UNIVERSIDADE
FEDERAL DO CEARÁ
CAMPUS DE QUIXADÁ

Ulisses Queiroz da Silva - 471285
Felipe Gabriel da Silva - 475189

Trabalho 3
Estrutura
de
Dados

Prof. Atílio Gomes Luiz

Quixadá-CE
20/06/2019

Sumário

Introdução

Neste trabalho foi solicitado que fosse implementado na linguagem de programação c++, todos os códigos de ordenação demonstrados em sala de aula, dentre eles: BubbleSort, InsertionSort, SelectionSort, MergeSort, HeapSort e QuickSort.

• Uma versão iterativa e uma versão recursiva para cada um desses seis algoritmos usando vetor;

• Uma versão (iterativa ou recursiva) para cada um desses seis algoritmos usando lista duplamente encadeada.

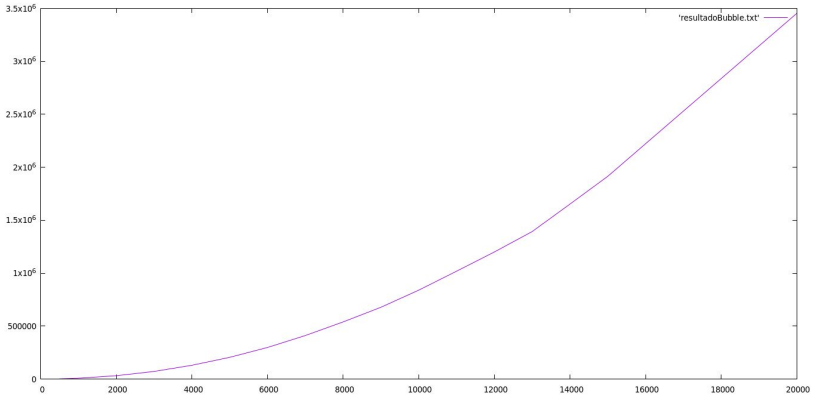
Ademais, este relatório com arquivos de gráficos que se originaram de arquivos.txt. Este relatório tem como intuito comparar os algoritmos de ordenação já supracitados.

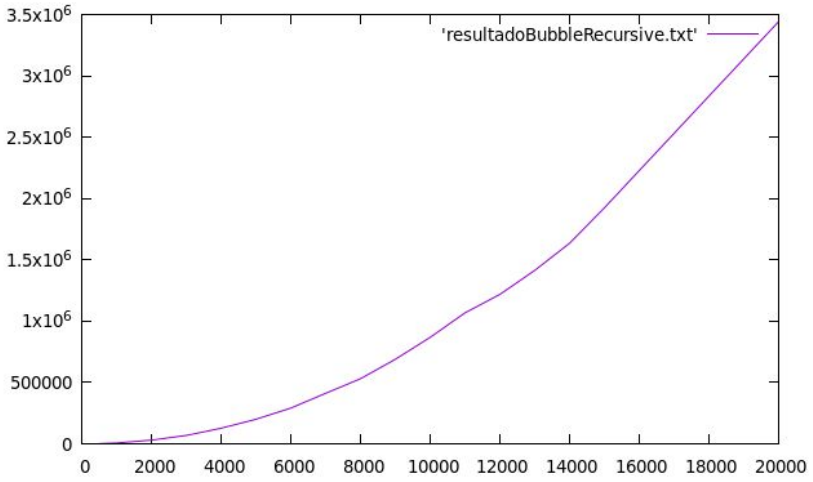
Ordenação Vetor

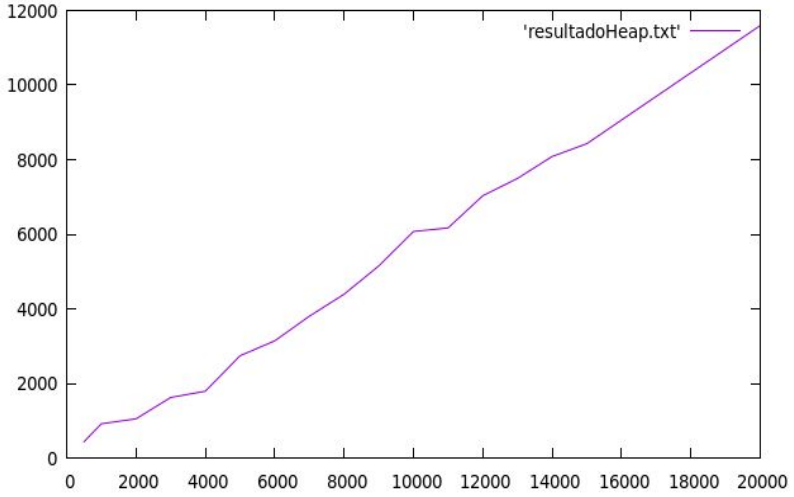
Como foi solicitado no pdf que foi disponibilizado com as instruções do trabalho foi solicitado que fosse utilizado no mínimo o tamanho do vetor N = {1000, 5000, 10000, 50000, 100000,500000, 1000000}. Entretanto, devido ao baixo desempenho de meu computador não consegui executar os seguintes algoritmos de ordenação: Bubblesort, Insertionsort.

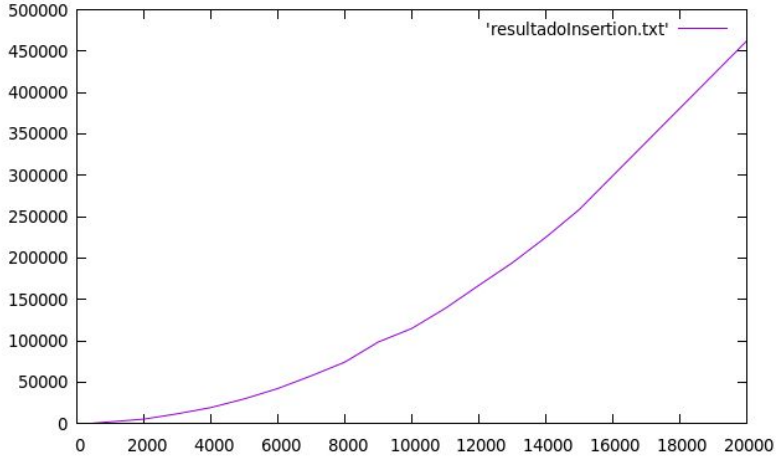
Com os dados e testes a seguir é notório a disparidade da diferença de desempenho nos casos de testes feito, a recursividade, principalmente em mergesort, quicksort e heapsort, toma a sua vantagem que chega a ser visível até mesmo em vetores de tamanho médio e em tamanhos grande como foi solicitado no pdf são os únicos que conseguem dar resultados.

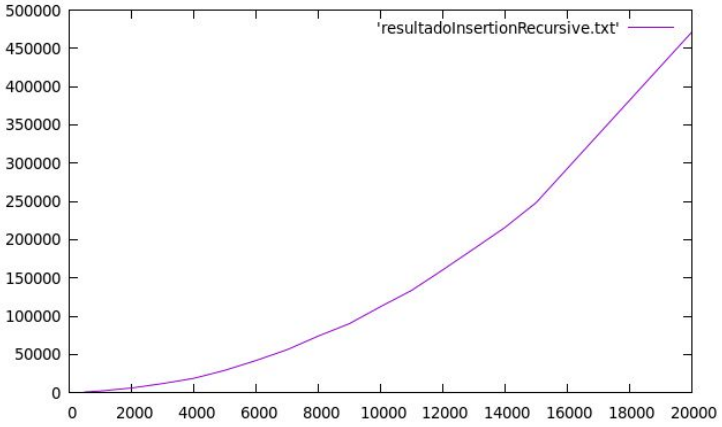
Gráficos de Desempenho

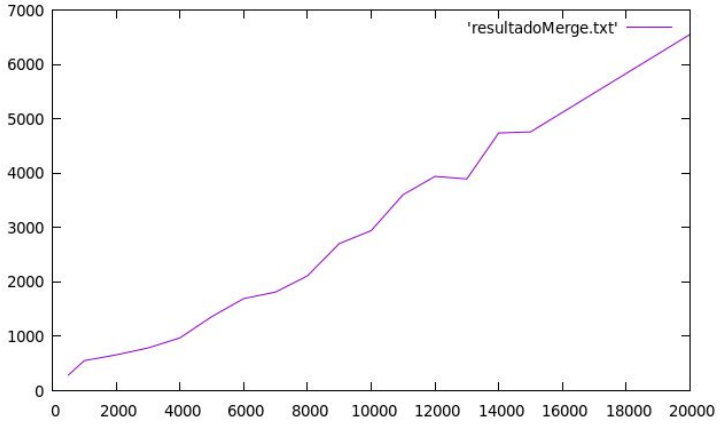


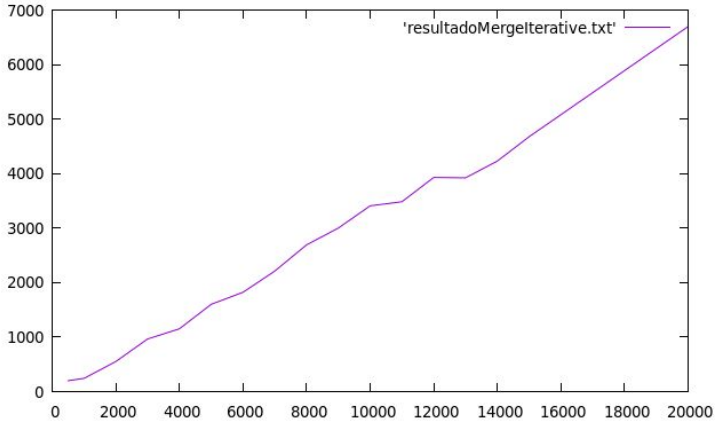

'resultadoBubbleRecursive.txt'
3.5x10^6
3x10^6
2.5x10^6
2x10^6
1.5x10^6
1x10^6
500000
0
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
20000

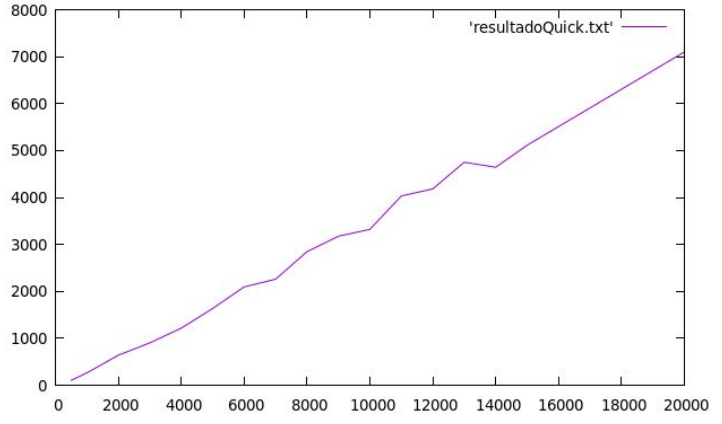

'resultadoHeap.txt'
12000
10000
8000
6000
4000
2000
0
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
20000

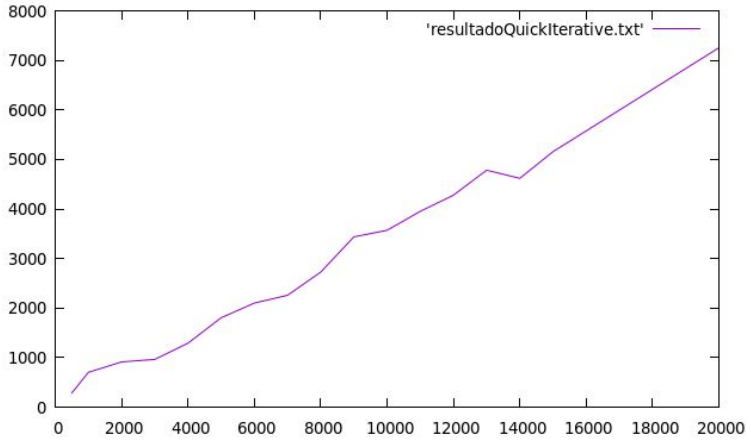

'resultadoInsertion.txt'
500000
450000
400000
350000
300000
250000
200000
150000
100000
50000
0
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
20000

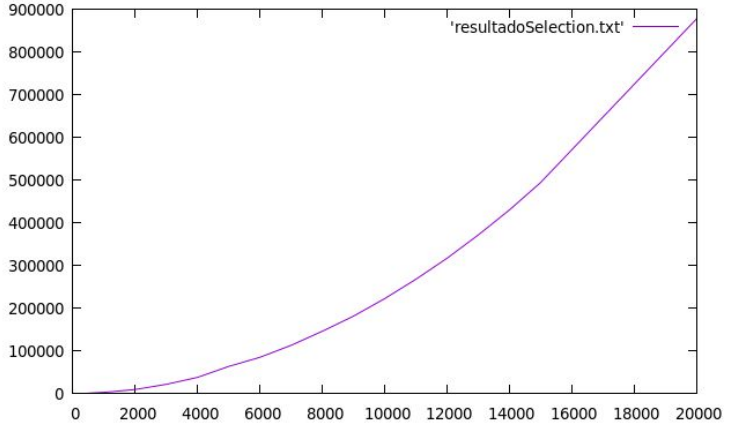

'resultadoInsertionRecursive.txt'
500000
450000
400000
350000
300000
250000
200000
150000
100000
50000
0
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
20000

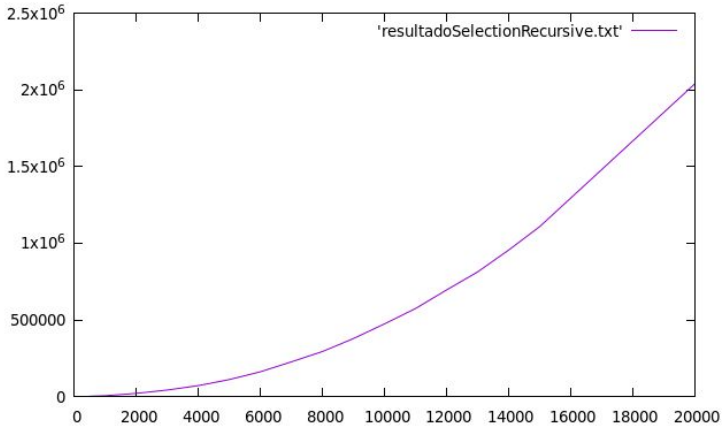

'resultadoMerge.txt'
7000
6000
5000
4000
3000
2000
1000
0
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
20000


'resultadoMergeIterative.txt'
7000
6000
5000
4000
3000
2000
1000
0
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
20000


'resultadoQuick.txt'
8000
7000
6000
5000
4000
3000
2000
1000
0
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
20000


'resultadoQuickIterative.txt'
8000
7000
6000
5000
4000
3000
2000
1000
0
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
20000


'resultadoSelection.txt'
900000
800000
700000
600000
500000
400000
300000
200000
100000
0
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
20000


'resultadoSelectionRecursive.txt'
2.5x10^6
2x10^6
1.5x10^6
1x10^6
500000
0
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
20000

Ordenação com lista duplamente encadeada

Bubblesort com lista

Inicializa dois ponteiros do tipo Lista, cada um recebendo o primeiro e o último elemento da lista respectivamente.

Um for com um ponteiro aux recebendo o primeiro elemento e vai avançando para a direita e outro for com um ponteiro aux2 recebendo o último elemento e vai caminhando para a esquerda.

Faz a comparação se o valor que está em aux2 é menor que o valor que está no elemento anterior do aux2

Insertionsort com lista

Assim como o bubblesort ele utiliza dois ponteiros que recebem o primeiro e o último elemento é mais um ponteiro chave que vai receber o valor do elemento que está na posição do ponteiro aux2.

Um for com o aux2 recebe o elemento que está após o primeiro e vai avançando para a direita,

Ponteiro aux recebe o elemento anterior ao o aux2. Agora será feita uma verificação se o valor de aux é menor que o valor de chave, caso for o que valor que está na no elemento posterior a aux vai receber o valor de aux. Ao terminar as verificações o valor que está no elemento posterior ao aux recebe o valor que está no chave, para ficar podendo fazer as inserções do valor no lugar correto.

Selectionsort com lista

Começa com dois ponteiros que recebem o primeiro e último elemento da lista. Um for com aux que recebe o primeiro elemento e percorre para a direita, um ponteio min recebe o aux.

Um segundo ofr com um aux2 recebe o elemento posterior de aux e avança para a direita, verifica de o valor de aux2 é menor que o valor de min, se for min recebe o aux2.

Ao finalizar o segundo for troca o valor de aux pelo valor de min, assim colocando os menores valores no começo da lista.

Conclusão

As tarefas foram divididas de forma que um implementasse com vetor e outro com lista duplamente encadeada, porém sentimos muita dificuldade em implementar com lista. Entretanto, Felipe Gabriel da Silva, conseguiu implementar quase todas com vetor, entrando com a exceção de heapsort interativo que foi desenvolvido, está compilando entretanto há uma falha de segmentação, porém o responsável não foi capaz de solucionar. Por outro lado Lista duplamente encadeada, ficou sobre responsabilidade de Ulisses Queiroz da Silva, que por sua vez foi capaz de implementar Bubblesort, Insertionsort e Selectionsort.

Agora falando sobre os resultados e aprendizados que este trabalho trouxe. Foi de grande ajuda para entender o desempenho destes algoritmos que até então estávamos acompanhando durante essas aulas, após obter o conhecimento que os algoritmos que utilizam de recursividade obtém uma grande vantagem sobre os outros é notório que o uso do Mergesort irá trazer grandes benefícios para futuros projetos, que necessitarão que seus dados sejam ordenados.

Devido ao fato de não termos conseguido implementar todos os algoritmos de ordenação implementados em lista duplamente encadeada, não foi possível concretamente avaliar qual implementação é mais eficiente.