Relatório sobre a comparação dos tempos de execução dos algoritmos InsertionSort, MergeSort, Heapsort e QuickSort

Os algoritmos de ordenação para termos de comparação foram executados numa mesma máquina com especificações: processamento 2.50Ghz, Memória RAM 8.00 GB, para que com isso, recursos de hardware diferentes não viessem a influenciar os resultados nas comparações dos tempos de execução.

O algoritmo Insertion Sort, funciona de forma, semelhante a classificação de cartas em um jogo no qual se tem uma carta, e que para se inserir essa carta no leque deve-se mover todo o leque para esquerda ou direita, depende da situação de ordenação, para que com isso seja possivel abrir uma posição, para entrada daquela carta no leque.

A complexidade deste algoritmo está estritamente relacionada com a entrada não ordenada dos elementos

- **Melhor Caso:** ocorre quando o vetor já se encontra na forma de ordenação desejada ficando neste caso com complexidade de tempo de execução linear O(n), ficando neste assim mais eficiente que os algoritmos de Quick, Merge e Heap.
- **Pior Caso**: ocorre, por exemplo, quando o vetor está ordenado em crescente e deve-se, ordená-lo na decrescente, ou seja, inverte-lo, ficando neste caso com tempo de execução O(n^2) .

O algoritmo MergeSort por sua vez, parte do pressuposto da divisão e conquista, no qual se tem um grande problema a ser resolvido, e este acaba sendo sub-dividido em pequenos problema mas simples de serem resolvidos.

A intercalação de dois vetores ordenados ocorre de forma superficial da seguinte forma:

> Uma variável em cada vetor indica o próximo elemento a ser inserido a lista intercalada;

> Enquanto ambas os vetores tiverem elementos, coloque o menor entre os dois elemento

indicados no vetor intercalado e incremente o índice respectivo;

> Quando um dos dois vetores não tiver mais elementos, concatene o outro no final do vetor intercalado.

- **Melhor Caso:** Tem complexidade de O(N log N);
- **Pior Caso:** Tem complexidade de O(n).

O algoritmo HeapSort utiliza uma estrutura denominada Heap, esta tem uma semelhança muito grande com uma árvore binária, é semelhante ao tipo de seleção, onde primeiro encontramos o elemento máximo e colocamos o elemento máximo no final.

Algoritmo de ordenação na crescente com o uso do HeapSort

1. Produz o heal a partir do vetor de entrada;

2. Nesta etapa, o maior elemento do vetor é armazenado da raiz da "árvore". Substitua-o pelo último elemento do heap seguido por reduzir o tamanho do heap em 1. Finalmente, heapify a raiz da árvore;

3. Repita as etapas acima enquanto o tamanho da pilha for maior que 1.

- **Melhor Caso:** Tem complexidade também de O(N log N), ou seja, seu médio caso também é O(N log N);
- **Pior Caso:** Tem complexidade de O(N log N), este caso acontece quando se está exatamente na ordem inversa da desejada.

O heap é mais rápido que o quick, quando se tem muitos valores repetidos.

O algoritmo Quick Sort também parte do pressuposto da divisão e conquista, falado acima. passo básicos da construção deste algoritmo:

> Escolhe-se um pivô;

> O pivô é posicionado de forma que todos os elementos anteriores a ele sejam menores e todos os posteriores,maiores

> Ordena recursivamente os subvetores à esquerda e à direita

O pivô neste algoritmo é o elemento central da ordenação, a sua escolha implica correta ou errada, resulta respectivamente, no melhor e pior caso da ordenação

- Melhor Caso: o melhor caso ocorre quando o processo de partição sempre escolhe o elemento do meio como pivô com complexidade O(N log N).
- Pior Caso: ocorre quando o processo de partição sempre escolhe o elemento maior ou menor como pivô. Na qual o último elemento é sempre escolhido como pivô, o pior caso ocorrerá quando o vetor já estiver classificada em ordem crescente ou decrescente com complexidade O(n^2). Uma forma de se evitar isso é utilizando da aleatoriedade na escolha do pivô.

Comparações dos tempos execução

Observação: Todos os algoritmos de ordenação estão configurados para ordenar na ordem crescente.
Complexidade de tempo com base na implementação dos algoritmos, ou seja, **tempo final da execução - tempo inicial da execução.**

1 - vetor: tem como entrada um vetor de 1000 posições ordenando na crescente

[1]-Insertion: 57300 < [2]-Heapsort: 504000 < [3]-Mergesort: 797400 < [4]-Quicksort: 4740400

2- Vetor: tem como entrada um vetor de 1000 posições ordenando na decrescente

[1]-Heapsort: 283200 < [2]-Mergesort: 295700 < [3]-Quicksort: 3935500 < [4]-InsertionSort: 5739800

3 - vetor: tem como entrada um vetor de 1000 posições preenchido de forma randômica

[1]-Quicksort: 209701 < [2]-Heapsort: 372900 < [3]-Mergesort: 500000 < [4]-InsertionSort: 855200

4 - vetor: tem como entrada um vetor de 1000 posições preenchido com um mesmo elemento em cada posição

[1]-InsertionSort: 10300 < [2]-Heapsort: 78300 < [3]-Mergesort: 357700 < [4]-Quicksort: 2386100

Portanto conclui-se que, os resultados obtidos na implementação foram os esperados, por conta dos melhores e piores casos de cada algoritmo, demonstrado assim que cada algoritmo vai ser mais eficiente que o outro em alguns casos já em outros não.