Equipe: Francisco Hartur Lopes de Alcântara e Antonio Everton Cosmo

Professor: Roberto Cabral Rabelo Filho

Universidade Federal do Ceará – Campus Crateús

Descrição do Trabalho

O trabalho segue uma seção de explicação de todos algoritmos de ordenação implementados no código e sobre a eficiência deles, e as comparações entre os algoritmos, contendo os gráficos que representam a competência dos algoritmos para diferentes tamanhos de entrada. Outro tópico de comparação entre recursivos e interativos usando vetor e lista e pôr fim a comparação entre os algoritmos que usam vetor e os que usam lista (quem tem menos ciclos). Além disso um tópico de como o trabalho foi dividido e as seções de dificuldades encontradas.

Descrição dos Algoritmos

Bubble Sort a ideia dele é comparar pares de elementos adjacentes e os troca de lugar se estiverem na ordem errada, esse processo se repete até que mais nenhuma troca seja necessária (elementos já ordenados).

Performance:
Melhor caso: O(N);
Pior Caso: O(N^2);
Não recomendado para grandes conjuntos de dados.

Insertion Sort a ideia é similar a ordenação a cartas de baralho com as mãos, pega-se uma carta de cada vez e a coloca em seu devido lugar, sempre deixando as cartas da mão em ordem.

Funcionamento: Ele percorre um vetor de elementos da esquerda para a direita e à medida que avança vai ordenando os elementos à esquerda.

Performance:
Melhor Caso: O(N)
Pior Caso: O(N^2)
Eficiente para conjuntos pequenos de dados.

Selection Sort a ideia dele é procurar o menor valor do array e o coloca na primeira posição do array. Descarta-se a primeira posição do array e repete-se o processo para a segunda posição, isso é feito para todas as posições do array.

Funcionamento: Consiste em selecionar o menor valor e colocar na primeira posição, selecionar o segundo menor item e colocar na segunda posição, segue estes passos até que reste um único elemento.

Performance:
Melhor Caso: O(N^2);
Pior Caso: O(N^2)
Ineficiente para grandes conjuntos de dados

Merge Sort a ideia dele é dividir e conquistar, divide recursivamente o conjunto de dados até que cada subconjunto possua 1 elemento, combina 2 subconjuntos de forma a obter 1 conjunto maior e ordenado, esse processo se repete até que exista apenas 1 conjunto.

Performance:
Melhor Caso: O(N log N);
Pior Caso: O(N log N);
Desvantagem no algoritmo recursivo ele usa um vetor auxiliar durante a ordenação

Heap Sort a ideia dele é utilizar uma heap, para ordenar os elementos à medida que os insere. 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.

Performance:
Melhor Caso: O(N log 2N);
Pior Caso: O(N log 2N);

Quick Sort a ideia dele é dividir e conquistar, um elemento é escolhido como pivô. Usa-se uma função partition que os dados são rearranjados (valores menores do que o pivô é colocado antes dele e os maiores depois). Recursivamente ele ordena as duas partições.

Performance:
Melhor Caso: O(N log N);
Pior Caso: O(N log N);
Desvantagem dele é na hora de escolher o pivô.

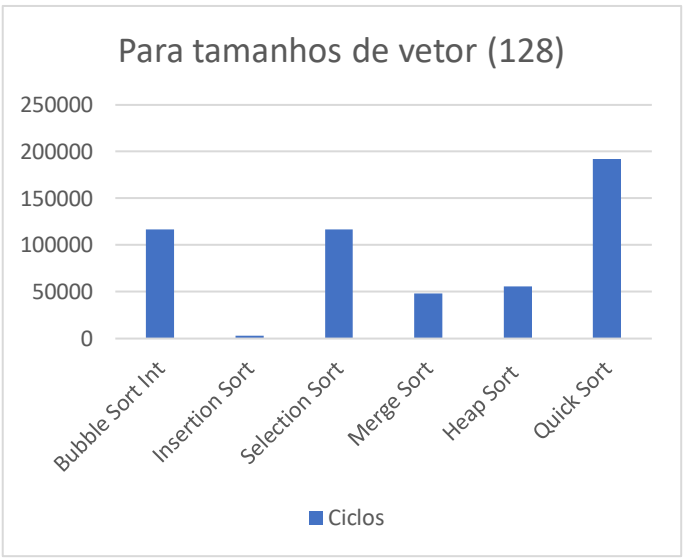
Sobre eficiência do Algoritmos

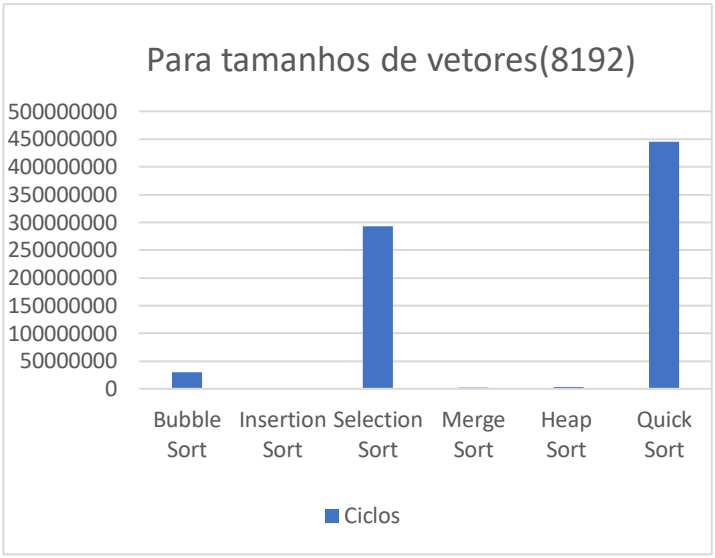
No código adicionamos outras funções para fazer as comparações entre os algoritmos uns modelos diferentes entre o bubble o heap e outros, obtemos uma surpresa nas comparações, o bubble feito usando do while obteve menos ciclos do que o bubble sort normal, sendo mais eficiente.

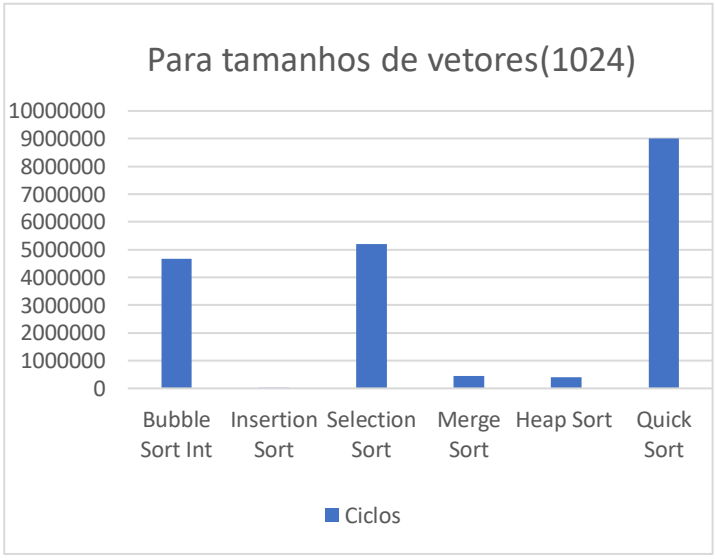
Comparando a Heap recursiva, com a outra segunda versão obtemos que a Heap recursiva é mais eficiente que a segunda Heap implementada, pois ela apresentou menos ciclos para rearranjar um vetor.

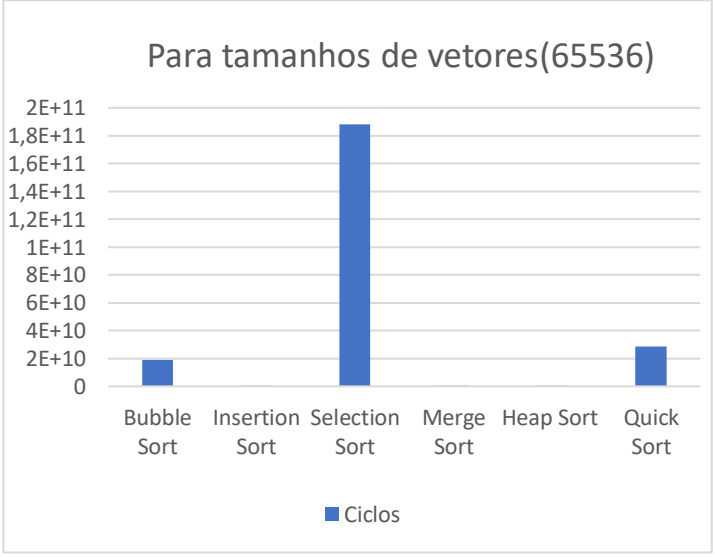
Gráfico de Comparações

Velocidade dos algoritmos de ordenação interativos

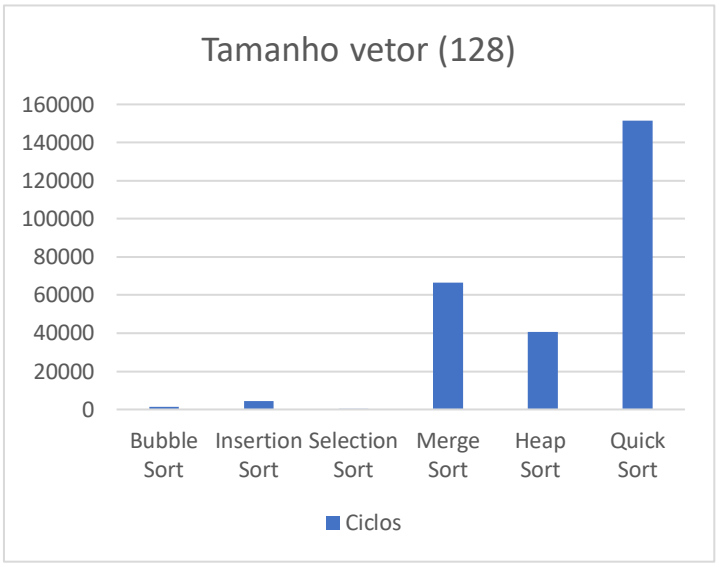


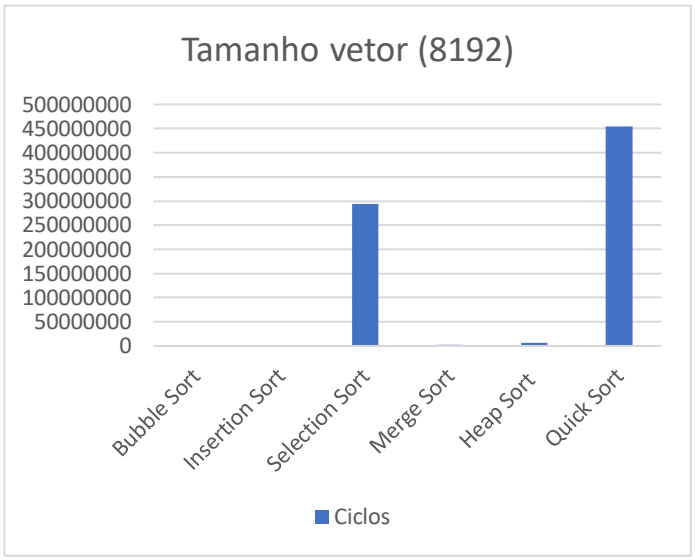


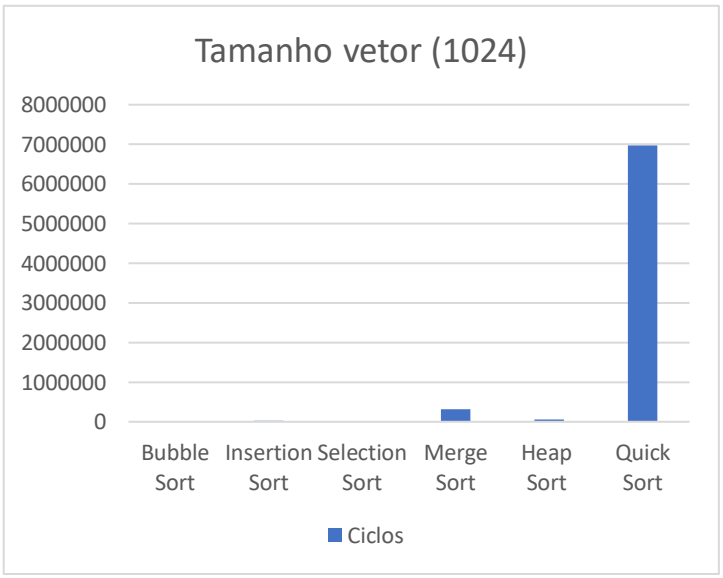


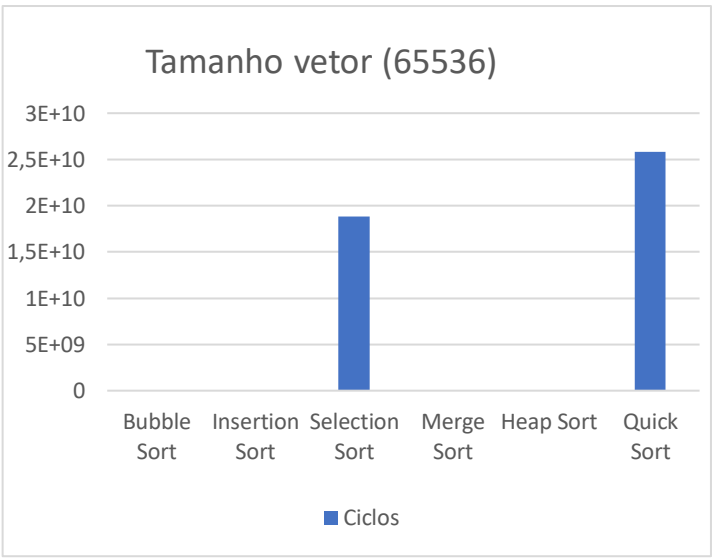


Velocidade dos algoritmos de ordenação recursivos

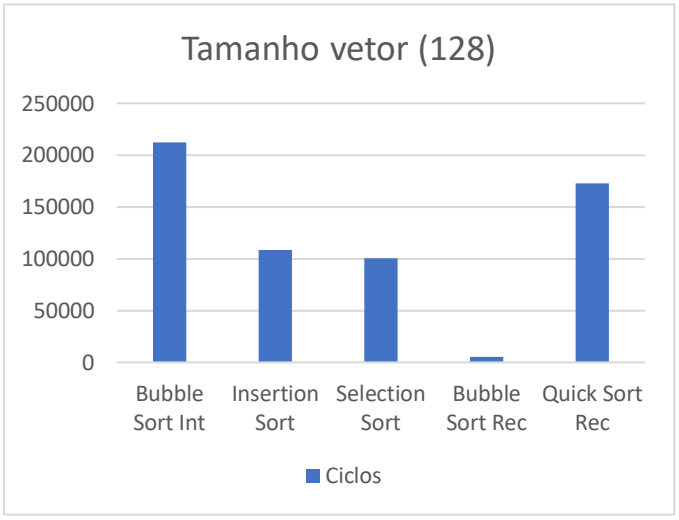


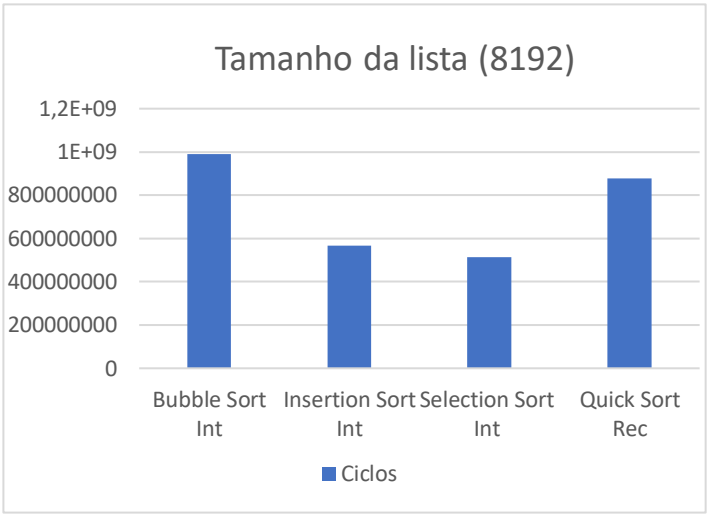


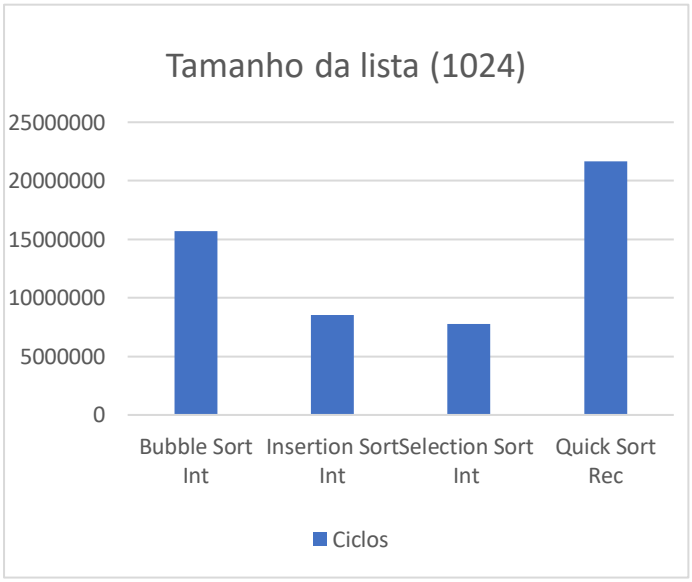


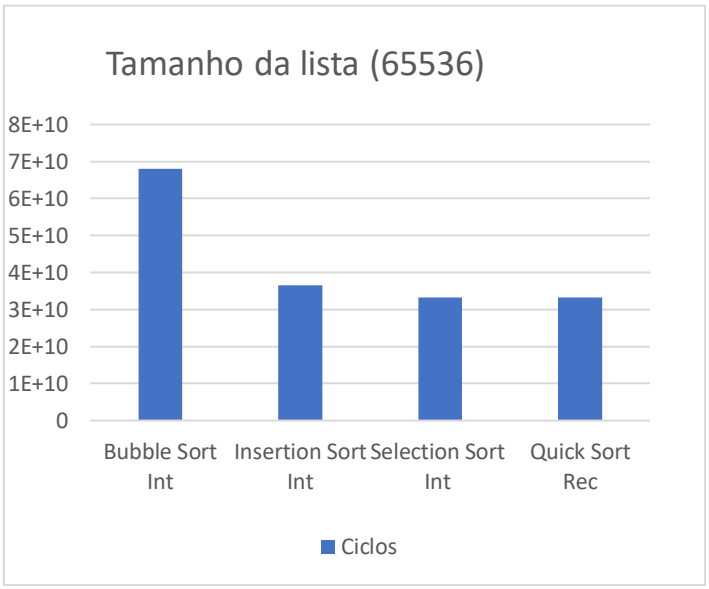


Velocidade dos algoritmos de ordenação lista duplamente encadeada









Comparação entre os algoritmos Interativos e Recursivos

Na comparação entre o Bubble Interativo e Recursivo: Em geral obtemos os resultados: O recursivo levou menos tempo de ciclos que o interativo, ou seja, melhor. O recursivo apresentou problemas a partir do tamanho sendo 1024, sendo obtido o somente resultados do interativo.

Na comparação entre o Insertion Interativo e Recursivo: Em geral obtemos os resultados: O interativo levou menos tempo de ciclos do que o recursivo.

Na comparação entre o Selection Interativo e Recursivo: Em geral obtemos os resultados: O recursivo levou menos tempo de ciclos que o interativo, ou seja, melhor.

Na comparação entre o Merge Interativo e Recursivo: Em geral obtemos os resultados: O interativo levou menos tempo de ciclos do que o recursivo.

Na comparação entre o Heap Interativo e Recursivo: Em geral obtemos os resultados: O interativo levou menos tempo de ciclos do que o recursivo.

Na comparação entre o Quick Interativo e Recursivo: Em geral obtemos os resultados: O recursivo levou menos tempo de ciclos que o interativo, ou seja, melhor.

O estranho é que em geral de todo os resultados obtidos o que teve o melhor desempenho foi o insertion sort.

Comparação entre os algoritmos de Vetor e de Lista

Conseguimos implementar alguns algoritmos, fazendo comparações entre eles os que contem no código versão lista são os três algoritmos elementares interativos e uma versão Bubble recursiva o Quick recursivo não foi implementado por nós estando lá para fazer comparações, os resultados obtidos foram que os de vetores foram mais rápidos que os lista, ou seja, levaram menos ciclos.

Divisão do trabalho

Foram realizados encontros semanais para discussão do trabalho, onde foi subdividido as tarefas, cada um tentava fazer os algoritmos usando vetor, assim concluímos qual foi a melhor implementação usando vetores, e juntamos em um só arquivo, em relação a lista foi necessário trabalho em conjunto para pensarmos em como fazer o código em lista duplamente encadeada, tendo sucesso em alguns outros não. Além disso, o encontro da equipe para fazer esse relatório para ser entregue juntamente com o código

Seção de dificuldades

Não foi nós dado instruções de como funcionava o bench.h, foi difícil usá-lo, após uma análise deu certo para os algoritmos de vetores, mas foi necessário usar às vezes uma função que chama outra para poder entrar na função (bench).

Em lista foi necessário fazer novas funções que cria números aleatórios para lista e outra função verifica se a lista está ordenada, após ter concluído isso, foi discutido várias horas de como implementar os algoritmos pedidos, felizmente não podemos trocar os valores complicando mais, fizemos um swap que troca os ponteiros das listas e assim fomos pensando e mais funções.

Com o curto prazo tivemos sucesso em algumas funções, mas para algoritmos como MergeSort e HeapSort requeria muito tempo, através de pesquisas encontramos uma implementação do QuickSort sua complexidade é muito grande, mas resolvemos colocar para comentar sobre ele no trabalho.