**INTRODUÇÃO**

Este trabalho tem como a finalidade de mostrar os principais algoritmos de ordenação e seus desempenhos em algumas situações, são eles: Bubble Sort, Insertion Sort, Selection Sort, Merge Sort e Quick Sort.

O Bubble Sort é um dos algoritmos de ordenação mais simples, que consiste em percorrer os N elementos de um vetor, para cada vez percorrida, todos os elementos são comparados com o seu próximo, para verificar se estão na ordem desejada.

O Insertion Sort consiste em percorrer o vetor da esquerda para a direita pegando um valor que está passando e comparando ele com o próximo da sua esquerda e verificando se ele é maior que o seu antecessor.

O Selection Sort se baseia sempre em passar o menor valor para primeira posição ( ou o maior dependendo da ordem desejada) e depois o segundo menor valor e assim sucessivamente até os últimos dois elementos restantes do vetor, até percorrer por inteiro.

Merge Sort é um algoritmo de ordenação de dados simples e compacto que normalmente é implementado utilizando a recursão. A sua ideia é embasada na divisão do vetor em vetores menores e realiza a comparação entre os elementos e depois os agrupa de novo. O Merge Sort se baseia em um padrão chamado divisão e conquista (divide-and-conquer) e pode ser dividido em três partes: **Divisão**: Se a quantidade de elementos que serão ordenados for um ou dois então resolva o problema diretamente, se tiver mais elementos então divida a quantidade de elementos em dois ou mais conjuntos; **Recursão**: Utilize a recursão para resolver cada um dos subconjuntos de elementos; **Conquista**: Depois da resolução de cada subconjuntos, reagrupe-as em uma única solução.

O Quick Sort assim como o Merge Sort também utiliza da recursão, porem seu funcionamento é um pouco diferente do Merge Sort

Quicksort funciona localmente. E seu tempo de execução no pior caso é tão ruim quanto o das ordenações por seleção e por inserção: *￼*Θ(*n*2)\Theta, left parenthesis, n, squared, right parenthesis. Mas seu tempo de execução médio é tão bom quanto o merge sort: *￼*Θ(*n*lg*n*)\Theta, left parenthesis, n, \lg, n, right parenthesis. Então por que pensar em quicksort quando o merge sort é, pelo menos, igualmente bom? Porque o fator constante oculto na notação grande-Θ do quicksort é muito bom. Na prática, o quicksort tem desempenho melhor que o merge sort, e é significativamente melhor do que os algoritmos de seleção e de inserção.

É desta forma que o quicksort usa divisão e conquista. Assim como o merge sort, pense na ordenação de um subarray array[p..r], onde inicialmente o subarray é array[0..n-1].

**Divida** por qualquer elemento escolhido na subarray array[p..r]. Chame esse elemento de **pivô**. Reorganize os elementos em array[p..r] para que todos os elementos em array[p..r] que são menores ou iguais ao pivô fiquem a esquerda e que todos os elementos em array[p..r] fiquem a direita do pivô. Nós chamamos esse procedimento de **particionamento**. Nesse ponto, não importa a orderm que os elementos a esquerda do pivô estão em relação entre eles, e o mesmo vale para os elementos a direita do pivô. Nós apenas nos preocupamos que cada elemento esteja do lado correto do pivô.

Por uma questão de prática, nós iremos sempre escolher o elemento mais à direita da subarray, array[r], como o pivô. Então, por exemplo, se a subarray consiste de [9, 7, 5, 11, 12, 2, 14, 3, 10, 6], então nós escolhemos o 6 como pivô. Depois de particionarmos, a subarray pode parecer [5, 2, 3, 6, 12, 7, 14, 9, 10, 11]. Deixe q ser o indice de onde o pivô termina.

**Conquista** ordenando recursivamente as subarrays array[p..q-1] (todos os elementos à esquerda do pivô, que devem ser menor ou igual ao pivô) e array[q+1..r] (todos os elementos à direita do pivô, que devem ser maiores que o pivô).

**Combinar** sem fazer nada. Uma vez que o conquer step organize recursivamente, nós estaremos prontos. Por que? Todos os elementos à esquerda do pivô, na array[p..q-1], são menores ou iguais ao pivô e são organizados, e todos os elementos à direita do pivô, na array[q+1..r], são maiores que o pivô e são organizados. Os elementos na array[p..r] não podem ajudar mas são classificados!