Atividade 2)
Implementamos os algoritmos Merge Sort e Bubble Sort e fizemos uma análise de como o algoritmo se comporta em diferentes casos.

Para o array = `[11, 12, 22, 25, 34, 64, 90]`

Usamos como melhor caso:
array melhor = `[11, 12, 22, 25, 34, 64, 90]`

Usamos como caso médio:
array médio = `[11,12,22,90,34,25,64]`

Usamos como pior caso:
array pior = `[90, 64, 34, 25, 22, 12, 11]`

Ao analisar as respostas dos dois algoritmos obtivemos:

**Para o Bubble Sort:**

Para o melhor caso no algoritmo bubble sort tivemos esse resultado;

```
Melhor caso
Array inicial: [11, 12, 22, 25, 34, 64, 90]
Tempo de execução: 0.0 segundos
```

Para o caso médio no algoritmo bubble sort tivemos esse resultado;

```
Caso medio
Array inicial: [11, 12, 22, 90, 34, 25, 64]
Iteração 1: [11, 12, 22, 34, 90, 25, 64]
Iteração 2: [11, 12, 22, 34, 25, 90, 64]
Iteração 3: [11, 12, 22, 34, 25, 64, 90]
Iteração 4: [11, 12, 22, 25, 34, 64, 90]
Tempo de execução: 0.0 segundos
```

Para o pior caso no algoritmo bubble sort tivemos esse resultado;

```
 Pior Caso
Array inicial: [90, 64, 34, 25, 22, 12, 11]
Iteração 1: [64, 90, 34, 25, 22, 12, 11]
Iteração 2: [64, 34, 90, 25, 22, 12, 11]
Iteração 3: [64, 34, 25, 90, 22, 12, 11]
Iteração 4: [64, 34, 25, 22, 90, 12, 11]
Iteração 5: [64, 34, 25, 22, 12, 90, 11]
Iteração 6: [64, 34, 25, 22, 12, 11, 90]
Iteração 7: [34, 64, 25, 22, 12, 11, 90]
Iteração 8: [34, 25, 64, 22, 12, 11, 90]
Iteração 9: [34, 25, 22, 64, 12, 11, 90]
Iteração 10: [34, 25, 22, 12, 64, 11, 90]
Iteração 11: [34, 25, 22, 12, 11, 64, 90]
Iteração 12: [25, 34, 22, 12, 11, 64, 90]
Iteração 13: [25, 22, 34, 12, 11, 64, 90]
Iteração 14: [25, 22, 12, 34, 11, 64, 90]
Iteração 15: [25, 22, 12, 11, 34, 64, 90]
Iteração 16: [22, 25, 12, 11, 34, 64, 90]
Iteração 17: [22, 12, 25, 11, 34, 64, 90]
Iteração 18: [22, 12, 11, 25, 34, 64, 90]
Iteração 19: [12, 22, 11, 25, 34, 64, 90]
Iteração 20: [12, 11, 22, 25, 34, 64, 90]
Iteração 21: [11, 12, 22, 25, 34, 64, 90]
Tempo de execução: 0.005764007568359375 segundos
```

**Para o Merge Sort:**

Para o melhor caso no algoritmo merge sort tivemos esse resultado;

```
Melhor caso
Array original: [11, 12, 22, 25, 34, 64, 90]
Iteração 1: [12, 22]
Iteração 2: [11, 12, 22]
Iteração 3: [25, 34]
Iteração 4: [64, 90]
Iteração 5: [25, 34, 64, 90]
Iteração 6: [11, 12, 22, 25, 34, 64, 90]
Total de iterações: 6
```

Para o caso médio no algoritmo merge sort tivemos esse resultado;

```
Caso Medio
Array original: [11, 12, 22, 90, 34, 25, 64]
Iteração 1: [12, 22]
Iteração 2: [11, 12, 22]
Iteração 3: [34, 90]
Iteração 4: [25, 64]
Iteração 5: [25, 34, 64, 90]
Iteração 6: [11, 12, 22, 25, 34, 64, 90]
Tempo de execução: 0.008049249649047852 segundos
```

Para o pior caso no algoritmo merge sort tivemos esse resultado;

```
Pior Caso
Array original: [90, 64, 34, 25, 22, 12, 11]
Iteração 1: [34, 64]
Iteração 2: [34, 64, 90]
Iteração 3: [22, 25]
Iteração 4: [11, 12]
Iteração 5: [11, 12, 22, 25]
Iteração 6: [11, 12, 22, 25, 34, 64, 90]
Tempo de execução: 0.0006568431854248047 segundos
```

Ao analisar as tabelas geradas por cada algoritmo em cada caso específico, é notável perceber que o algoritmo merge sort se torna muito mais rápido e barato para realizar processos onde os dados se encontram mais embaralhados.

**Agora, vamos verificar o funcionamento em um tamanho de dados maior**

```
arrMedio = [23, 7, 45, 12, 88, 34, 67, 56, 91, 11, 49, 72, 6, 39, 80]
arrMelhor = [6, 7, 11, 12, 23, 34, 39, 45, 49, 56, 67, 72, 80, 88, 91]
arrPior = [91,88,80,72,67,56,49,45,39,34,23,12,11,7,6]
```

***Para o algoritmo bubble sort tivemos:***
**Melhor caso:** Apenas retorna o vetor inicial ordenado;
**Caso Médio**: Tivemos **43** Iterações

```
Tempo de execução: 0.005266904830932617 segundos
```

**Pior Caso:** Tivemos **105** Iterações

```
Tempo de execução: 0.015620231628417969 segundos
Array ordenado: [6, 7, 11, 12, 23, 34, 39, 45, 49, 56, 67, 72, 80, 88, 91]
```

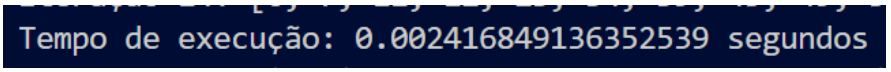
***Para o algoritmo merge sort tivemos:***
**Melhor caso**

```
Tempo de execução: 0.0 segundos
```

**Caso Médio**

```
Tempo de execução: 0.013520479202270508 segundos
```

**Pior Caso**




Com esses dois conjuntos de dados foi possível perceber que quanto maior a entrada de dados o algoritmo mergesort se mostra cada vez mais eficiente em relação ao algoritmo bubble sort

Indo mais afundo, podemos realizar uma análise assintótica dos dois algoritmos e percebemos que:

**Pior Caso BubbleSort**: O(n2)
**Pior Caso Mergesort**: O(nlogn)

Eficiência: O Mergesort é significativamente mais eficiente do que o Bubble Sort, especialmente para grandes conjuntos de dados, devido à sua complexidade de tempo O(nlogn)O(n \log n)O(nlogn) em todos os casos.

Uso de Espaço: Bubble Sort é mais econômico em termos de espaço, pois não requer espaço adicional significativo, enquanto Merge Sort utiliza espaço extra proporcional ao tamanho do array.

Aplicações: Merge Sort é geralmente preferido em aplicações que requerem um desempenho consistente em grandes volumes de dados. Bubble Sort, por outro lado, é raramente usado na prática devido à sua ineficiência, embora possa ser útil para fins educacionais e em conjuntos de dados muito pequenos.