UNIVERSIDADE FEDERAL DO MARANHÃO

CIÊNCIA DA COMPUTAÇÃO

ESTRUTURA DE DADOS II

Profº: João Dallyson

Aluno: Paulo Gabriel Borralho Gomes

**RELATÓRIO DA ATIVIDADE PRÁTICA**

SÃO LUÍS - MA

2019.2

1. **INTRODUÇÃO**

As [linguagens de programação](https://www.infoescola.com/informatica/o-que-sao-linguagens-de-programacao/) são linguagens usadas para a comunicação com o computador. Estas linguagens são constituídas de comandos, que quando utilizados corretamente, executam uma ação. A programação nos computadores não tem uma data correta de início. Tudo começou na década de 30, com os primeiros computadores elétricos. Com o passar dos anos diversas linguagens foram surgindo como: FORTRAN, C, C++ Java, Python, entre outras.

As linguagens foram criadas justamente para que o homem pudesse através das máquinas, usar sequências lógicas para resolver um problema. Só que um problema pode possuir muitas situações diferentes, e dependendo da situação a maneira de resolver este problema pode ser melhor que outra maneira. Assim utiliza-se a análise de complexidade de algoritmos, para saber qual é o mais eficiente.

1. **ALGORITMO DE KRUSKAL**

É um algoritmo em teoria dos grafos que busca uma árvore geradora mínima para um grafo conexo com pesos. Isto significa que ele encontra um subconjunto das arestas que forma uma árvore que inclui todos os vértices, onde o peso total, dado pela soma dos pesos das arestas da árvore, é minimizado. Se o grafo não for conexo, então ele encontra uma floresta geradora mínima. Ao fim do algoritmo, a floresta tem apenas um componente e forma uma árvore geradora mínima do grafo. Mas ele requer um algoritmo que ordene a estrutura que tem os grafos.

Os algoritmos de ordenação utilizados foram: QuickSort, MergeSort, InsertSort, Quicksort + Inserção Parcial, Quicksort +Inserção Final, Mergesort + Inserção Parcial e Mergesort + Inserção Final. Onde o programa lera arquivos que contém as informações dos vértices, aplicara o algoritmo de ordenação escolhido pelo usuário e aplicara o Kruskal.

1. **QUICKSORT**

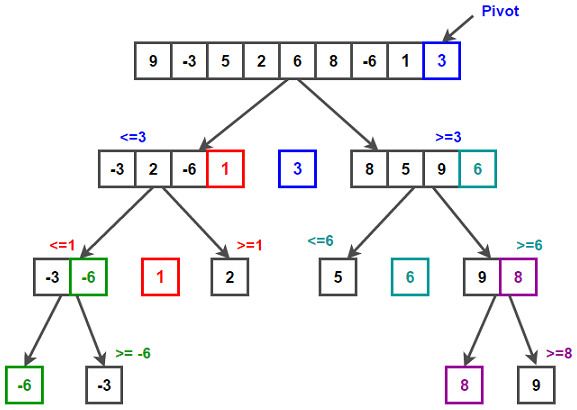
Este algoritmo adota a estratégia de “dividir e conquistar”, que consiste em rearranjar as chaves de modo que s chaves menores precedam as chaves maiores, em seguida ele ordena as duas sublistas de chaves menores e maiores recursivamente até que a lista completa se encontre ordenada. Ele possui três passo: escolher um pivô(no caso do algoritmo usado nos testes o pivô sempre é a mediana), particiona e coloque os elementos maiores que o pivô para a depois dele e os menores para antes dele, ao fim do processo o pivô estará na sua posição final e haverá duas sublistas não ordenadas, por conta disso, recursivamente estes processos serão aplicados as sublistas criadas, até que as sublistas tenham tamanho igual a um a recursão acontecera.

Figura 1. Exemplo de como funciona o QuickSort.

Ele possui uma complexidade de C:\Users\Cliente\Documents\ShareX\Screenshots\2019-09\chrome_NR0Tx0hfUw.png no seu pior caso, que é quando a lista é particionada de forma desbalanceada, e no seu melhor caso onde ele forma duas listas de tamanho não maior que n/2, sua complexidade é de .

1. **MERGESORT**

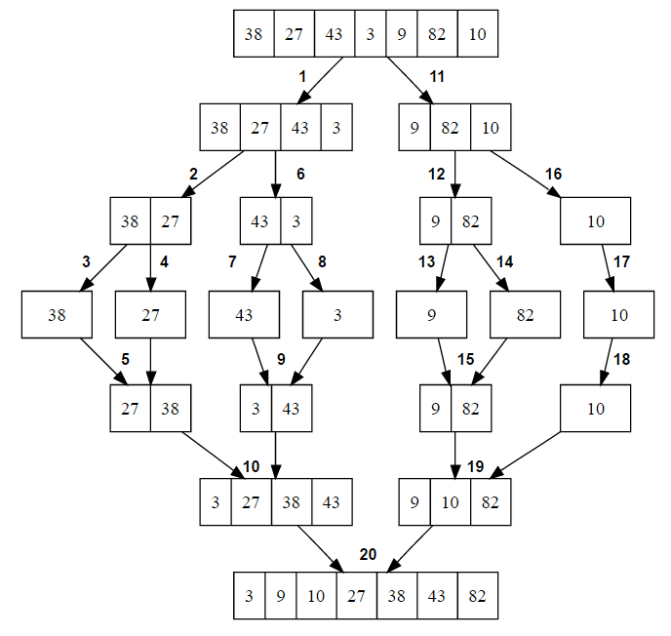
Também é um algoritmo que possui com estratégia o “dividir para conquistar”, sua ideia base é dividir o problema em vários subproblemas, dividindo a lista no meio, e recursivamente resolver dois subproblemas, cada um de tamanho n/2, e com todos os subproblemas resolvidos ocorre a conquista que é a união das resoluções dos subproblemas em um único conjunto ordenando. Sua complexidade é de C:\Users\Cliente\Documents\ShareX\Screenshots\2019-09\chrome_MpBYJyllDB.png, por conta disso ele se torna mais eficiente quando se tem uma grande quantidade de dados, já em casos que se tem uma pequena quantidade, outros algoritmos de ordenação se tornam mais eficientes que ele.

Figura 2. Exemplo de como funciona o MergeSort.

1. **INSERTSORT**

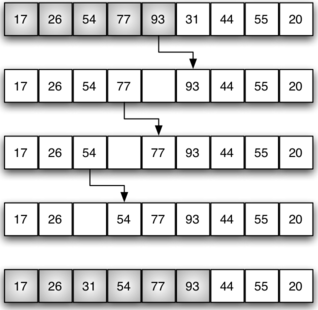
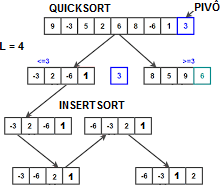
É um algoritmo que dado uma estrutura, como lista ou array, constrói uma matriz final com um elemento de cada vez, uma inserção por vez. Ele funciona dentro de laços de repetição onde a cada novo loop tem-se uma chave, que ira ser comparado aos outros elementos da estrutura, assim que encontrado um elemento menor que a chave, chave passara a ter o valor do valor menor, ou se a chave percorrer a estrutura inteira e não encontrar nenhum elemento menor significa que ele já esta na sua posição final e depois atribuindo o valor da chave ao seu próximo elemento, fazendo isso repetidas vezes no fim terá a estrutura ordenada. Ele é um algoritmo muito eficiente para se utilizar quando se tem poucos dados, podendo ter complexidade de C:\Users\Cliente\Documents\ShareX\Screenshots\2019-09\chrome_g9jmPe17qa.png, mas no seu pior caso, que é quando os valores estão em ordem inversa, ele tem uma complexidade de C:\Users\Cliente\Documents\ShareX\Screenshots\2019-09\chrome_NR0Tx0hfUw.png.

Figura 3. Execução de um dos passos do InsertSort.

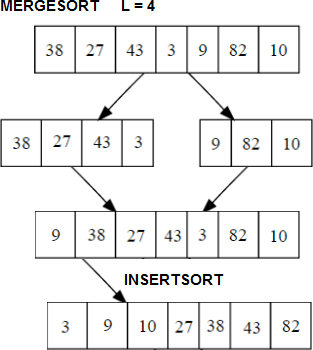
1. **QUICKSORT /MERGESORT + INSERÇÃO PARCIAL**

Inicialmente estes algoritmos perguntam ao usuário o tamanho de L, os algoritmos são executados recursivamente, como padrão, mas quando obtiverem uma partição de tamanho menor ou igual a L, essa partição deverá ser imediatamente ordenada usando o algoritmo de InsertSort.

No caso do MergeSort, assim que ordenado com o InsertSort essa partição devera ser efetivamente mesclada, como o MergeSort funciona normalmente.

Figura 4. Execução do QuickSort + Inserção Parcial.

1. **QUICKSORT/MERGESORT + INSERÇÃO FINAL**

Assim como o algoritmo anterior, ele começa perguntando o tamanho de L, mas quando ele encontra a partição de tamanho menor ou igual a L, o algoritmo para imediatamente.

No QuickSort, assim que isso ocorrer ele continua particionando as partições maiores, e uma vez que todas as partições tenham tamanho menor ou igual a L, o algoritmo para e executa o

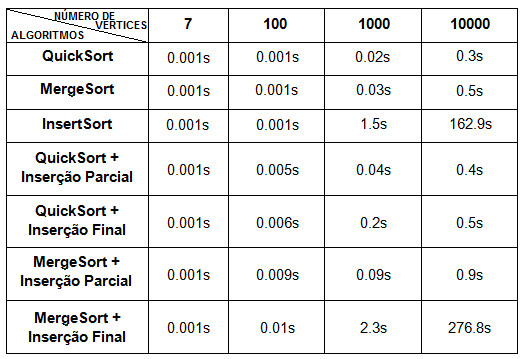
algoritmo InsertSort.

Figura 5. Execução do MergeSort + Inserção Final.

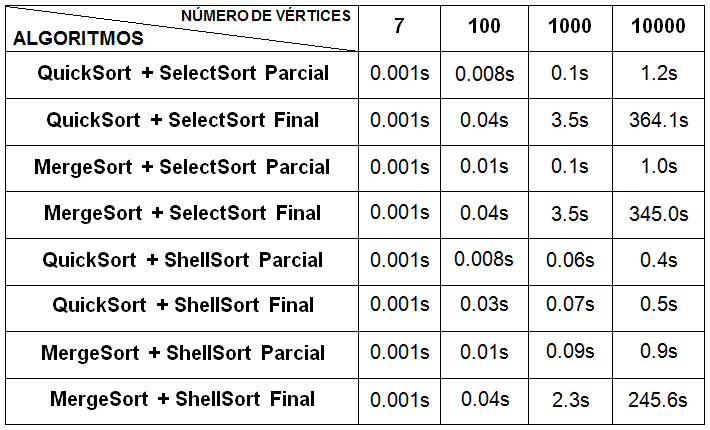
Já no MergeSort, ele mescla as listas com tamanho menor ou igual a L, retornando uma lista parcialmente ordenada, ai o algoritmo InsertSort é aplicado nessa lista.

1. **TESTES**

Os programas citados foram testados, e a tabela abaixo mostra seus resultados, onde no caso dos algoritmos de MergeSort/QuickSort + Inserção Parcial/Final, o tamanho de L foi de: para 7 vértices o L foi igual a 2, para 100 vértices o L foi igual a 10, para 1000 vértices o L foi igual a 50 e para 10000 vértices o L foi igual a 100. Os arquivos que possuíam uma grande quantidade de dados não foram utilizados nos testes, pois em certos algoritmos, como o de InsertSort, demorariam muito tempo para concluírem sua execução.



1. **SHELLSORT E SELECTSORT**

Basicamente o algoritmo ShellSort passa várias vezes pela lista dividindo o grupo maior em menores, nos grupos menores é aplicado o método da [ordenação por inserção](https://pt.wikipedia.org/wiki/Insertion_sort). E o SelectSort é composto por dois laços, um laço externo e outro interno, o laço externo serve para controlar o índice inicial e o interno percorre todo o vetor, na primeira iteração do laço externo o índice começa de 0 e cada iteração ele soma uma unidade até o final do vetor e o laço mais interno percorre o vetor começando desse índice externo + 1 até o final do vetor.

Substituindo o algoritmo InsertSort pelos algoritmos ShellSort e SelectSort, obtemos os seguintes resultados:

O tamanho de L foram os mesmo onde foi utilizado o InsertSort, já explicados anteriormente. Como se pode perceber somente em pouquíssimos casos houve uma melhora de tempo de execução, pois os dois são algoritmos não estáveis, diferente do InsertSort, portanto é melhor usar um algoritmo estável.

1. **CONCLUSÃO**

Algoritmos de ordenação tem um papel muito importante na computação, quase que essencial utilizá-los para melhorar outros algoritmos, como Kruskal, sem falar que eles tornam uma busca muito mais fácil de ser feita.

O programa esta rodando bem, depois de dificuldades em ter que aprender Python e na implementação dos algoritmos parcial e final, principalmente o QuickSort + Inserção Parcial.

1. **REFERÊNCIAS**

<http://www.geocities.ws/digitosistemas/download/complexidade.html>

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

<https://stackoverflow.com/questions/56403924/quick-sort-implementation-in-python-with-middle-element-as-pivot>

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

<https://www.java67.com/2018/03/mergesort-in-java-algorithm-example-and.html>

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

<https://www.youtube.com/watch?v=BSXIolKg5F8>

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

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

<https://www.youtube.com/watch?v=QnHOwrZllXk>