Nome: Beatriz Viana Costa NUSP: 13673214

Relatório EP3 – MAC0121 – Algoritmos e Estruturas de Dados:

1. Especificações de hardware e software que foram utilizados nos testes:

- CPU: 12th Gen Intel(R) Core(TM) i7-1255U;
- Sistema Operacional: Ubuntu 22.04.1 LTS;
- GCC: gcc version 11.3.0 (Ubuntu 11.3.0-1ubuntu1~22.04).

2. Introdução:

O objetivo do trabalho em questão é analisar empiricamente o desempenho dos seguintes algoritmos de ordenação:

- *Insertionsort;*
- *Mergesort;*
- *Quicksort;*
- *Heapsort.*

Para isso, foram realizados diversos testes com o uso de vetores testes que consistem em listas de no mínimo 250 palavras aleatórias com no máximo 10 letras cada uma. Estas palavras foram dispostas em ordem aleatória, parcialmente ordenada e totalmente ordenada.

Dessa forma, foi possível medir o tempo de execução de cada *sort*, o número de movimentações/troca de elementos dentro do vetor, e quantidade de comparações entre letras/palavras.

Assim, espera-se chegar a conclusões sobre os desempenhos de cada algoritmo.

3. Descrição dos métodos testados:

Os *sorts* testados foram os á mencionados anteriormente.

Para os algoritmos *insertionsort*, *mergesort* e *quicksort* foi implementada a biblioteca *bibliotecaSorts.h e bibliotecaSorts.c* que consiste em uma *struct* de lista de palavras, que armazena *n* palavras com tamanho máximo de 10 letras.

Já no *heapsort* foi utilizada a biblioteca *filaPrioridades.h,* implementada no arquivo *filaPrioridades.c,* onde se encontram as funções que criam a estrutura de dados utilizada neste *sort,* o *heap.*

Para a adaptação dos algoritmos de números inteiros para *strings*, no momento das comparações, para saber se uma palavra deve vir antes ou depois de outra, as letras de índice *x* em cada palavra são comparadas, caso sejam iguais o valor do índice é incrementado até que as letras sejam diferentes ou até que o índice chega ao tamanho máximo permitido para cada palavra (neste último caso as palavras seriam consideradas iguais).

Além disso foi também utilizada a biblioteca *time.h* para a cronometragem do tempo gasto por cada algoritmo de ordenação.

4. Descrição do tipo de instâncias:

Para a realização dos testes foram criados 27 vetores testes com palavras criadas de maneiras pseudoaleatória. Estes foram distribuídos da seguinte maneira:

- 9 vetores testes com palavras dispostas em ordem aleatória, iniciando com uma lista de 250 palavras, que dobra até 64.000 palavras;
- 9 vetores testes com palavras dispostas de forma parcialmente ordenada, ou seja, algumas palavras estão em suas posições corretas e outras não, iniciando com uma lista de 250 palavras, que dobra até 64.000 palavras;
- 9 vetores testes com palavras dispostas de forma ordenada, ou seja, todas as palavras estão em suas posições corretas, iniciando com uma lista de 250 palavras, que dobra até 64.000 palavras;

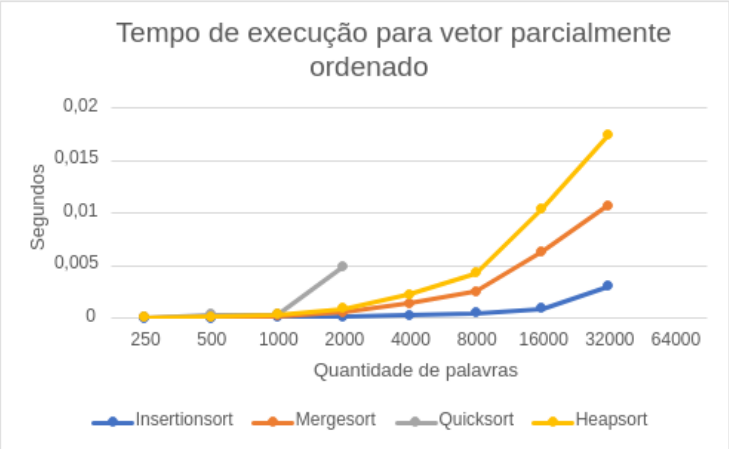
5. Resultados:

Nenhum dos algorimos conseguiram ordenar uma lista com 64.000 palavras, sendo que o *quicksort* no vetor aleatório ordenou até 8.000 palavras, e no parcialmente ordenado e totalmente ordenado conseguiu ir até 2.000 palavras antes de dar *segmentation fault.*

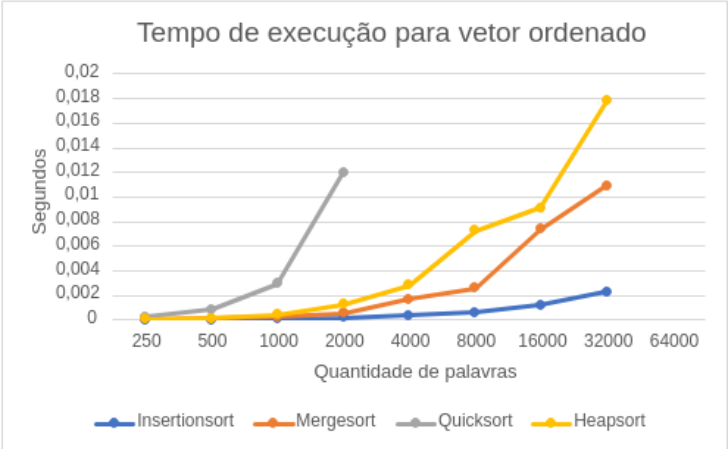
Dos testes realizados foram extraídos os seguintes gráficos:

- Tempo de execução de cada algoritmo em segundos:

Como esperado, no vetor com as palavras dispostas de forma aleatória, o *insertionsort* apresentou o maior tempo de execução, já os demais mantiveram tempos de execução muito próximos e abaixo de 1 segundo.

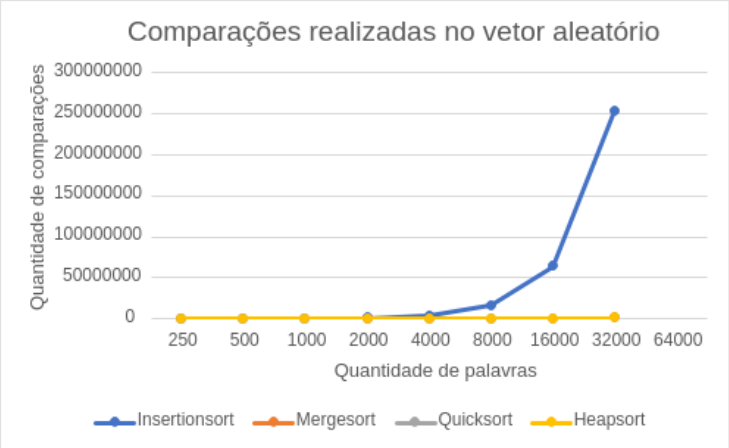


Para o vetor parcialmente ordenado todos os algoritmos apresentaram piora no tempo de execução, sendo que o *quicksort* foi o que teve maior aumento.

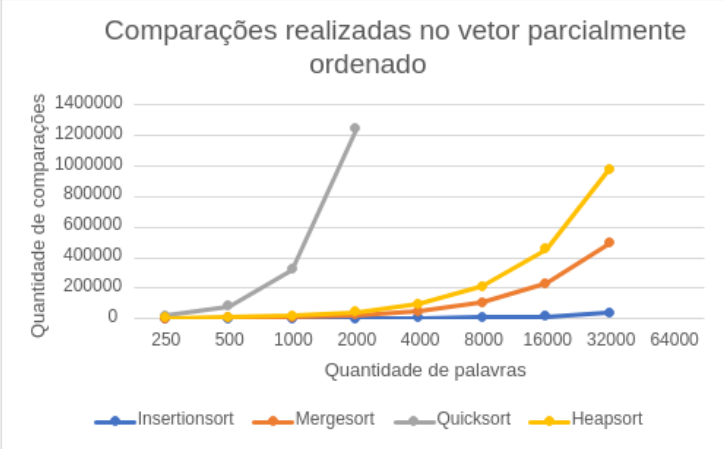


O mesmo pode ser observado neste teste, todos os algoritmos tiveram piora no tempo de execução em relação ao teste anterior, em especial o *quicksort.*

- Quantidade de comparações em cada algoritmo:

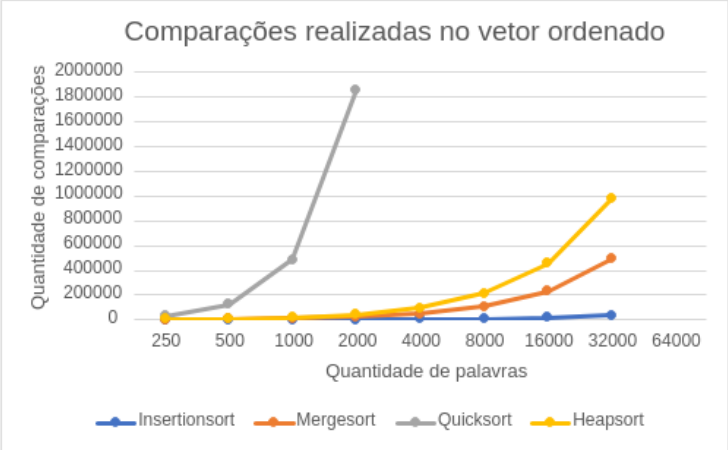


Foi possível observar que todos os algoritmos apresentaram número constante de comparações, exceto o *insertionsort*, que à medida que o tamanho da lista aumentava, a quantidade de comparações também aumentada.



Os algoritmos *mergesort*, *quicksort* e *heapsort* apresentaram piora, em especial o *quick*sort.

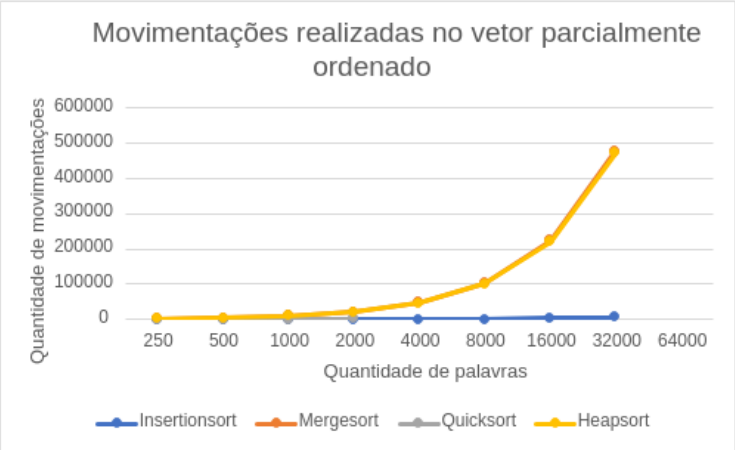
Contudo, o *insertionsort* apresentou grande melhora.



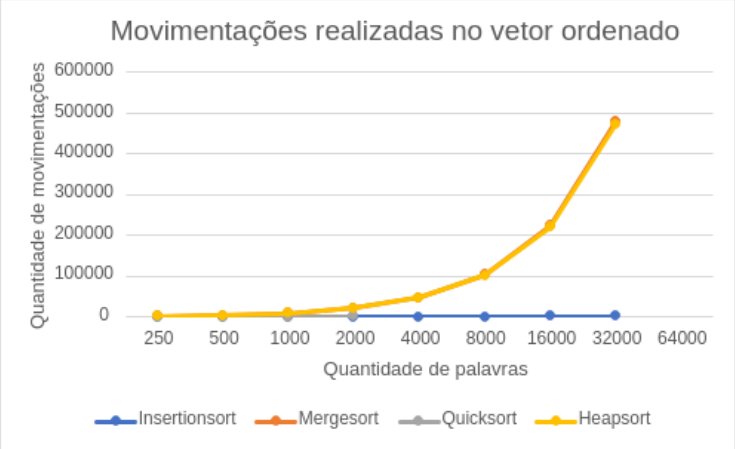
O *quicksort* apresentou grande piora, enquanto os outros algoritmos mantiveram uma média de comparações próximas dos valores observados no teste anterior, ou seja, houve certa estabilidade.

- Quantidade de movimentações em cada algoritmo:

Todos os algoritmos apresentaram bons resultados, com exceção do *insertionsort*, que à medida que o tamanho do vetor teste aumentava, a quantidade de movimentações também aumentava.



Já aqui as posições foram invertidas, o *insertionsort* apresentou poucas movimentações, o quicksort até 2.000 palavras



A quantidade de movimentações realizadas por cada código neste teste foram muitos próximas dos valores obtidos no teste anterior.

6. Conclusão:

Como esperado, pudemos observar que o *insertionsort* apresenta melhores resultados quando utilizado para ordenar listas que estão pelo menos parcialmente ordenadas e/ou listas que são pequenas. Isto acontece, pois, a cada palavra que é observada por este algoritmo, o insertion precisa olhas todas as outras palavras para saber onde encaixar a palavra atual, o que faz com que as comparações e movimentações aumentem de forma não estável a medida que a lista cresce e quando esta não está ordenada.

Já os demais *sorts* apresentaram melhores resultados nos 3 tipos de teste (tempo, comparações, movimentações) no vetor teste com palavras dispostas de maneira aleatória. Nos testes em que a lista está parcialmente ou totalmente ordenada, no caso do *merge* e *quicksort*, por exemplo, a quantidade de palavras que precisarão ser observadas (e logo comparadas) após *loop* diminuirá em apenas uma no pior caso (lista totalmente ordenada).