**1 Introdução**

Problemas são questões propostas em busca de uma solução. Com o propósito de conceder uma solução para certo problema, existem os algoritmos, cada problema que é decidível possui um algoritmo que determina uma solução para cada instância desse problema.

Um algoritmo é uma sequência de passos utilizados para resolver determinados problemas. Em certas situações um problema que já possui um algoritmo como solução, precisa ser resolvido com uma velocidade de processamento maior que a atual, ou seja, precisaremos de um outro algoritmo para resolver o mesmo problema, mas de forma diferente.

Um tipo de algoritmo muito usado na resolução de problemas computacionais são os algoritmos de ordenação, que servem para ordenar/organizar uma lista de números ou palavras de acordo com a sua necessidade. As linguagens de programação já possuem métodos de ordenação, mas é bom saber como funcionam os algoritmos, pois há casos de problemas em que o algoritmo de ordenação genérico não resolve, às vezes é necessário modificá-lo.

Neste artigo serão abordados três algoritmos de ordenação, que são eles: Heap Sort, Merge Sort, Counting Sort. Apontando suas definições, os aspectos históricos, a ordem de complexidade, comparação de funcionamento dos algoritmos e por fim os resultados obtidos na execução destes algoritmos numa mesma plataforma.

Este artigo está estruturado da seguinte maneira: na seção 2, estão sendo definidos cada um dos algoritmos de ordenação com seus respectivos aspectos históricos. Na Seção 3 é apresentada a ordem de complexidade de cada algoritmo. N seção 4 é apresentado a comparação de funcionamento dos algoritmos. A seção 5 apresenta os resultados obtidos na execução dos mesmos. Encerramos nosso trabalho na seção 6, onde apresentamos a conclusão obtida diante dos resultados. E por fim, na seção 7 está nossa referência bibliográfica.

**2 Definição dos algoritmos**

**2.1 Heap Sort**

O algoritmo heapsort é um algoritmo de ordenação generalista, e faz parte da família de algoritmos de ordenação por seleção. Foi desenvolvido em 1964 por Robert W. Floyd e J.W.J Williams. Uma versão inicial do HeapSort foi publicada por Robert W. Floyd em 1962 sob o nome Treesort. Dois anos mais tarde, J.W.J. Williams publicou uma versão melhorada sob o nome HeapSort. E em dezembro de 1964, Robert W. Floyd publicou uma versão final com o nome Treesort 3.

Tem um desempenho em tempo de execução muito bom em conjuntos ordenados aleatoriamente, tem um uso de memória bem comportado e o seu desempenho em pior cenário é praticamente igual ao desempenho em cenário médio.

O heapsort não é um algoritmo de ordenação estável. Porém, é possível adaptar a estrutura a ser ordenada de forma a tornar a ordenação estável. Cada elemento da estrutura adaptada deve ficar no formato de um par (elemento original, índice original). Assim, caso dois elementos sejam iguais, o desempate ocorrerá pelo índice na estrutura original.

**2.2 Merge Sort**

O Merge Sort, ou ordenação por mistura, é um exemplo de algoritmo de ordenação por comparação do tipo dividir-para-conquistar. O algoritmo foi proposto por John Von Neumann em 1945. A atribuição a ele veio de Knuth, que argumentou no seu livro ‘Arte de Programação Computacional: Ordenando e Procurando’ que Von Neumann foi o primeiro a descrever a ideia. Como o algoritmo *Merge Sort* usa a recursividade, há um alto consumo de memória e tempo de execução, tornando esta técnica não muito eficiente em alguns problemas.

É possível implementar o *merge sort* utilizando somente um vetor auxiliar ao longo de toda a execução. Ele é um algoritmo estável na maioria das implementações, em que elas podem ser iterativas ou recursivas. É possível também implementar o algoritmo com espaço adicional.

Tem como desvantagens a utilização de funções recursivas e o gasto extra de memória. O algoritmo cria uma cópia do vetor para cada nível da chamada recursiva, totalizando um uso adicional de memória.

**2.3 Counting Sort**

Counting Sort é um algoritmo de ordenação estável. As chaves podem tomar valores entre 0 e M-1. Se existirem k0 chaves com valor 0, então ocupam as primeiras k0 posições do vetor final: de 0 a k0-1.

Tem como uma de suas características a necessidade de se ter memória auxiliar. Logo, não é in-place.

**3 Ordem de complexidade**

**3.1 Heap Sort**

Comparações no pior caso: 2n log2n + O(n) é o mesmo que 2n lgn + O(n)

Trocas no pior caso: n log2n + O(n) é o mesmo que n lgn + O(n)

Pior e médio caso: O(n log2n) é o mesmo que O(n lgn)

**3.2 Merge Sort**

Complexidade de tempo: Θ(nlogn)

Complexidade de espaço: Θ(n)

Complexidade no melhor, médio e pior caso: O(n\*log(n))

**3.3 Counting Sort**

Complexidade: O(n)

**4 Comparação do funcionamento dos algoritmos**

**4.1 Heap Sort**

O heapsort 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, lembrando-se sempre de manter a propriedade de max-heap.

A heap pode ser representada como uma árvore (uma árvore binária com propriedades especiais) ou como um vetor. Para uma ordenação decrescente, deve ser construída uma heap mínima (o menor elemento fica na raiz). Para uma ordenação crescente, deve ser construído uma heap máxima (o maior elemento fica na raiz).

Na abordagem em que a Heap é um vetor, podemos resumir o funcionamento do algoritmo utilizando a seguinte estratégia:

1. Selecione o menor item do vetor.

2. Troque-o com o item da primeira posição do vetor.

3. Repita estas operações com os (n – 1) itens restantes, depois com os (n – 2) itens, e assim sucessivamente.

**4.2 Merge Sort**

Sua ideia básica consiste em Dividir (o problema em vários sub-problemas e resolver esses subproblemas através da recursividade) e Conquistar (após todos os subproblemas terem sido resolvidos ocorre a conquista que é a união das resoluções dos sub-problemas).

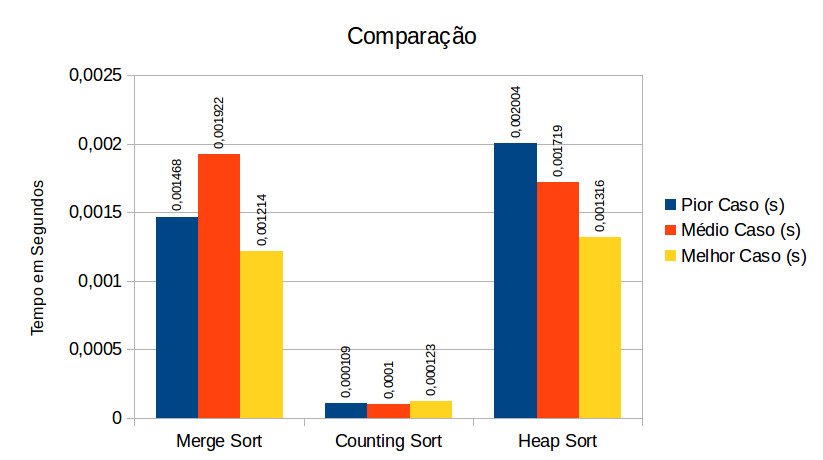
**4.3 Counting Sort**

A ideia básica do counting sort é determinar, para cada entrada x, o número de elementos menor que x. Essa informação pode ser usada para colocar o elemento x diretamente em sua posição no array de saída. Por exemplo, se há 17 elementos menor que x, então x pertence a posição 18. Esse esquema deve ser ligeiramente modificado quando houver vários elementos com o mesmo valor, uma vez que nós não queremos colocar eles na mesma posição.

**5. Experimento**

Para a realizado dos testes foram foi utilizado um computador com a seguinte configuração:

* **Processador:**
  + *Modelo:* Intel Core i7-5500U
  + *Memória cache:* 4M
  + *Número de núcleos:* 2
  + *Número de threads:* 4
  + *Frequência base:* 2,4 GHz
* **Memória RAM:**
  + *Arquitetura:* DDR3 L
  + *Tamanho:* 8GB
* **Sistema Operacional:** 
  + *Nome:*Ubuntu 16.04 LTS
  + *Versão Kernel:* 4.15.15-041515-generic
* Softwares utilizados:
  + *IDE:* Code::blocks IDE V16.01
  + *Compilador:* GCC 5.4.0



**6 Conclusão**

**7 Referências bibliográficas**

Counting Sort

**Counting sort** é um algoritmo de ordenação estável cuja complexidade é O(n). As chaves podem tomar valores entre 0 e M-1. Se existirem k0 chaves com valor 0, então ocupam as primeiras k0 posições do vetor final: de 0 a k0-1.

A ideia básica do counting sort é determinar, para cada entrada x, o número de elementos menor que x. Essa informação pode ser usada para colocar o elemento x diretamente em sua posição no array de saída. Por exemplo, se há 17 elementos menor que x, então x pertence a posição 18. Esse esquema deve ser ligeiramente modificado quando houver vários elementos com o mesmo valor, uma vez que nós não queremos colocar eles na mesma posição.

**Características**

* Estável
* Necessita de memória auxiliar. Logo, não é in-place
* Complexidade linear