Universidade Federal do Espirito Santo

Departamento de Engenharia

Engenharia de Computação

Relatório

Aluno(a): Arthur Coelho Estevão e Milena da Silva Mantovanelli

Professora: Veruska Carretta Zamborlini

Vitória

2022

1 Introdução

Este trabalho tem como objetivo implementar um método de busca em textos. Para isso foram utilizados alguns métodos de ordenação tais como, system qsort, Bubblesort, Selectionsort, Insertionsort, Shellsort, Quicksort e Heapsort. O programa implementado é capaz de:

- Processar o texto (retira todas as quebras de linha e caracteres extras de espaço).
- Imprimir o Array de sufixos de forma ordenada.
- Imprimir o tempo gasto na ordenação de cada método de ordenação.
- Imprimir o número de comparações, trocas e atribuição de cada método de ordenação. (No qsort possui apenas o número de comparações).
- Imprimir uma query dentro de um contexto(ambos passados na linha de comando), ignorando a diferença entre letras maiúsculas e minúsculas.
- Ler uma query do teclado e imprimir suas ocorrências dentro de um contexto, ignorando a diferença entre letras maiúsculas e minúsculas.

2 Teoria

2.1 Métodos de Ordenação

Algoritmo de ordenação em ciência da computação é um algoritmo, de manipulação de dados, que coloca os elementos de uma dada sequência em uma certa ordem, em outras palavras, efetua sua ordenação completa ou parcial. As ordens mais usadas são a numérica e a lexicográfica. Existem várias razões para se ordenar uma sequência, uma delas é a possibilidade se acessar seus dados de modo mais eficiente.

2.2 Métodos de ordenação simples

Os métodos simples são adequados para pequenos vetores, são programas pequenos e fáceis de entender. Possuem complexidade C(n) = O(n²), ou seja, requerem O(n²) comparações. Exemplos: Insertion Sort, Selection Sort, Bubble Sort. Nos algoritmos de ordenação as medidas de complexidade relevantes são:

- Número de comparações C(n) entre chaves.
- Número de movimentações M(n) dos registros dos vetores.

Onde n é o número de registros.

2.2.1 Bubblesort

O bubble sort, ou ordenação por flutuação (literalmente ”por bolha”), é um algoritmo de ordenação dos mais simples. A ideia é percorrer o vector diversas vezes, e a cada passagem fazer flutuar para o topo o maior elemento da sequência.

No melhor caso, o algoritmo executa **n** operações relevantes, onde **n** representa o número de elementos do vector. No pior caso, são feitas n^2 operações. A complexidade desse algoritmo é de ordem quadrática. Por isso, ele não é recomendado para programas que precisem de velocidade e operem com quantidade elevada de dados.

2.2.2 Selectionsort

A ordenação por seleção, baseado em se passar sempre o menor valor do vetor para a primeira posição (ou o maior dependendo da ordem requerida), depois o de segundo menor valor para a segunda posição, e assim é feito sucessivamente com os $n-1$ elementos restantes, até os últimos dois elementos.

Para todos os casos (melhor, médio e pior caso) possui complexidade C(n) = O(n²) e não é um algoritmo estável.

2.2.3 Insertionsort

Insertion Sort, ou ordenação por inserção, é um algoritmo de ordenação que, dado uma estrutura (array, lista) constrói uma matriz final com um elemento de cada vez, uma inserção por vez. Assim como algoritmos de ordenação quadrática, é bastante eficiente para problemas com pequenas entradas, sendo o mais eficiente entre os algoritmos desta ordem de classificação.

É o método que percorre um vetor de elementos da esquerda para a direita e à medida que avança vai ordenando os elementos à esquerda. Possui complexidade C(n) = O(n) no melhor caso e C(n) = O(n²) no caso médio e pior caso. É considerado um método de ordenação estável.

2.3 Métodos de ordenação eficientes

Os métodos eficientes são mais complexos nos detalhes, requerem um número menor de comparações. São projetados para trabalhar com uma quantidade maior de dados e possuem complexidade C(n) = O(n log n). Exemplos: Quick sort, Merge sort, Shell sort, Heap sort, Radix sort, Gnome sort, Count sort, Bucket sort, Cocktail sort, Timsort.

2.3.1 Shellsort

Shell sort é o mais eficiente algoritmo de classificação dentre os de complexidade quadrática. É um refinamento do método de inserção direta.O algoritmo difere do método de inserção direta pelo fato de no lugar de considerar o array a ser ordenado como um único segmento, ele considera vários segmentos sendo aplicado o método de inserção direta em cada um deles.Basicamente o algoritmo passa várias vezes pela lista dividindo o grupo maior em menores. Nos grupos menores é aplicado o método da ordenação por inserção, possui complexidade C(n) = O($n \log_2 n$) no melhor e pior caso e para coso médio depende da sequência do gap.

2.3.2 Quicksort

O algoritmo quicksort é um método de ordenação muito rápido e eficiente, sendo um algoritmo de ordenação por comparação não-estável e adota a estratégia de divisão e conquista. A estratégia consiste em rearranjar as chaves de modo que as chaves ”menores”precedam as chaves ”maiores”. Em seguida o quicksort ordena as duas sublistas de chaves menores e maiores recursivamente até que a lista completa se encontre ordenada, possui complexidade C(n) = O(n²) no pior caso e C(n) = O(n log n) no melhor e médio caso.

2.3.3 Heapsort

O algoritmo heapsort é um algoritmo de ordenação generalista, tem um desempenho em tempo de execução muito bom em conjuntos ordenados aleatoriamente, com um uso de memória bem

comportado e o seu desempenho em pior cenário é praticamente igual ao desempenho em cenário médio. O heapsort trabalha no lugar e o tempo de execução em pior cenário para ordenar n elementos é de O($n \log n$). Utiliza uma estrutura de dados chamada heap, para ordenar os elementos à medida que os insere na estrutura. Assim, ao final das inserções, os elementos podem ser sucessivamente removidos da raiz da heap, na ordem desejada, podendo ser representada como uma árvore (uma árvore binária com propriedades especiais) ou como um vetor. Para uma ordenação decrescente.

2.4 Função Qsort

A função qsort() nada mais é do que a implementação do QuickSort contida na biblioteca padrão da linguagem C. Ela permite ordenar qualquer tipo de vetor (int, float, string, struct, etc). Para tanto, basta definir a maneira como os elementos do vetor devem ser comparados.

2.5 Busca em Memória Principal

2.5.1 Array de Sufixos

Um array de sufixos é um array ordenado de todos os sufixos de uma cadeia de caracteres. É uma estrutura de dados simples, porém poderosa que é usada, dentre outras, em índices de textos inteiros, algoritmos de compressão de dados e dentro do campo da bioinformática.

Seja S= s_1,s_2,...,s_n uma cadeia de caracteres e seja S[i,j] a subcadeia de S que vai de i até j. O array de sufixos A de S é definido como sendo um array de inteiros que proveem as posições iniciais dos sufixos de S em ordem lexicográfica. Isto significa que uma entrada A[i] contem a posição inicial do i-ésimo menor sufixo em S e, da mesma forma, para todo $1 < i \leq$ n: S[A[i-1],n]$<$S[A[i],n].

3 Resultados

Foram feitas pequenas melhorias no código para que o usuário tenha a possibilidade de decidir o método ao qual deseja adquirir a ordenação do texto, possibilitando também a escolha de imprimir na tela as estatísticas do método de ordenação escolhido, tais estatísticas englobam quantidade de comparações, atribuições e trocas executados pelo mesmo, juntamente com o tempo de sua execução. Outra melhoria executada foi a implementação de ignorar a diferença entre maiúsculas e minúsculas na busca em memoria.

Foi elaborada uma planilha anexada a planilha do ultimo relatório na página 2 com os dados de saída, disponibilizamos também o acesso ao github com o código e algoritmos analizados.

Planilha : `www.encurtador.com.br/fiFKV`

Github : `www.encurtador.com.br/asADQ`.

Temos demonstrados nas Fig.1,2 e 3 a representação gráfica e comparativa referente a quantidade de comparações, trocas e atribuições executadas pelos algoritmos de ordenação mencionados anteriormente, buscando ordenar o array de suffix, com tudo pode-se observar que algoritmos de ordenação simples como bubble,selection e insertion por possuírem uma complexidade de algoritmo elevada, ficam incapacitados de ordenar texto extensos como por exemplo talex.txt e moby.txt, algo a destacar, pelo fato do qsort ser um função da biblioteca stdlib.h não foi possível retornar a quantidade de trocas ou atribuições feitas por ele dentro da função, somente a quantidade de comparações, executada dentro do método de com comparação.

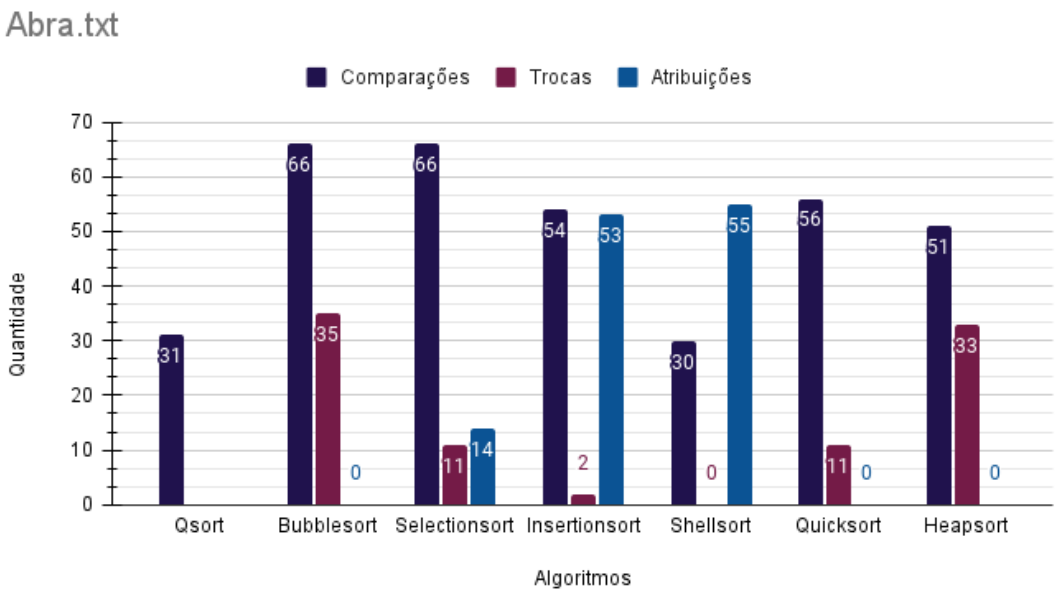


Figura 1: Analise gráfica do desempenho dos algoritmos para a base de dados abra.txt

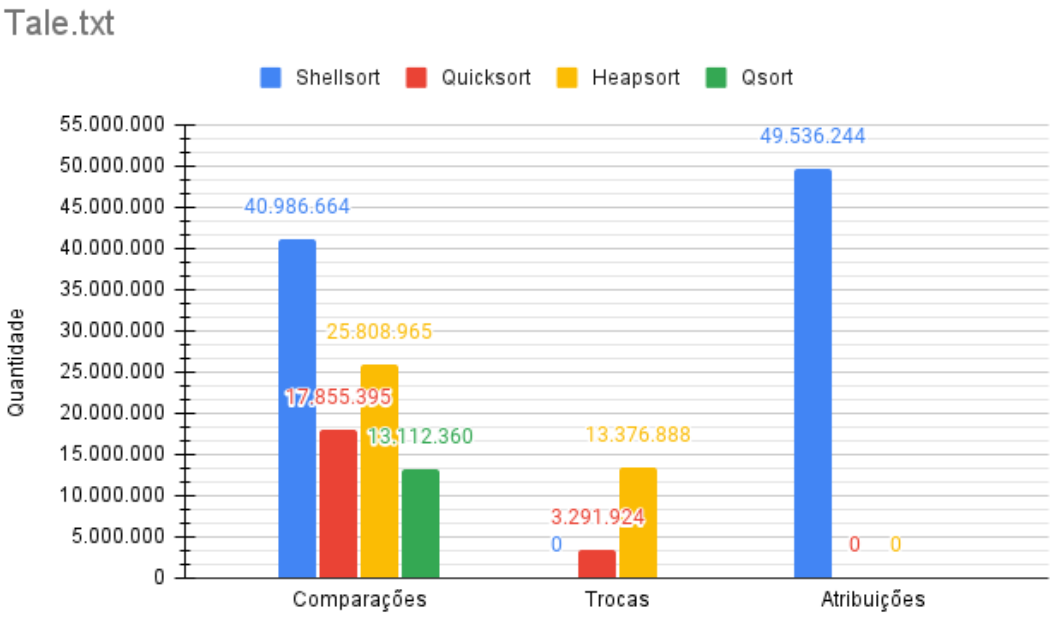


Figura 2: Analise gráfica do desempenho dos algoritmos para a base de dados tale.txt

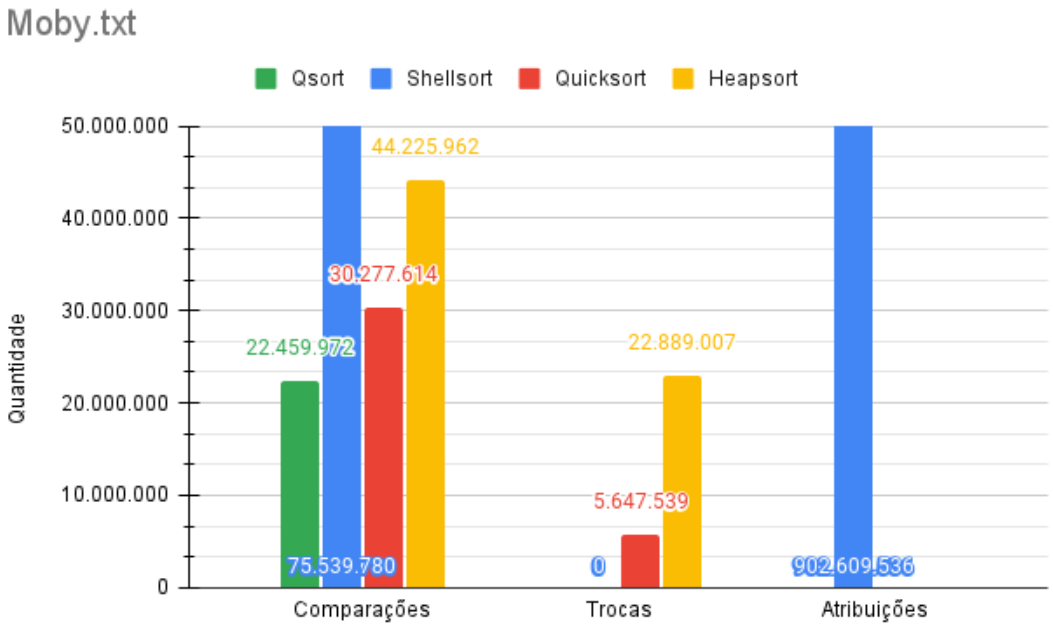
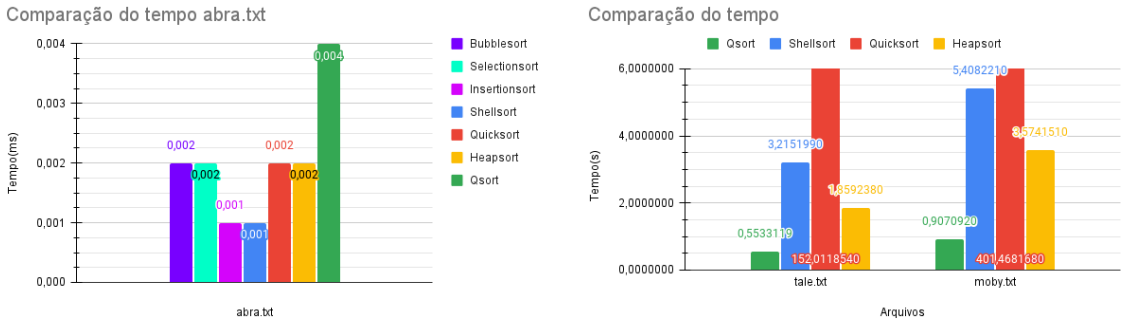


Figura 3: Análise gráfica do desempenho dos algoritmos para a base de dados moby.txt

Dentre todos os algoritmos analisado o Quick sort se apresentou o algorítimo mas eficiente em relação a quantidade de comparações,trocas e atribuições executadas. No desenvolvimento do código afim de comparação desenvolvemos a implementação do Quicksort para posteriormente compararmos com a função Qsort, com isso concluímos que o qsort method

devido suas abstrações de uma maneira estável e transitiva possui um resultado melhor que a aplicação da Propriá função Quicksort, podendo ser melhor observado suas diferença de execução nas figuras 4a e 4b onde temos o tempo que cada algoritmo leva para ordenar o vetor de array.



(a) Dados abra.txt

(b) Dados tale.txt e moby.txt

Figura 4: Comparação do tempo gasto

4 Conclusão

Podemos concluir que a função de sistema qsort foi o método de ordenação com maior eficiência quando o array de sufixos é grande, pois apresenta o menor tempo gasto na ordenação do array, sendo quase quatro vezes mais rápido do que o segundo menor tempo (Heapsort). Entretanto, quando o array de sufixos é pequeno o qsort, gastou mais tempo que todos os outros métodos implementados, apesar de realizar menos comparações.

5 Referências

https://pt.wikipedia.org/wiki/Algoritmo$_{d}e_{o}rdena$ção

https://www.treinaweb.com.br/blog/conheca-os-principais-algoritmos-de-ordenacao https://pt.wikipedia.org/wiki/Bub

https://pt.wikipedia.org/wiki/Selection$_{s}ort$

https://pt.wikipedia.org/wiki/Insertion$_{s}ort$

https://pt.wikipedia.org/wiki/Shell$_{s}ort$

https://pt.wikipedia.org/wiki/Quicksort

https://pt.wikipedia.org/wiki/Heapsort

http://www.faccamp.br/osvaldo/SequenciasConjuntos.pdf

https://pt.wikipedia.org/wiki/Arranjo$_{d}e_{s}ufixos$