Nomes:

```
 - Antonio Manuel Sitoe
- Nurdine Bacar
- Marcel Nota
```

Turma: 2P4LDS1

Relatório Comparativo sobre Algoritmos de Ordenação

Introdução: Algoritmos de ordenação são métodos utilizados para organizar um conjunto de elementos em uma determinada ordem. Neste relatório, analisaremos quatro algoritmos de ordenação comuns: Selection Sort, Bubble Sort, Insertion Sort e QuickSort. Exploraremos o funcionamento de cada algoritmo, suas diferenças, vantagens e desvantagens, além de fornecer uma tabela comparativa e discutir o desempenho relativo de cada método.

1. Selection Sort:

- **Funcionamento:** O Selection Sort funciona encontrando o menor elemento da lista e trocando-o com o primeiro elemento. Em seguida, encontra o próximo menor elemento e troca-o com o segundo elemento, e assim por diante.
- **Tempo de Execução:** O tempo de execução do Selection Sort é O(n^2) no pior caso, onde 'n' é o número de elementos na lista.
- **Diferenças:** É ineficiente para listas grandes devido à sua natureza de comparação repetitiva.
- **Defeitos:** Não é adaptativo (ou seja, não aproveita a ordenação parcial da lista).
- **Qualidades:** Simples de implementar e eficaz para listas pequenas.

a) Quantidade de Elementos do Array: 500:

```
 Tempo de execução em Segundos: 0.01119982
Numeros de Trocas: 493
Numeros de comparacoes: 124750
```

b) Quantidade de Elementos do Array: 5000:

```
 Tempo de execução em Segundos 0.080755689
Numeros de Trocas 4988
Numeros de comparacoes 12497500
```

c) Quantidade de Elementos do Array: 50_000:

```
 Tempo de execução em Segundos: 1.970780401
Numeros de Trocas: 49988
Numeros de comparacoes: 1249975000
```

2. Bubble Sort:

- **Funcionamento:** O Bubble Sort compara elementos adjacentes e os troca se estiverem na ordem errada, fazendo com que o maior elemento "borbulhe" para o final da lista.
- **Tempo de Execução:** O tempo de execução do Bubble Sort é O(n^2) no pior caso.
- **Diferenças:** Possui um alto número de comparações, o que o torna menos eficiente para listas grandes.
- **Defeitos:** Não é eficiente para grandes conjuntos de dados. Não é adaptativo.
- **Qualidades:** Simples de entender e implementar.

a) Quantidade de Elementos do Array: 500:

```
 Tempo de execução em Segundos: 0.004146787
Numeros de Trocas: 64195
Numeros de comparacoes: 64195
```

b) Quantidade de Elementos do Array: 5000:

```
 Tempo de execução em Segundos 0.076820231
Numeros de Trocas 6304219
Numeros de comparacoes 6304219
```

c) Quantidade de Elementos do Array: 50_000:

```
 Tempo de execução em Segundos 4.483014312
Numeros de Trocas 626434661
Numeros de comparacoes 626434661
```

3. Insertion Sort:

- **Funcionamento:** O Insertion Sort percorre a lista, inserindo cada elemento em sua posição correta na parte já ordenada da lista.

- **Tempo de Execução:** O tempo de execução do Insertion Sort é O(n^2) no pior caso.
- **Diferenças:** Melhora o desempenho para listas quase ordenadas, pois requer menos comparações.
- **Defeitos:** Ainda possui complexidade quadrática em cenários desfavoráveis.
- **Qualidades:** Eficiente para listas pequenas e quase ordenadas.

a) Quantidade de Elementos do Array: 500:

```
Tempo de execução em Segundos 0.00560595
Numeros de Trocas 64195
Numeros de comparacoes 64195
```

b) Quantidade de Elementos do Array: 5000:

```
Tempo de execução em Segundos 0.042750499
Numeros de Trocas 6304219
Numeros de comparacoes 6304219
```

c) Quantidade de Elementos do Array: 50_000:

```
Tempo de execução em Segundos 0.295699073
Numeros de Trocas 626434661
Numeros de comparacoes 626434661
```

4. QuickSort:

- **Funcionamento:** O QuickSort escolhe um elemento pivô e particiona a lista em dois subconjuntos: elementos menores que o pivô e elementos maiores que o pivô. Em seguida, aplica recursivamente o mesmo processo nos subconjuntos.
- **Tempo de Execução:** Em média, o QuickSort tem um tempo de execução O(n log n), tornando-o mais eficiente para listas maiores.
- **Diferenças:** Usa estratégia de "dividir para conquistar", o que o torna mais rápido para listas grandes.
- **Defeitos:** Pode degradar para O(n^2) no pior caso se o pivô for mal escolhido.
- **Qualidades:** Um dos algoritmos de ordenação mais rápidos em média.

a) Quantidade de Elementos do Array: 500:

```
Tempo de execução em Segundos 0.05479349
Numeros de Trocas 80783
Numeros de comparacoes 80646
```

b) Quantidade de Elementos do Array: 5000:

```
Tempo de execução em Segundos 4.892469862
Numeros de Trocas 12288660
Numeros de comparacoes 12283770
```

c) Quantidade de Elementos do Array: 50_000:

```
Tempo de execução em Segundos: _
Numeros de Trocas: _
Numeros de comparacoes: _
```

Comparação: Aqui está uma tabela comparativa dos algoritmos com base em diferentes critérios:

Algoritmo	Tempo de Execução	Melhor Caso	Pior Caso	Uso Adaptativo
SelectionSort	O(n^2)	O(n^2)	O(n^2)	Não
BubbleSort	O(n^2)	O(n)	O(n^2)	Não
InsertionSort	O(n^2)	O(n)	O(n^2)	Sim
QuickSort	O(n log n) em média	O(n log n)	O(n^2)	Não

Desempenho: Para listas pequenas ou quase ordenadas, o Insertion Sort tende a ter melhor desempenho. Para listas grandes, o QuickSort é geralmente mais eficiente devido à sua complexidade média de tempo O(n log n). Ambos o Selection Sort e Bubble Sort são menos eficientes, especialmente para listas grandes.

Conclusão: Neste relatório, analisamos quatro algoritmos de ordenação: Selection Sort, Bubble Sort, Insertion Sort e QuickSort. Cada algoritmo possui suas próprias vantagens e desvantagens em termos de tempo de execução e eficiência. A escolha do algoritmo depende do tamanho da lista, do grau de ordenação prévia e das restrições de tempo. O QuickSort é frequentemente preferido para listas maiores, enquanto o Insertion Sort pode ser útil para listas menores ou quase ordenadas.