**Radix Sort:**

O radix sort funciona com a ideia de ordenar de acordo com a base do número, fazendo ordenação através dos dígitos dos números, pegando do menos significativo para o mais significativo.

***Desempenho***:

***Número de comparações:***

Nenhuma comparação é realizada;

Inspeção de dígitos:

2\*n\*qtd\_digitor, se a quantidade de dígitos for a mesma em todos os números, o radix sort tem custo linear O(n).

***Número de trocas***:

N\*qtd\_digitos, o número de trocas também é linear O(n).

**Vantagens:**

* É estável;
* não faz comparações.

**Desvantagens**:

Depende do hardware, nem sempre é fácil otimizar a inspeção de dígitos.

Serve apenas para números inteiros positivos e pequenos.

Referência: <http://homepages.dcc.ufmg.br/~cunha/teaching/20121/aeds2/radixsort.pdf>

**Heap Sort:**

O heap sort trabalha de forma a rearranjar o vetor em uma heap, rearranjar a heap em ordem crescente, e para isso ele usa o conceito de árvore binária para trocar de lugar os elementos da heap, e ao final o vetor estará ordenado.

***Desempenho***:

***Número de comparações:***

2 n lg(m), o tempo do heap sort é proporcional ao número de comparações entre os elementos do vetor, e portanto proporcional a n log n no pior caso.

***Vantagens:***

Comportamento O(n log n) no pior caso

***Desvantagens***:

Não é estável.

Referência: <https://www.ime.usp.br/~pf/algoritmos/aulas/hpsrt.html>

<http://homepages.dcc.ufmg.br/~cunha/teaching/20121/aeds2/heapsort.pdf>

**Merge sort:**

O merge sort parte do princípio de divisão e conquista, ele divide o vetor em subvetores e recursivamente vai dividindo até não for mais possível dividir, após isso esses vetores menores são ordenados e juntados novamente até formar o vetor inicial ordenado.

***Desempenho***:

O(n log n), o tempo é proporcional a quantidade de movimentações no vetor.

***Vantagens***:

Tempos no pior caso O(n log n)

***Desvantagem***:

Consome muito mais memória devido a utilização de vetores auxiliares.

Referências: <http://www.cos.ufrj.br/~rfarias/cos121/aula_07.html>

<https://www.ime.usp.br/~pf/algoritmos/aulas/mrgsrt.html>

**Busca binária:**

Consiste em fazer busca de um elemento em um vetor ordenado, começando a busca na metade do vetor e se não estiver no elemento central do vetor é então verificado se o elemento procurado está para a metade direita ou esquerda do vetor e então esse processo é repetido até encontrar o elemento desejado ou que a busca se esgote não encontrando o elemento.

***Desempenho***:

O desempenho é proporcional ao número de comparações que mesmo no pior dos casos é aproximadamente log n.

Referências:

https://www.ime.usp.br/~pf/algoritmos/aulas/bubi.html