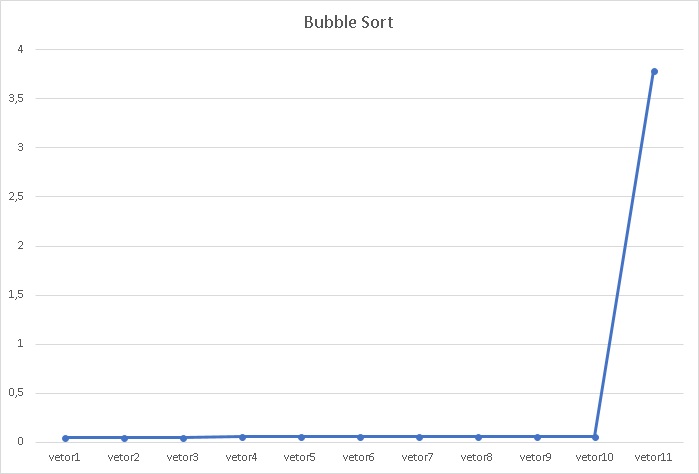


Imagem 3.1

O *Bubble Sort* de acordo com os testes realizados é o mais lento de todos os métodos de ordenação, tanto para conjuntos pequenos quanto para os grandes. Foi apenas o terceiro método mais rápido no vetor 8, composto por 5.000 números aleatórios distintos ordenados e no vetor 10, composto por 5.000 números aleatórios distintos invertidos.

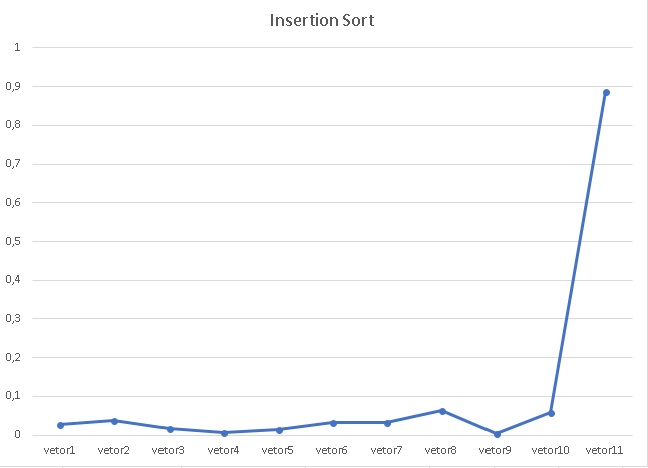


Imagem 3.1.1

O *Insertion Sort* comparado a todos os outros métodos foi o segundo mais eficaz na maioria dos testes. Foi o mais rápido a concluir a ordenação do vetor 4, composto por 100 números distintos quase ordenados. Isso porque o método por inserção foi exclusivamente feito para conjuntos pequenos de dados quase ordenados, o vetor 4 se encaixa no seu melhor caso, de complexidade O (n). Foi também o segundo mais eficaz na maioria dos testes, comparando-o apenas com os métodos simples de ordenação. Dos 11 vetores testados só não o mais rápido entre os métodos simples nos vetores 2, composto por 100 números aleatórios com algumas repetições. Vetor 5.000 números aleatórios distintos ordenados. E no vetor 5.000 números aleatórios distintos invertidos.

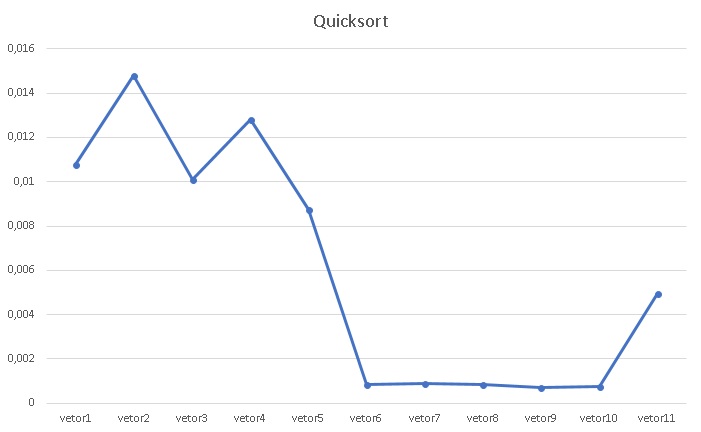


Imagem 3.1.1.1

O *Quick Sort* pela forma eficiente de comparação e ordenação dos elementos comparado com os demais, foi o mais eficaz em quase todos os vetores testados. Exceto no vetor 4, composto de 100 números distintos quase ordenados.

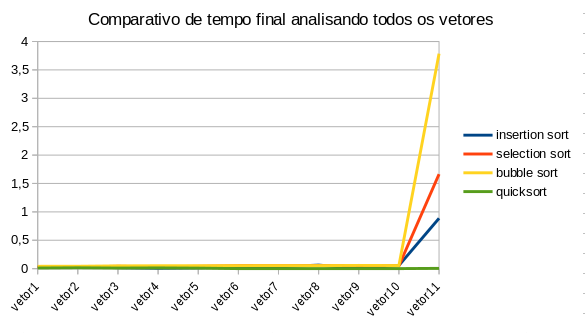
****

Imagem 3.2

Ao analisar a imagem 3.2 podemos observar que o método de ordenação *Quick sort* foi o que apresentou uma menor taxa de tempo para ordenar todos os vetores sem uma variação significativa. Logo após ele podemos ver que o *Insertion Sort* se coloca em segundo lugar em termos de tempo de ordenação apresentando uma pequena variação no Vetor 8.Em terceiro se coloca o método de ordenação *Selection Sort*,que não apresentou grandes variações do Vetor 1 ao 10,porém ficou atrás dos outros dois métodos (*Quick sort e Insertion Sort*) em relação ao Vetor 11. E por último,o método de ordenação *Bubble Sort* que apresentou significativos intervalos de tempo, principalmente no vetor 11 de 30 mil elementos.

**Tabela 1.Tempo em segundos (s) de cada método aplicado em cada um dos vetores.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Vetores | Insertion Sort | Selection Sort | Bubble Sort | Quicksort |
| Vetor 1 | 0.02627 | 0.02829 | 0.04177 | 0.01078 |
| Vetor 2 | 0.03773 | 0.02829 | 0.04244 | 0.01482 |
| Vetor 3 | 0.01735 | 0.04244 | 0.04918 | 0.01010 |
| Vetor 4 | 0.00741 | 0.03368 | 0.05188 | 0.01280 |
| Vetor 5 | 0.01487 | 0.04177 | 0.05188) | 0.00875 |
| Vetor 6 | 0.03272 | 0.05246 | 0.053094 | 0.00083 |
| Vetor 7 | 0.031849 | 0.052166 | 0.053938 | 0.000893 |
| Vetor 8 | 0.06266 | 0.052797 | 0.053818 | 0.000841 |
| Vetor 9 | 0.005018 | 0.052504 | 0.057108 | 0.000724 |
| Vetor 10 | 0.057188 | 0.05055 | 0.054966 | 0.000734 |
| Vetor 11 | 0.88665 | 1.66418 | 3.78765 | 0.004934 |

**Conclusão**

Com base no que foi discutido no desenvolvimento do artigo, tivemos a oportunidade de conhecer melhor os principais métodos de ordenação e suas características. Como por exemplo a diferença em segundos de um método para outro ao resolver um certo problema. Tanto em grupos pequenos de dados como em grupos grandes mesmo que por pouco tempo a olhos humanos, a diferença em muito grande para a máquina.

O artigo tratou uma breve introdução aos métodos de ordenação, contemplando o tema com uma abordagem genérica tendo em foco a utilização dos métodos de ordenação para resolução dos problemas com vetores. Nos testes foram observadas as diferenças de cada método, a forma que são feitas as interações em cada um deles, como o problema é tratado e a complexidade de tempo utilizando a notação *big* O, nos seus respectivos pior caso, médio caso e melhor caso de cada algoritmo.

**Referências**

Bruno, (2003) Devmedia, Algoritmos de ordenação: Análise e comparação. Encontrado em:

<https://www.devmedia.com.br/algoritmos-de-ordenacao-analise-e-comparacao/28261> Acessado em: 22/06/2018.

Fonseca, Mauro, Métodos de Ordenação. Encontrado em:

<http://dainf.ct.utfpr.edu.br/~maurofonseca/lib/exe/fetch.php?media=cursos:if63c:if63ced\_08\_ordenacao.pdf> Acessado em: 22/06/2018.

Vianna, Daniel, (2016) TreinaWeb, Principais algoritmos de Ordenação. Encontrado em:

<https://www.treinaweb.com.br/blog/conheca-os-principais-algoritmos-de-ordenacao/>

Acessado em: 24/06/2018.

Wikipédia, Quick Sort. Encontrado em: <https://pt.wikipedia.org/wiki/Quicksort>

Acessado em: 22/06/2018.

Wikipédia, Bubble Sort. Encontrado em: < https://pt.wikipedia.org/wiki/Bubble\_sort>

Acessado em: 22/06/2018.

Wikipédia, Selection Sort. Encontrado em: < https://pt.wikipedia.org/wiki/Selection\_sort >

Acessado em: 22/06/2018.

Wikipédia, Insertion Sort. Encontrado em: <https://pt.wikipedia.org/wiki/Insertion\_sort>

Acessado em: 22/06/2018.

# 