Bruno Carvalho

Diogo Destefano

Pedro Bellotti

Rafael Terra

**Estrutura de Dados II**

Relatório Referente à Parte 1:

- Algoritmos de Ordenação

- Palavras do Momento

- Hashing

**UNIVERSIDADE FEDERAL DE JUIZ DE FORA**

24 de Setembro de 2017

**Índice**

1. Introdução
2. BubbleSort
3. Dados Brutos
4. Analise dos Resultados
5. Conclusão
6. Dados Sobre o Desenvolvimento

**1 – Introdução**

Desenvolvido em linguagem C++ e utilizando a IDE Visual Basic, a primeira parte do projeto tem como objetivo analisar e comparar os diferentes algoritmos de ordenação e funções de Hashing, a fim de se concluir quais métodos se apresentam mais eficientes em determinados contextos.

Variáveis como número de entradas, gasto em memória e tempo de execução foram determinantes para definir e comparar os resultados dentre os algoritmos testados.

O projeto foi desenvolvido visando consumir uma quantidade reduzida de memória, visto que por vezes, os algoritmos foram testados com 1.000.000 entradas. A cada iteração, o conjunto de dados foi *randomizado* a fim de se obter um resultado mais confiável (gasto de randomização: O(n)).

Já na segunda parte, foi utilizada a mesma linguagem e IDE da primeira parte. Neste projeto, os textos dos Tweet`s lidos da base de dados foram analisados e o número de palavras com tamanho maior ou igual a dois caracteres foram contabilizados utilizando funções de hash, para assim, obter-se a frequência de cada palavra contida nos textos.

**2 – BubbleSort**

O BubbleSort foi o algoritmo escolhido como o algoritmo não visto em sala, devido a sua popularidade e a sua facilidade para implementação, sendo um dos algoritmos de ordenação mais simples.

Uma forma de trabalhar com o algoritmo BubbleSort é comparando os elementos adjacentes (dois a dois), por exemplo: compara-se a primeira posição do vetor com a segunda, na segunda iteração (repetição), compara-se a segunda posição do vetor com a terceira, e assim sucessivamente. De acordo com o algoritmo, podemos ordenar o vetor de forma crescente ou decrescente.

O algoritmo BubbleSort percorre todo o vetor diversas vezes. No melhor caso, o algoritmo executa **n** operações relevantes, onde **n** representa o número de elementos do vetor. No pior caso, são feitas **n²** operações. A complexidade desse algoritmo é de ordem quadrática. Por isso, não é recomendado o uso dele para aplicações que requerem velocidade ou trabalhem com uma grande quantidade de dados.

Essa ordenação lembra como as bolhas num tanque de água que procuram seu próprio nível, e disso vem o nome do algoritmo, Bubble Sort.

**3 – Dados Brutos**

-Análise dos QuickSorts

-Tabelas

As tabelas a seguir foram criadas usando os dados retornados por cada tipo de QuickSort com N variando entre 1000, 5000, 10000, 50000, 100000, 500000 e 1000000.











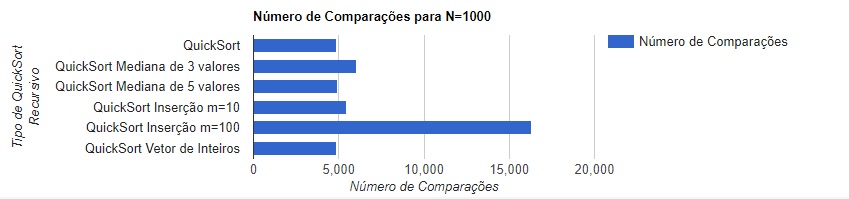


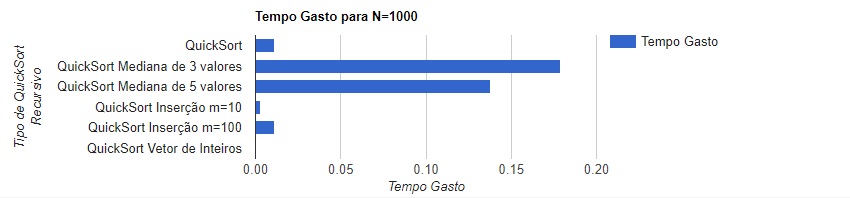


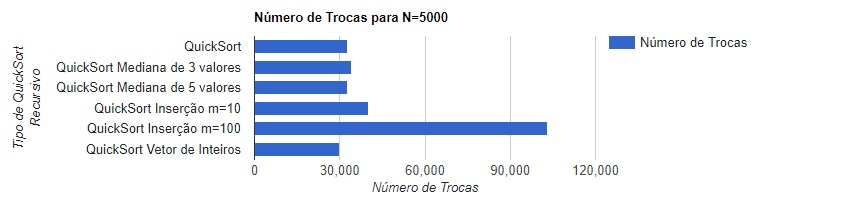
-Gráficos

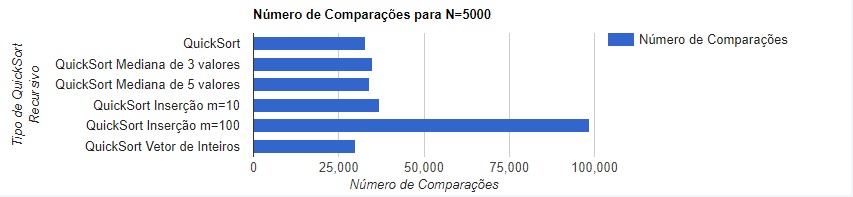
Os gráficos a seguir foram criados usando os dados retornados por cada tipo de QuickSort com N variando entre 1000, 5000, 10000, 50000, 100000, 500000 e 1000000.

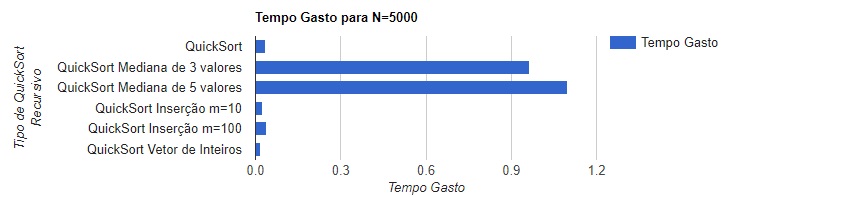


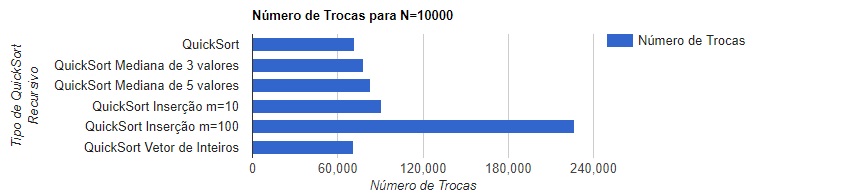




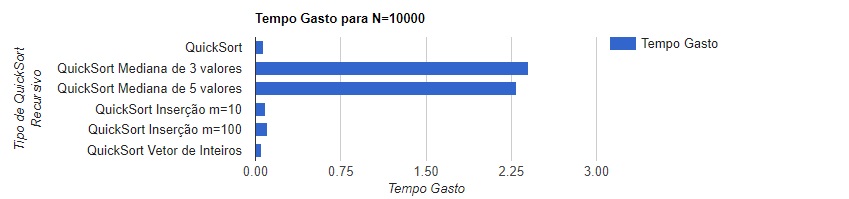




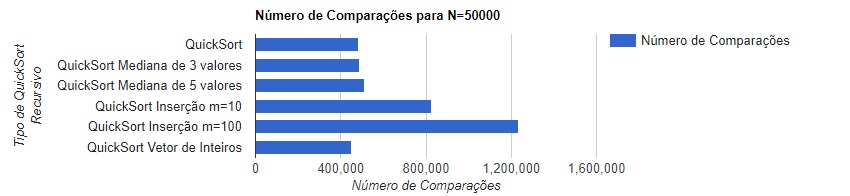




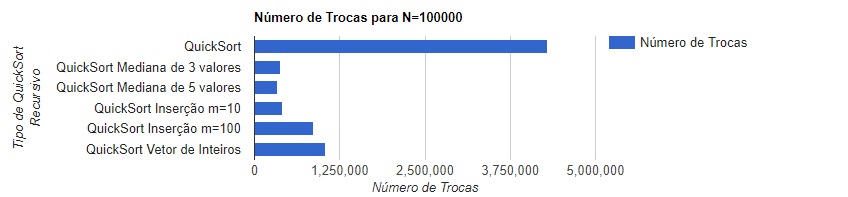


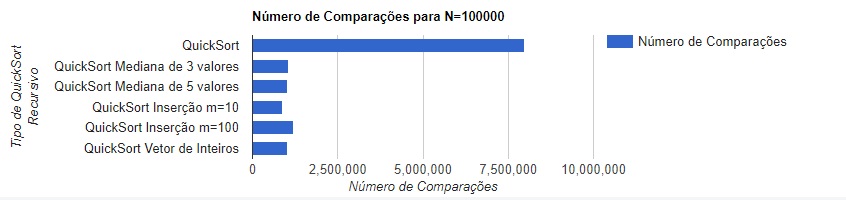






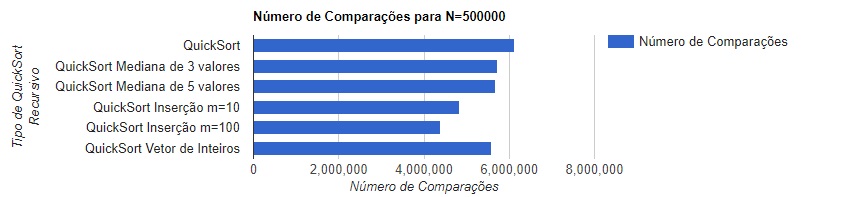


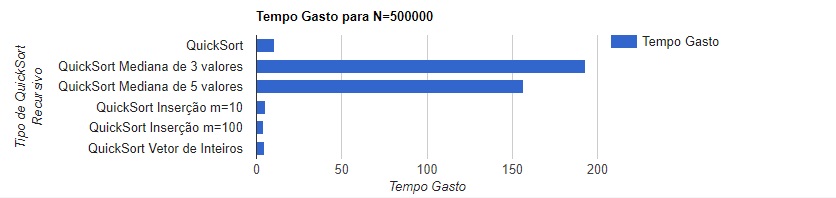


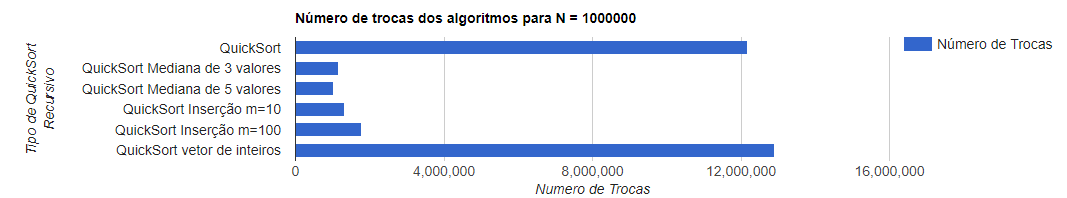


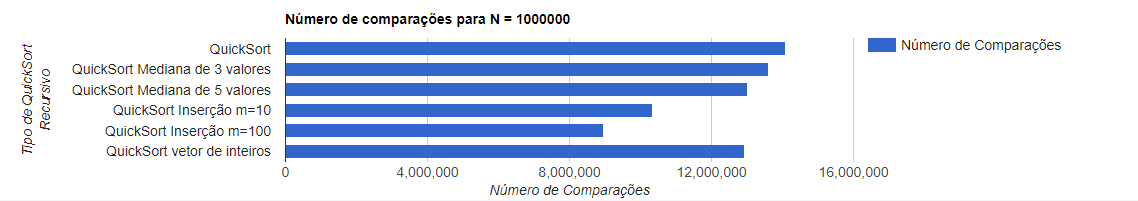














-Análise dos Algoritmos de Ordenação

-Tabelas

As tabelas a seguir foram criadas usando os dados retornados por cada algoritmo de ordenação com N variando entre 1000, 5000, 10000, 50000, 100000, 500000 e 1000000.

De acordo com o desempenho o QuickSort escolhido para comparação foi o Algoritmo QuickSort Recursivo com Insercao com m=10.

Os algoritmos InsertionSort e BubbleSort apresentam alguns dados iguais a 0 a partir de N=100000, isso ocorreu devido ao tamanho do inteiro retornado durante os testes, eles apresentavam valores maiores do que o aceitável pelo tipo de variável int.













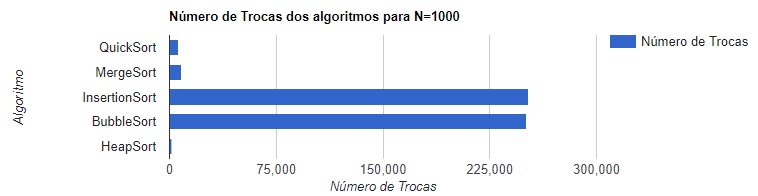


-Gráficos

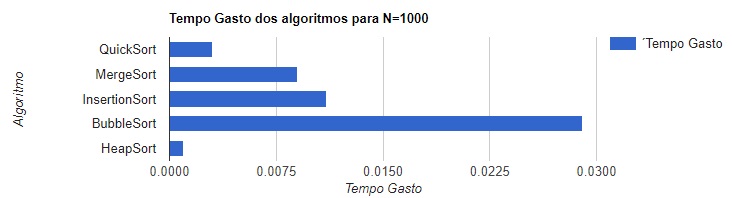
Os gráficos a seguir foram criados usando os dados retornados por cada algoritmo de ordenação com N variando entre 1000, 5000, 10000, 50000, 100000, 500000 e 1000000.

De acordo com o desempenho o QuickSort escolhido para comparação foi o Algoritmo QuickSort Recursivo com Insercao com m=10.

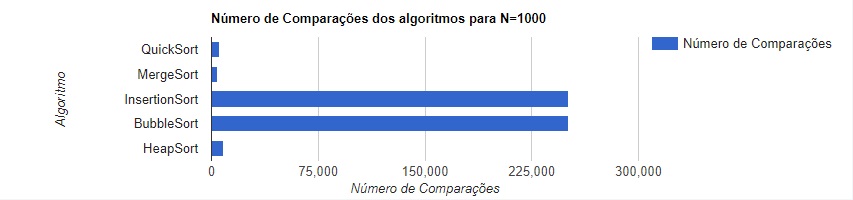
Os algoritmos InsertionSort e BubbleSort apresentam alguns dados iguais a 0 a partir de N=100000, isso ocorreu devido ao tamanho do inteiro retornado durante os testes, eles apresentavam valores maiores do que o aceitável pelo tipo de variável int.

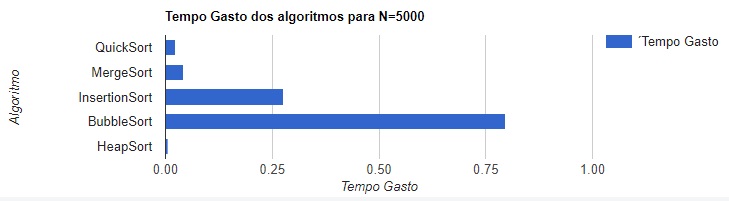


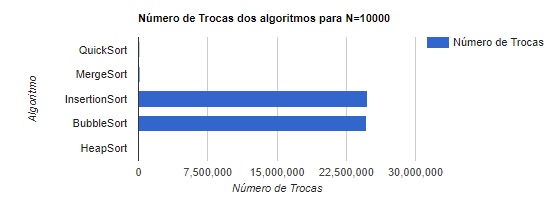


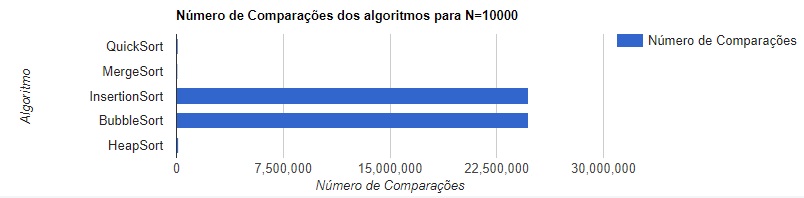


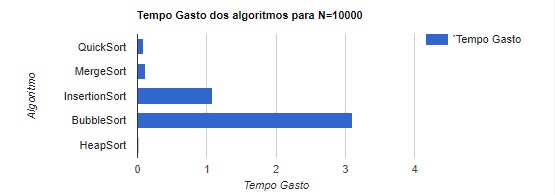
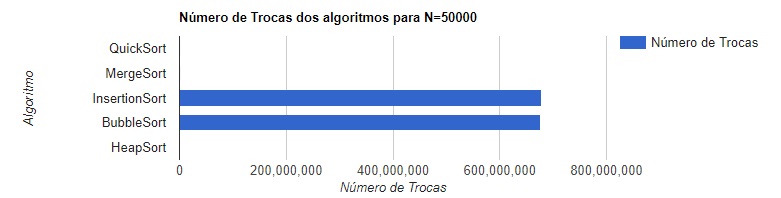


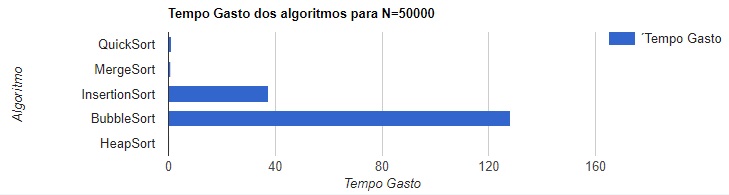
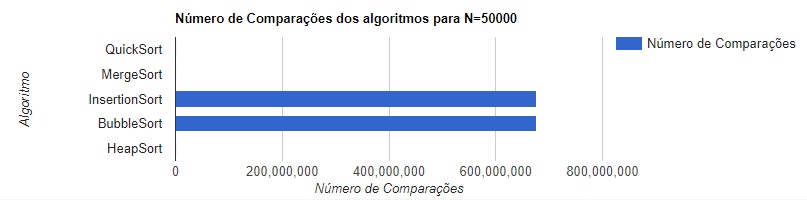


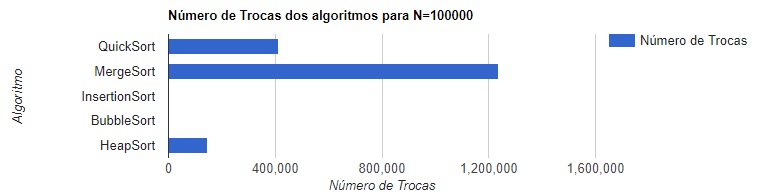


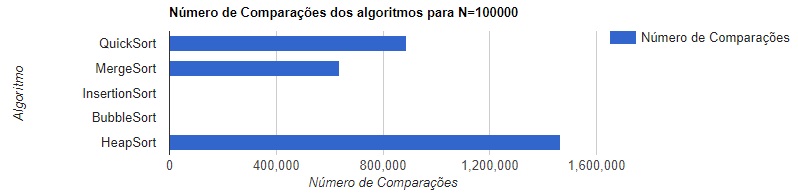


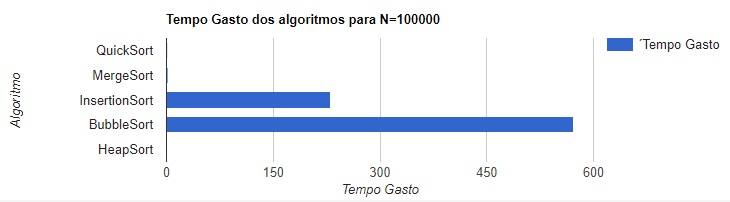




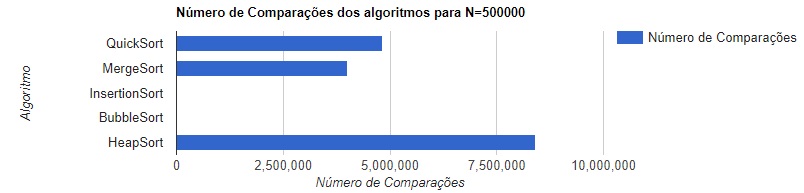


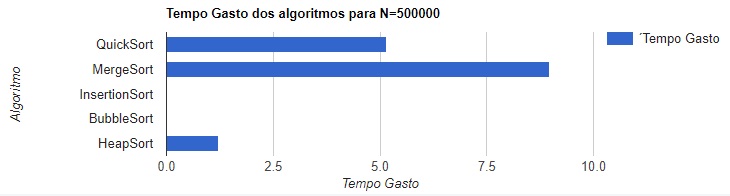
















**4 – Análises dos Resultados**

4.1. Cenário I

Os tipos de dados comparados foram o Vetor de Objetos do tipo *Tweet* e o vetor de Inteiros. Em termos de trocas e números de comparações, não houve diferença significativa de resultados. Porém, em termos de tempo de execução, a ordenação do Vetor de Inteiros se mostrou mais eficiente, visto que para uma amostra de 1.000.000 entradas o algoritmo gastou 11.039 segundos para finalizar sua execução, tempo significativamente melhor que os 30.161 segundos gastos pelo QuickSort do vetor de Objetos do tipo *Tweet*.

4.2. Cenário II

Dentre os diferentes Tipos de QuickSort, o melhor resultado geral obtido foi pelo Algoritmo de QuickSort de Inserção. Para Amostras menores (1.000 e 5.000) não houve diferença significativa entre os resultados, mas à medida em o tamanho da amostra aumentou, o Algoritmo de Inserção com sub-vetores de tamanho menor ou igual a 10 se mostrou mais eficiente. Quanto ao número