UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

CALEBE SUCUPIRA DE OLIVEIRA
JOSUÉ SUCUPIRA DE OLIVEIRA

ALGORITMOS DE ORDENAÇÃO

QUIXADÁ/CE
2022

ALGORITMOS DE ORDENAÇÃO

- BUBBLESORT.

O algoritmo de ordenação bubblesort, ou ordenação em bolha, percorre um vetor e ordena os elementos contidos. O algoritmo consiste em fazer trocas fazendo com que o menor elemento fique à esquerda. A comparação é feita a partir do último elemento do vetor (sendo ele um valor x) com um elemento a sua esquerda(y), se x for menor que o y então vai haver um troca entre eles, e x vai ser comparada denovo com um elemento que está em sua esquerda, se x ainda for menor a troca vai ser feita, senão o próximo elemento que vai ser comparado. Depois que as primeiras comparações tiverem sido feitas, o primeiro elemento já estará ordenado, então o último elemento vai ser comparado com um elemento à esquerda e vai ficar fazendo as trocas enquanto a condição for satisfeita. A execução para quando o vetor já estiver ordenado. Esse algoritmo tem complexidade de $O(n^2)$ em seu pior caso, e $O(n)$ em seu melhor caso, que é quando um vetor já está ordenado.

- INSERTIONSORT

O algoritmo de ordenação insertionsort, ou ordenação por inserção, tem ideia de organização de cartas de um baralho, cada vez que uma carta é puxada, ela é comparada com as demais e inserida no local correto. O algoritmo consiste em fazer comparações com um elementos e todos que tiverem a esquerda do elemento, fazendo com que o menor elemento fique a esquerda. É considerado que o primeiro elemento do vetor já está ordenado, porque não existe nenhum elemento à esquerda dele. Assim, a comparação é feita a partir da posição 1 do meu vetor até n. Esse algoritmo tem complexidade de $O(n^2)$ em seu pior caso, e $O(n)$ em seu melhor caso.

- SELECTIONSORT

O algoritmo de ordenação selectionsort, ou ordenação por seleção, faz a busca da posição do menor elemento e faz uma troca com a primeira posição do vetor, sendo a cada elemento que é encontrado a posição inicial é incrementada para realizar a troca. O algoritmo considera que o primeiro elemento do vetor é o menor, em seguida esse elemento vai ser comparado com todos os elementos do vetor e a cada elemento menor encontrado o índice do menor elemento do vetor é armazenado e só depois de verificar que é o menor elemento, é feita uma troca entre o menor elemento com o elemento que está na primeira posição do vetor. Esse processo acontece até o vetor estar totalmente ordenado. Esse algoritmo tanto em seu pior melhor caso é de $O(n^2)$, porque ele não tem condição de parada, todo esse processo vai acontecer independente se o vetor já estiver ordenado.

- MERGESORT

O algoritmo de ordenação mergesort, tem o método de divisão e conquista, onde o problema é divido em problemas menores para facilitar a resolução. Ele é dividido em duas partes, uma com a função mergesort que é onde vetor é dividido, e uma outra que é a função intercala, que depois que o vetor estiver dividido compara quem é o menor elemento e

intercala. A ideia da função intercala é intercalar dois subvetores que já estão ordenados, e isso acontece por causa da função mergesort, que através de chamadas recursivas divide o vetor ao meio até conter apenas um elemento, porque um vetor que só tem um elemento já está ordenado. A função intercala funciona da seguinte maneira: dois subvetores já ordenados vão ser intercalados e armazenados em um vetor auxiliar, a comparação que é feita é se o primeiro elemento de um subvetor X é menor que o primeiro elemento do subvetor Y, se a condição for verdadeira o primeiro elemento do sub vetor X é armazenado na primeira posição do vetor auxiliar, caso contrário o primeiro elemento do subvetor Y que é o menor e é armazenado no vetor auxiliar, esse processo ocorre até todos os elementos já estiverem no vetor auxiliar. Esse algoritmo tanto em seu pior melhor caso é de O(nlgn).

- QUICKSORT

Como o mergesort, o quicksort também tem a metodologia de divisão e conquista. A forma como o algoritmo funciona é da seguinte estratégia: O QuickSort divide o vetor de entrada em dois subvetores a partir de um pivô, em seguida realiza o mesmo procedimento nos dois vetores menores até o caso trivial, que é onde está meu pivô. A escolha do pivô é feita na função separa, nela o pivô é o último elemento do vetor e a ideia é colocar esse elemento na posição correta, fazendo com que todos os elementos menores ou iguais a ele fique a esquerda e os elementos maiores que ele fica a direita, independe da ordem que ficar a direita ou esquerda. Através de chamadas recursivas na função quicksort o vetor fica ordenado, pois a cada chamada o vetor é dividido pelo pivô, e o mesmo vai acontece nas partes que foi dividida, até que fica um vetor unitário onde o mesmo está na posição correta do vetor. Esse algoritmo tem complexidade de $O(n^2)$ em seu pior caso, que é em casos quando o vetor já está ordenado, e o melhor caso é O(nlgn) que é quando a esquerda do meu pivô tem n/2 elementos e na direita do pivô quando tem n/2 -1 elementos, isso mostra que a escolha do pivô implica no desempenho do algoritmo.

COMPARAÇÃO ENTRE OS ALGORITMOS ITERATIVOS E RECURSIVOS

Todos os algoritmos que foram estudados têm a mesma finalidade, ordenar um vetor. Acontece que para diferentes casos, ou seja, das diversas formas que o vetor esteja ele pode influenciar no desempenho do algoritmo. Isso influencia no tempo de execução, quantidade de comparações e trocas feitas no vetor. As principais diferenças entre os os cinco algoritmos estudados, está justamente ligado ao desempenho de cada algoritmo, de como um problema pode ser resolvido de formas diferentes, e que existe uma forma pior ou melhor para a resolução do problema.

Dos algoritmos estudados, cada um tem sua forma iterativa e recursiva. A dos recursivos ideia é bem semelhante à dos interativos, porque as mesmas comparações são feitas, com diferença das chamadas recursivas.

Bubble sort: Umas das diferenças entre o bubble sort recursivo e iterativo é que enquanto o interativo faz trocas do último elemento até o primeiro utilizando dois ‘for’ um mais externo

que vai até n-1, e uma mais interno que vai do último elemento até o primeiro, que através de trocas o menor elemento fique na primeira posição. O recursivo pega o maior elemento e joga para o final do vetor e na chamada recursiva o vetor é reduzido, tirando a parte que já está ordenada, que é onde está localizado o último elemento.

Insertion sort: A ideia do insertion recursivo não é diferente. Diferente do bubble sort que a chamada recursiva é feita no final, no insertion sort a chamada recursiva é feita no começo. No começo do insertion sort recursivo o vetor vai ser empilhado, e para cada pilha o vetor vai ficando com um elemento a menos até o vetor ficar com apenas um elemento. Depois que tiver que o vetor tiver com um elemento é que as comparações vão ser feitas, e o vetor que foi empilhado vai começar a ser desempilhado até o vetor ficar ordenado.

Selection sort: Uma das diferenças do selection sort iterativo para o recursivo é o parâmetro usado, enquanto no iterativo é passado o vetor e a quantidade de elementos, no recursivo é preciso passar o vetor, o início do vetor e o final do vetor. As comparações feitas não são diferentes, depois de percorrer o vetor e encontrar o índice do menor elemento a troca é feita para posição inicial do vetor, depois disso é feita a chamada recursiva diminuindo o tamanho do vetor pela esquerda. A parte que é reduzida é onde se encontra o elemento que já está ordenado, o vetor para de ser reduzido até ficar com um elemento e quando isso acontece o vetor por completo já estará ordenado.

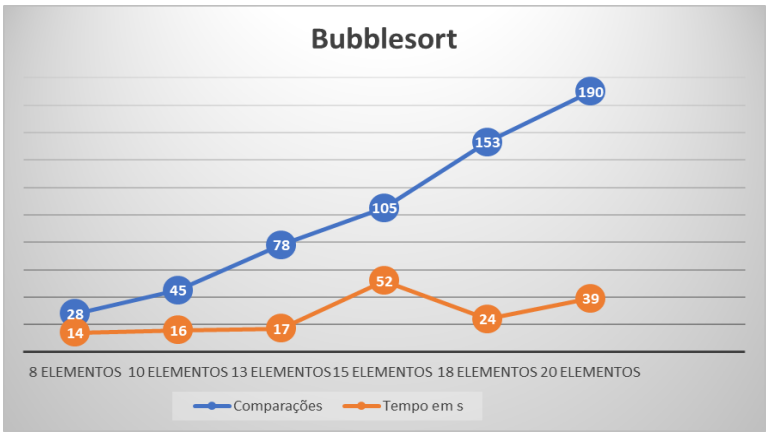
Merge Sort: O merge sort iterativo e recursivo utiliza a mesma função auxiliar, a função intercala, que tem como trabalho de intercalar dois subvetores já ordenado. A principal diferença do recursivo e iterativo é que enquanto no recursivo o vetor primeiro é dividido até até ficar sub vetores unitários, no iterativo é considerado que os elementos do vetor já estão divididos em sub vetores unitários. Por exemplo, considerando um vetor de 6 elementos, no recursivo o primeiro passo seria dividi-lo ficando dois sub vetores de 3 elementos, depois esses mesmos sub vetores seriam divididos até ficar um vetor unitário.

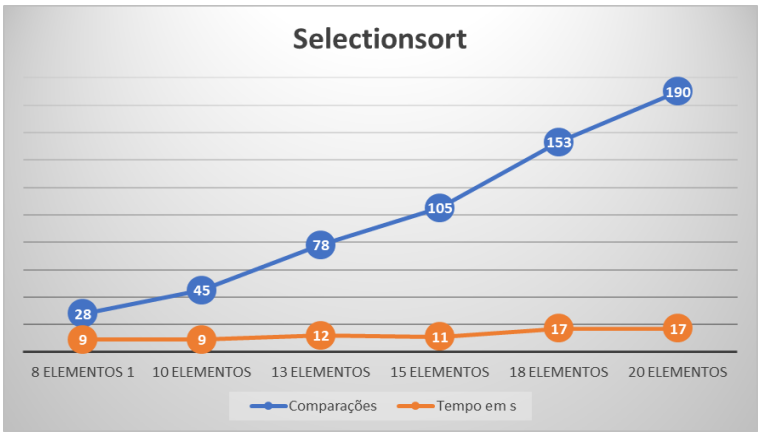
GRÁFICOS E ANÁLISE

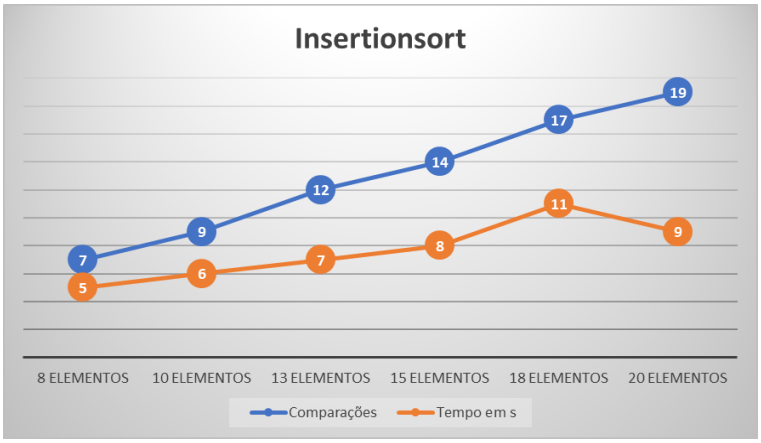
A análise de cada algoritmo foi feito com seis diferentes tamanho de vetores e cada um dos seis vetores, foram vetores aleatórios gerados pela função (randomVector) que se encontra no código. Os algoritmos analisados foram: bubblesort, insertionsor e selectionsort iterativos em oredenação crescentes, mergesort e quicksort recursivos em oredenação crescente. Foram analisados os seguintes vetores: 8 elementos = {68, 95, 90, 92, 68, 48, 6, 82}, 10 elementos = {38 ,49, 85, 52, 22, 40, 3, 4, 99, 67}, 13 elementos = {92, 81, 52, 30, 40, 8, 43, 34, 69, 46, 32, 7, 71}, 15 elementos = {72, 60, 7, 2, 20, 99, 71, 67, 75, 47, 42, 22, 85, 18, 5}, 18 elementos = {48, 66, 62, 17, 69, 4, 12, 71, 61, 39, 80, 11, 40, 68, 44, 79, 67, 10}e 20 elementos = {9, 10, 35, 29, 98, 32, 93, 46, 23, 33, 18, 31, 21, 56, 54, 59, 25, 34, 62, 68}.

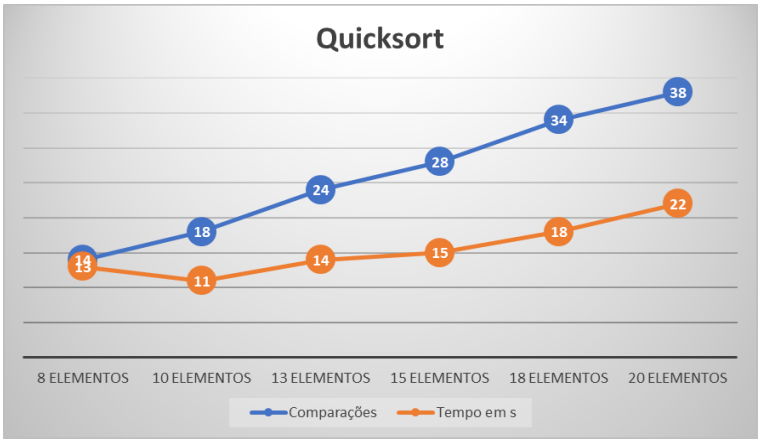
Em cada algoritmos analisamos o tempo de execução, com a ajuda da função steady_clock::now(); da biblioteca chrono e a quantidade de comparações feitas.

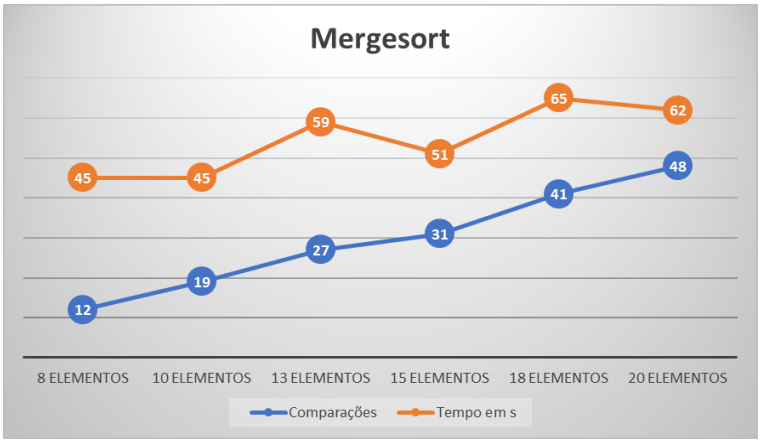
OBS: O tempo de execução mostrado nos gráficos é tempo x 10^-7 em segundos.











No bubblesort é possível ver que o tempo de execução com 15 elementos foi maior do que o de 18 e 20 elementos, que logicamente era para ser ao contrário, mas isso se deu pelo fato de que o segundo menor elemento do vetor estar na última posição, sendo necessário de mais trocas. No selectionsort os números de comparações feitas são iguais com a do bubble, e isso se dá pelo fato da complexidade tanto no melhor ou pior caso ser $O(n^2)$. No insertionsort ele foi melhor que os outros no tempo de execução e quantidade de comparações, o insertion não precisa fazer muitas comparações, pois a sua lógica já coloca o elemento do vetor no lugar correto. O algoritmo mergesort, por mais que sua complexidade seja O(nlgn), foi o que teve que maior tempo de execução, isso acontece devido dois fatores, a forma que os elementos foram posicionados e devido também ao computador que foi feito a execução, esse dois fatores podem ter causado esse maior tempo de execução. O quicksort ele foi basicamente o único que seguiu que seguiu um "padrão" conforme os números de elementos foram aumentando o tempo de execução também.

COMO O TRABALHO FOI DIVIDIDO:

O desenvolvimento do trabalho foi bem dividido entre as duplas, o primeiro passo do desenvolvimento foi fazer todas as implementações do código. O processo de implementação foi dividido em duas partes, a implementação iterativa e recursiva. Depois das implementações dos algoritmos, foi feito o trabalho de organização código, comentando em cada função dizendo o que ela fazia, e em seguida a implementação do TAD. A parte final do trabalho foi fazer o relatório que também foi dividido em duas etapas, a parte das comparações dos algoritmos, testes, contagem de comparações e tempo de execução, e a segunda parte elaboração dos demais tópicos do trabalho, sendo que para cada etapa dividida, uma membro da dupla ficou responsável

OTIMIZAÇÕES FEITAS NO CÓDIGO

A lógica usada para todas as implementações de ordenação ímpares crescente e pares decrescente nos algoritmos em geral foi a mesma, respeitando cada forma como o algoritmo trabalha. A comparação é baseada em três, quando tiver um número par e impar, par e par, e impar e impar. A preferência sempre é os números ímpares quando se compara entre par e um ímpar, quando é no caso de dois ímpares, é verificado quem é o menor e no caso de dois pares é verificado o maior.

Na parte de ordenação de um vetor de estrutura, as modificações que foram feitas foram poucas, e também houve a utilização de funções da biblioteca string, para auxiliar na comparação para ordenar em ordem alfabética. Uma função 'troca' também teve que ser implementada, já que a troca se tratava de um vetor de estrutura.

DIFICULDADES ENCONTRADAS

A maior dificuldade encontrada foi durante a implementação do algoritmo, na parte ordenação ímpares crescente e pares decrescente, foi um processo bem demorado nessa parte, a dupla passou muito tempo presa, sem conseguir fluir uma solução boa para implementação dos algoritmos. Depois que a dupla realmente conseguiu entender de como as comparações deveriam ser feitas, em que casos é para haver troca ou não ficou muito fácil implementar nos algoritmos. Por mais que a dupla tenha ficado muito tempo nessa parte do trabalho, foi necessário para o entendimento de cada passo dos algoritmos.

.