Algoritmos e Estruturas de Dados I

Professor: André Carlos Ponce de Carvalho

**Algoritmos de Ordenação**

Análise assintótica, de memória, e estatística de algoritmos de ordenação (Bubble, Selection, Quick e Heap Sort)

Aluno: Felipe Barbosa de Oliveira

Nº USP: 10276928

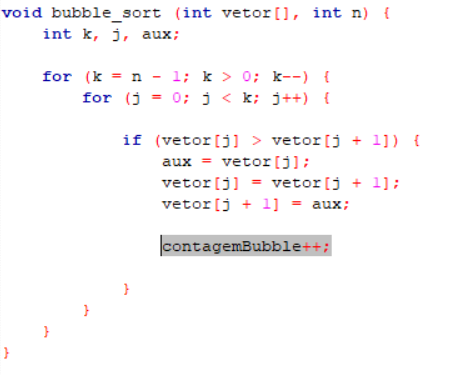
**INTRODUÇÃO**

Neste trabalho, foi analisada a complexidade de quatro algoritmos de ordenação de vetores: o Quick Sort, o Heap Sort, o Bubble Sort e o Selection Sort. Também foi discutido o uso de memória e o funcionamento de cada um deles. Além disso, os algoritmos foram testados usando vetores desordenados de tamanho 10 até 100000, utilizando de um contador para se ter uma noção da sua curva de complexidade.

**ANÁLISE ASSINTOTICA**

Bubble Sort:

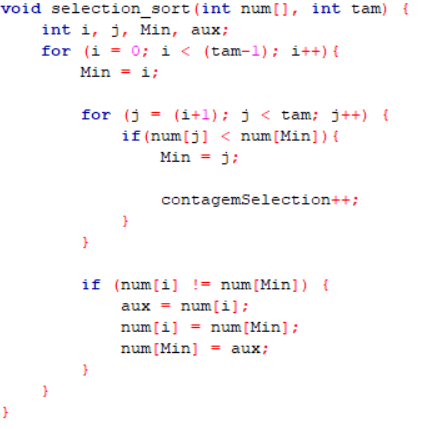
O algoritmo com o contador foi implementado da seguinte maneira, utilizando a mesma lógica de posicionamento do contador dos trabalhos anteriores:



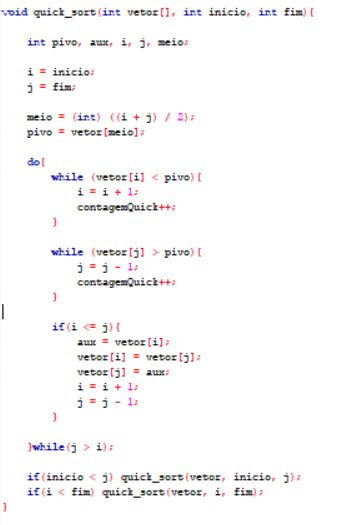
Vemos que o algoritmo inteiro se resume a um procedimento dentro de duas estruturas de repetição, sendo que cada uma repete um número linear a de vezes, sendo o tamanho do vetor. Portanto, podemos facilmente ver que o algoritmo é sempre .

Selection Sort:

O algoritmo foi implementado do seguinte modo:



Vemos uma estrutura muito parecida com o Bubble Sort, com uma estrutura de repetição dentro de outra, com ambas lineares a . Porém desta vez, possui uma condição apenas dentro da estrutura de repetição de fora, que repete vezes. Portanto, podemos logicamente afirmar que o algoritmo é sempre , que pode ser simplificado a .



Quick Sort:

O algoritmo com o contador se encontra ao lado. Por se tratar de um algoritmo recursivo, utilizaremos o método de árvore de recorrência para resolver os casos. Temos que a função da equação de recorrência será , pois se trata basicamente de um while principal dependente de com dois whiles dentro dele, porém sendo muito instáveis dependendo do vetor, e de poucas repetições. O pior caso se dá quando o pivô é o menor ou o maior número do vetor, criando duas separações muito desbalanceadas. Deste modo, a recursividade chamará um vetor de tamanho . Então temos:

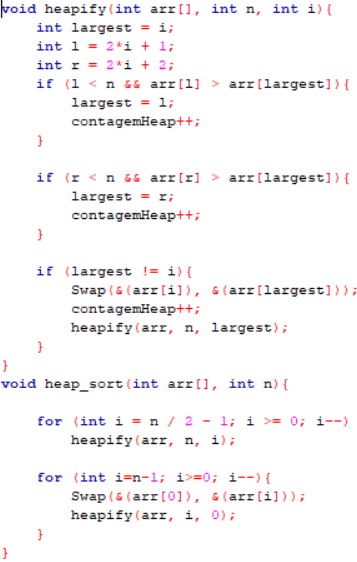
Que resolvendo pelo método de árvore (recorrência já resolvida no trabalho anterior), resultamos em .

Já no melhor caso, temos que o pivô separa o vetor em dois vetores de tamanhos iguais (ou diferentes apenas de 1 caso seja ímpar). Assim, temos:

Que, também resolvendo pelo método de árvore (recorrência também resolvida no trabalho anterior), resultamos em .

Heap Sort:

Foi implementado da seguinte maneira:



**RESULTADOS E DISCUSSÕES**

Bubble Sort:

Os valores encontrados nos testes foram:

E com isso, foram feitas as seguintes estatísticas:



Vemos facilmente que o número de contagens cresce de maneira exponencial, o que está de acordo com o fato do algoritmo ser . Vemos também que se trata de um algoritmo não muito estável, com grandes variações em menores tamanhos de vetor.

Selection Sort:



Novamente, podemos ver o tempo cresce de maneira exponencial, porém com valores muito menores que do Bubble Sort. O algoritmo é bem mais estável e com crescimento exponencial bem pequeno.

Quick Sort:



No caso do Quick Sort, vemos que ele cresce de maneira ainda mais suave que o Selection Sort, o que faz sentido contado o fato de ser . A variância de resultados é relativamente grande, principalmente para valores menores, e se mostra um algoritmo muito mais rápido e eficiente que os dois anteriores (contando que o número de contadores neste é o dobro que nos anteriores).

Heap Sort:





O algoritmo mostrou-se menos eficiente que o Quick Sort (mesmo considerando o fato de ter um contador a mais). O crescimento é maior que o do Quick Sort, porém mais suave que os dois primeiros (também sendo coerentes com o fato de ser da mesma complexidade do Quick Sort).

**MEMÓRIA**

Podemos facilmente inferir que tanto o Bubble Sort quanto o Selection Sort são os menores consumidores de memória dentre os quatro algoritmos. Isso se dá pelo fato de serem os únicos iterativos, não criando novas variáveis a cada repetição. Já o Quick Sort e o Heap Sort, por serem recursivos, irão alocar mais memória pois irão criar novas variáveis a cada recorrência. Portanto a complexidade espacial do Bubble Sort e do Selection Sort teóricamente é sempre a mesma, não dependendo do tamanho do vetor. Já nos outros dois, a complexidade dependerá de quantas recorrências ocorrerem, ou seja, depende diretamente do tamanho do vetor e de seu grau de organização.

**INVERSÃO DOS ALGORITMOS**

Os algoritmos na sua forma inversa (ordenando de maneira decrescente) foram implementados no código junto com este relatório.

**CONCLUSÕES**

Com isso, concluímos que tanto o Bubble como o Selection Sort são os dois algoritmos mais lentos em quase todos os casos (principalmente o Bubble Sort), porém, consomem consideravelmente menos memória, podendo então ser viáveis em situações onde o tempo é menos importante que o espaço. Já o Quick e o Heap Sort se tratam do contrário. Por serem recursivos, alocam muito mais memória que os outros dois, porém, mostraram-se muito mais rápidos e de complexidade menor. O algoritmo mais rápido dentre estes, foi o Quick Sort.