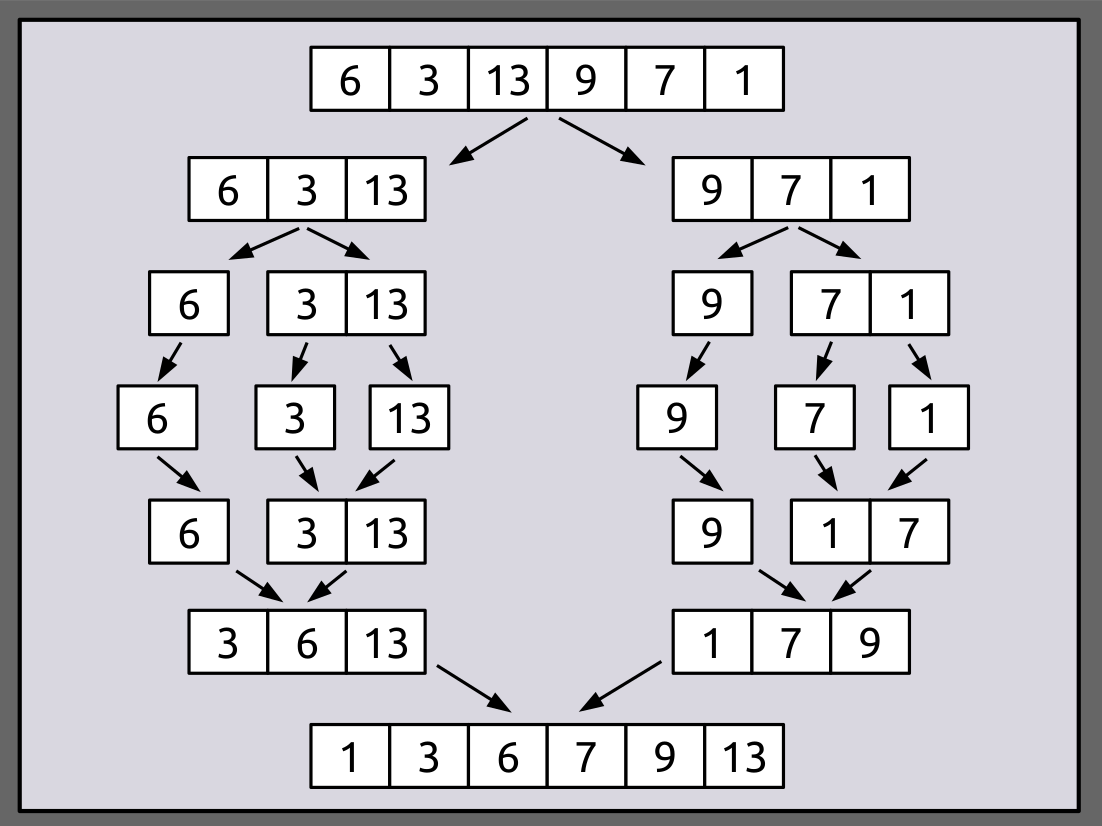
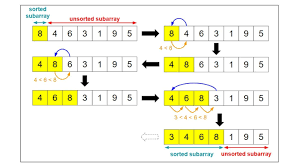
**Pesquisar sobre complexidade de algoritmo e calcule a complexidade dos algoritmos anteriores**

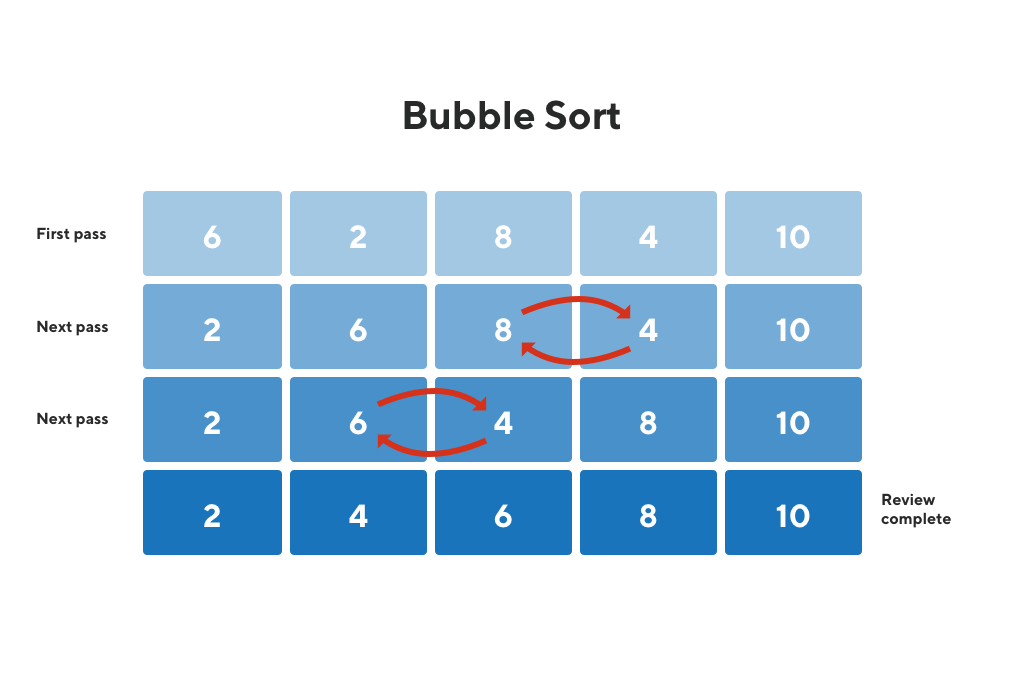
* **Merge Sort:** Ele é o algoritmo estável com o objetivo de ordenar uma lista original que é capaz de ser dividida em duas metades aproximadamente iguais. Ou seja, ele funciona basicamente, como se fosse uma combinação de técnica de divisão e se torna tão essencial e eficiente. E também ela consegue dividir uma lista original em sublistas menores, para ordenar separadamente entre uma delas, tanto para uma lista original, como também criar uma sublistas menores.

**- Complexidade:** A sua complexidade dele, depende do código em que ele necessita, por exemplo: É tempo de **O (n log n),** conforme usando a complexidade dele com base nesse tempo, ele consegue aumentar de forma proporcional ao produto de um número pela base de logarítmica desse número. O que é “n”? -> o n é uma representação de números que cria os elementos da lista para ser ordenada.

* **Insertion Sort:** Ele é um algoritmo que consegue percorrer um vetor de elementos da esquerda para a direita, e a medida que avança uma lista de elementos de números, ele vai alinhar e avançar sempre para direita quando vão ser ordenando os elementos à esquerda. Mas, ele é um algoritmo bem simples de usar quando se trata de usar um código pelo Python, porque consiste apenas cada passo a partir do segundo elemento selecionar o próximo item da sequência para direita e coloca-lo e alinhar no local apropriado da sua ordenação.

**- Complexidade:** Sua complexidade é **C(n) = O (N)**, considerado o método ordenação estável. Mas o que é? -> É quando uma lista numérica e tipos strings não se altera durante a ordenação da lista.

* **Bubble Sort:** Ele é um algoritmo simples e menos eficiente, no qual, o objetivo dele é receber uma lista de elementos que produz uma lista ordenada de acordo com um critério. Funciona para reorganizar os números até que estejam na ordem correta.

**- Complexidade:** A sua complexidade é o **O(1),** essa complexidade significa que ele ordena os elementos chamado “inplace” ou seja, não requer espaço adicional para armazenar os elementos durante a ordenação.

* **Busca Linear:** É um algoritmo que procura um valor específico em uma lista, começa a verificar cada elemento um a um. E também faz uma comparação entre cada elemento, começando pelo primeiro item até ele percorrer a lista para encontrar o seu alvo de comparação ao final da lista. É um algoritmo bem simples e intuitivo.

**- Complexidade:** A sua complexidade é

* **Busca Binária:** É um algoritmo de busca eficiente utilizado para encontrar um determinado elemento em uma lista ordenada. Ela é baseada no princípio de dividir pela metade, ou seja, a cada iteração, o algoritmo divide a lista em duas partes e verifica em qual metade o elemento procurado está presente. Dessa forma, elimina-se metade da lista a cada iteração, reduzindo drasticamente o tempo de busca em comparação com outros algoritmos de busca.

**- Complexidade:** A complexidade de tempo da busca binária em Python é **O(logn),** onde **n**, é o tamanho da lista. Isso significa que o tempo de execução do algoritmo cresce de forma proporcional ao logaritmo do tamanho da lista.