

**Instituto Federal de Educação, Ciência e Tecnologia da Paraíba**

**Curso: Análise e Desenvolvimento de Sistemas**

**Disciplina: Análise de Algoritmos**

**Docente: José Gomes Lopes Filho**

**Aluno: André Felipe Ferraz da Silva**

**Análise Experimental de Algoritmos**

**Monteiro – PB**

**2021**

**SUMÁRIO**

[1. Introdução 3](#_Toc79924830)

[2. Algoritmos 3](#_Toc79924831)

[2.1. Bubble Sort 3](#_Toc79924832)

[2.1.1. Implementação do Bubble Sort 3](#_Toc79924833)

[2.2. Insertion Sort 3](#_Toc79924834)

[2.2.1. Implementação do Insertion Sort 4](#_Toc79924835)

[2.3. Quick Sort 4](#_Toc79924836)

[2.3.1. Implementação do Quick Sort 4](#_Toc79924837)

[2.4. Selection Sort 5](#_Toc79924838)

[2.4.1. Implementação do Selection Sort 5](#_Toc79924839)

[2.5. Merge Sort 5](#_Toc79924840)

[2.5.1. Implementação do Merge Sort 6](#_Toc79924841)

[3. Análise dos Algoritmos 6](#_Toc79924842)

[3.1. Vetores Ordenados 6](#_Toc79924843)

[3.1.1. Desempenho Bubble Sort 6](#_Toc79924844)

[3.1.2. Desempenho Insertion Sort 7](#_Toc79924845)

[3.1.3. Desempenho Quick Sort 7](#_Toc79924846)

[3.1.4. Desempenho Selection Sort 8](#_Toc79924847)

[3.1.5. Desempenho Merge Sort 8](#_Toc79924848)

[3.2. Vetores Ordenados Inversamente 9](#_Toc79924849)

[3.2.1. Desempenho Bubble Sort 9](#_Toc79924850)

[3.2.2. Desempenho Insertion Sort 10](#_Toc79924851)

[3.2.3. Desempenho Quick Sort 10](#_Toc79924852)

[3.2.4. Desempenho Selection Sort 10](#_Toc79924853)

[3.2.5. Desempenho Merge Sort 11](#_Toc79924854)

[3.3. Vetores Quase Ordenados 11](#_Toc79924855)

[3.3.1. Desempenho Bubble Sort 12](#_Toc79924856)

[3.3.2. Desempenho Insertion Sort 12](#_Toc79924857)

[3.3.3. Desempenho Quick Sort 12](#_Toc79924858)

[3.3.4. Desempenho Selection Sort 13](#_Toc79924859)

[3.3.5. Desempenho Merge Sort 13](#_Toc79924860)

[3.3.6. Comparação dos Algoritmos 14](#_Toc79924861)

[3.4. Vetores Aleatórios 15](#_Toc79924862)

[3.4.1. Desempenho Bubble Sort 15](#_Toc79924863)

[3.4.2. Desempenho Insertion Sort 16](#_Toc79924864)

[3.4.3. Desempenho Quick Sort 16](#_Toc79924865)

[3.4.4. Desempenho Selection Sort 17](#_Toc79924866)

[3.4.5. Desempenho Merge Sort 17](#_Toc79924867)

[3.4.6. Comparação dos Algoritmos 17](#_Toc79924868)

[4. Conclusão 19](#_Toc79924869)

# Introdução

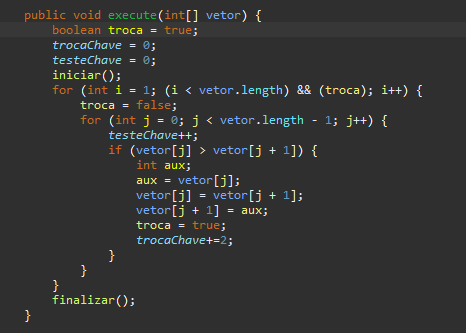
Este trabalho consiste em analisar o desempenho de um algoritmo de ordenação através de análises assintóticas, número de comparações de chaves, número de movimentações de registros e tempo total gasto de execução. Para a análise, foram usados vetores inteiros com tamanho: 10, 100, 1.000, 10.000, 100.000 e 1.000.000. Além disso, os vetores foram testados em arranjos ordenados, inversamente ordenados, quase ordenados e aleatórios, sendo testado 1 vez para cada algoritmo em vetores ordenados, 1 vez para cada algoritmo em vetores inversamente ordenados e 14 vezes para cada algoritmo em vetores quase ordenados e aleatórios.

# Algoritmos

# Bubble Sort

É um algoritmo que percorre várias vezes um mesmo vetor com a intenção de ordená-lo. Seu custo é depor isso, sua complexidade é no pior e médio caso e no melhor caso. O método Bubble Sort usado neste trabalho foi um Bubble Sort melhorado.

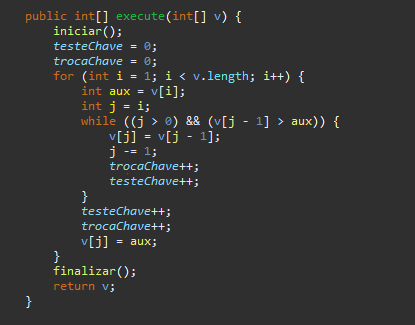
# Implementação do Bubble Sort



# Insertion Sort

É um algoritmo que, para ordenar um vetor, faz a inserção de um elemento por vez. Este algoritmo e bastante eficiente principalmente em casos pequenos, ele vai ordenando o vetor aos poucos e, ao ordenar aquela parte, introduz mais um elemento do vetor para ordena-lo. Seu custo é de por isso, sua complexidade é no pior e médio caso e no melhor caso.

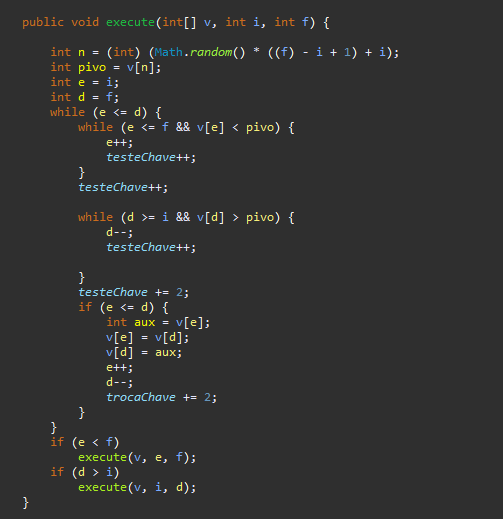
# Implementação do Insertion Sort

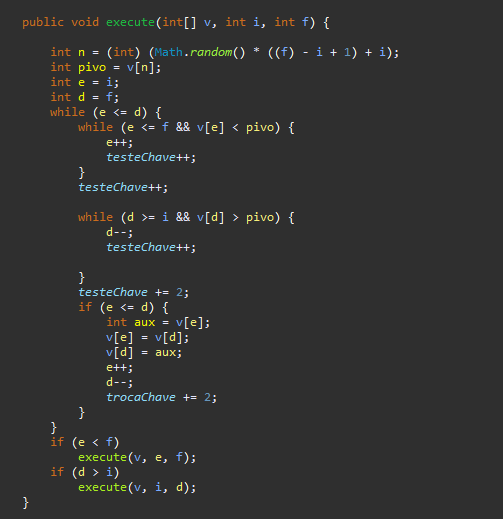


# Quick Sort

É um algoritmo que, através da fragmentação do vetor e da escolha de um pivô como meta comparativa para fazer a troca de chaves, ordena todas as fragmentações criadas apartir do vetor. Seu custo é de por isso, utilizando o método mestre, vemos que sua complexidade é no pior caso e no médio e no melhor caso.

# Implementação do Quick Sort

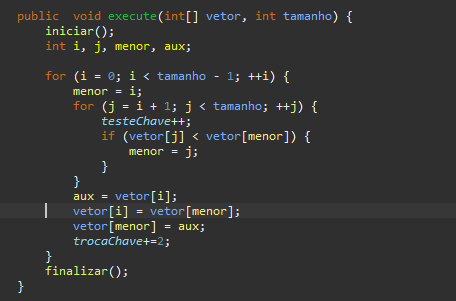




# Selection Sort

É um algoritmo baseado em percorrer o vetor a procura do menor elemento da vez e colocá-lo no começo do vetor, o que faz que ele seja muito pouco eficiente. Seu custo é de por isso sua complexidade é no pior, médio e melhor caso.

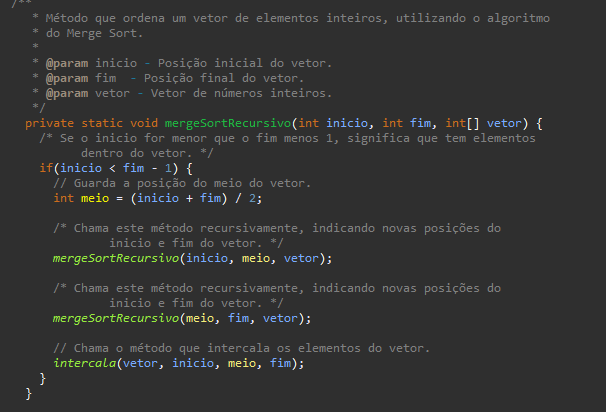
# Implementação do Selection Sort

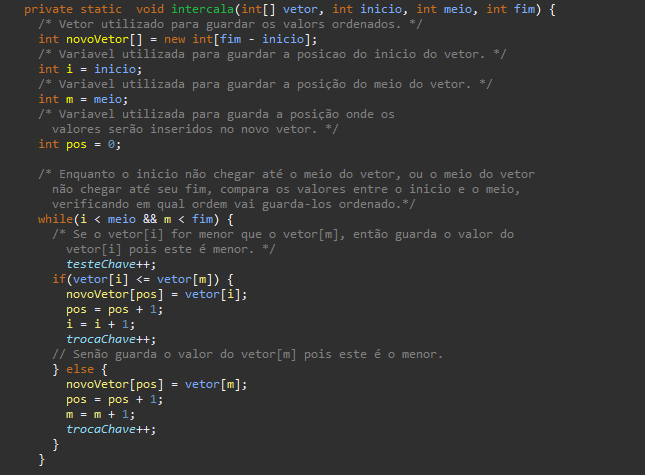


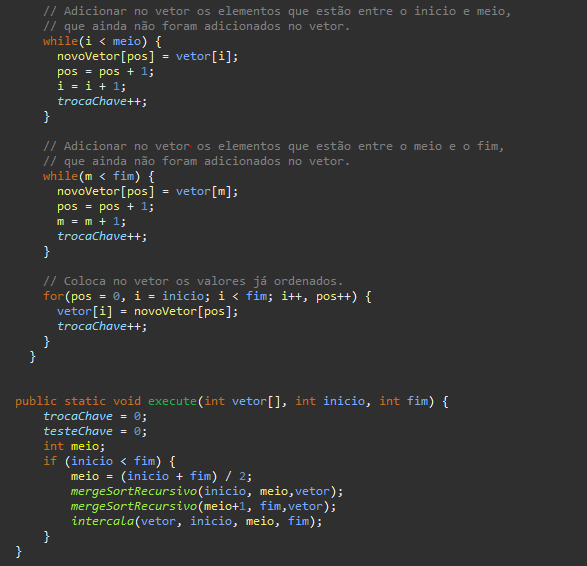
# Merge Sort

É um algoritmo que divide o vetor em várias partes até chegar em um valor mínimo, ele começa a ordenar essas partes e vai juntando ao final todas as partes ordenadas, isso permite que este algoritmo seja bastante efetivo, mas tem um alto consumo de memória. Seu custo é de por isso, utilizando o método mestre vemos que sua complexidade é sempre no melhor, médio e no pior caso.

# Implementação do Merge Sort







# Análise dos Algoritmos

# Vetores Ordenados

# Desempenho Bubble Sort

Esse gráfico descreve o desempenho do método Bubble Sort melhorado para ordenar vetores com 10, 100, 1.000, 10.000, 100.000 e 1.000.000 de elementos já ordenados. Usando vetores já ordenados o Bubble Sort melhorado sempre fica com sua taxa de tempo de processamento em 0 milissegundos e seus testes de chaves sempre vão ser o tamanho do vetor – 1.

# Desempenho Insertion Sort

Esse gráfico descreve o desempenho do método Insertion Sort para ordenar vetores com 10, 100, 1.000, 10.000, 100.000 e 1.000.000 de elementos já ordenados. Usando vetores já ordenados o Insertion Sort mantem sua taxa de tempo de processamento abaixo de 20 milissegundos, seus testes e suas trocas de chaves sempre vão ser o tamanho do vetor – 1.

# Desempenho Quick Sort

Esse gráfico descreve o desempenho do método Quick Sort para ordenar vetores com 10, 100, 1.000, 10.000, 100.000 e 1.000.000 de elementos já ordenados. Este metódo de ordenação, mesmo usando vetores já ordenados, realiza muitas trocas e testes de chaves, mas isso não afeta muito em seu tempo de processamento, isso se da pelo fato que o Quick Sort trabalha fragmentando o vetor.

# Desempenho Selection Sort

Esse gráfico descreve o desempenho do método Selection Sort para ordenar vetores com 10, 100, 1.000, 10.000, 100.000 e 1.000.000 de elementos já ordenados. Dos metódos de ordenação usados em vetores ordenados, o Selection Sort foi o que teve o pior desempenho disparado, pois ele, em qualquer situação, sempre vai ser O(N²). Só aprensentou baixo tempo de processamento para entradas pequenas mas, mesmo assim, ele realiza trocas desnecessárias.

# Desempenho Merge Sort

Esse gráfico descreve o desempenho do método Merge Sort para ordenar vetores com 10, 100, 1.000, 10.000, 100.000 e 1.000.000 de elementos já ordenados. O Merge Sort se mostrou um algortimo que realiza muitas trocas de chaves (Mesmo com o vetor já ordenado) seu tempo de processamento é relativamente baixo, quase se aproximando do tempo do Quick Sort.

# Vetores Ordenados Inversamente

# Desempenho Bubble Sort

Esse gráfico descreve o desempenho do método Bubble Sort para ordenar vetores com 10, 100, 1.000, 10.000, 100.000 e 1.000.000 de elementos ordenados inversamente. Por os vetores estarem inversamente ordenados, o Bubble Sort sempre entrava em seu pior caso, que é O(N²). Tanto seu tempo de processamento, seus testes de chaves e trocas de chaves alcançaram números altissímos, sendo muito inviável usar este metódo de orndenação para este tipo de problema.

# Desempenho Insertion Sort

Esse gráfico descreve o desempenho do método Insertion Sort para ordenar vetores com 10, 100, 1.000, 10.000, 100.000 e 1.000.000 de elementos ordenados inversamente. A mesma coisa que acontece com o Bubble Sort acontece aqui, sempre entra em seu pior caso (O(N²)), mas o Insertion Sort ainda se saiu melhor que o algoritmo anterior, cortando seu tempo de processamento quase pela metade.

# Desempenho Quick Sort

Esse gráfico descreve o desempenho do método Quick Sort para ordenar vetores com 10, 100, 1.000, 10.000, 100.000 e 1.000.000 de elementos ordenados inversamente. Comparando com o vetor já ordenado, este metódo de ordenação teve um leve crescimento em seu tempo de processamento.

# Desempenho Selection Sort

Esse gráfico descreve o desempenho do método Selection Sort para ordenar vetores com 10, 100, 1.000, 10.000, 100.000 e 1.000.000 de elementos ordenados inversamente. Não há muito o que falar do Selection Sort, ele sempre terá a mesma quantidade de troca e teste de chaves, apenas seu tempo de processamento que é mudado e, como vemos aqui, seu tempo teve um grande aumento, seu tempo é quase equivalente ao do Bubble Sort.

# Desempenho Merge Sort

Esse gráfico descreve o desempenho do método Merge Sort para ordenar vetores com 10, 100, 1.000, 10.000, 100.000 e 1.000.000 de elementos ordenados inversamente. O Merge Sort mantem a mesma quantidade de trocas de chaves, mesmo com um número alto de troca de chaves ele teve um tempo de processamento melhor que o do Quick Sort.

# Vetores Quase Ordenados

Para está análise, foi usado o mesmo vetor para todos os metódos de ordenação (5), os metódos foram rodados 14 vezes, o que significa que foram testados 14 tipos de vetores quase ordenados. O que será aprensentado é a média do tempo, troca de chaves e teste de chaves desses 14 vetores.

# Desempenho Bubble Sort

O Bubble Sort, como de se esperando, apresentou números bons para entradas pequenas e pessímos números para entradas grandes, mesmo com 10% porcento do vetor desorganizado, o Bubble Sort leva muito tempo e faz muitos teste de chaves para poder ordenar os números.

# Desempenho Insertion Sort

O Insertion Sort continua se saindo melhor que o Bubble Sort, ficando atrás apenas em vetores de tamanho 10. Com vetores grandes seu tempo, trocas e testes de chaves aumentam muito, ficando inviável de se usar para esses problemas.

# Desempenho Quick Sort

O Quick Sort é o que apresenta os melhores resultados no quesito tempo de processamento, ele ainda continua apresentando um número de teste de chaves bem alto, mas nada que o comprometa.

# Desempenho Selection Sort

Todos os seus casos são O(N²). O algoritmo aprensentou ser mais eficais que o Bubble Sort para vetores de tamanho grande.

# Desempenho Merge Sort

Realiza menos teste de chaves que o Quick Sort, mas realiza mais trocas de chaves, seu tempo é um pouco maior que o do Quick Sort. Sua troca de chaves é fixa para cada tamanho de vetor.

# Comparação dos Algoritmos

**Quadro 1 – Desempenho do Tempo dos Algoritmos**

**Quadro 2 – Desempenho dos Testes de Chaves dos Algoritmos**

**Quadro 3 – Desempenho das Trocas de Chaves dos Algoritmos**

# Vetores Aleatórios

Para está análise, foi usado o mesmo vetor para todos os metódos de ordenação (5), os metódos foram rodados 14 vezes, o que significa que foram testados 14 tipos de vetores quase ordenados. O que será aprensentado é a média do tempo, troca de chaves e teste de chaves desses 14 vetores.

# Desempenho Bubble Sort

Com valores aleatórios no vetor o Bubble Sort continou como o pior algoritmo para tratar vetores deste tipo, com seu tempo de processamento chegando à quase 35 minutos e inumeros testes e trocas de chaves.

# Desempenho Insertion Sort

O Insertion Sort se sai bem melhor que o algoritmo de ordenação anterior, seu tempo pra ordenar um vetor com um milhão de elementos fica na casa dos 4 minutos.

# Desempenho Quick Sort

Continua apresentando números satisfatórios, tendo numeros de tempos superiores a 10 milisegundo apartir de vetores de 100 mil elementos.

# Desempenho Selection Sort

É caracteristica do Selection Sort ter resutados até que bons para vetores pequenos e resultados ruins para grandes vetores, e aqui, não teve um resultado diferente.

# Desempenho Merge Sort

O Merge Sort continua se saindo muito bem, mesmo sendo um algoritmo recursivo, seu tempo com vetores grandes não aumenta tanto assim.

# Comparação dos Algoritmos

**Quadro 1 – Desempenho do Tempo dos Algoritmos**

**Quadro 2 – Desempenho dos Testes de Chaves dos Algoritmos**

**Quadro 3 – Desempenho das Trocas de Chaves dos Algoritmos**

# Conclusão

Tanto o Quick Sort quanto o Merge Sort são ótimos algoritmos de ordenação, alcançando tempos baixos para ordenar vetores de grande porte, sendo o Quick o maior realizador de testes chaves entre os dois e o Merge o que realiza mais trocas (Pelo fato de não reconhecer se um vetor já está ordenado). Já os outros três algortimos apresentaram números bons para pequenos vetores, mas com grandes vetores eles não se sairam tão bem, sendo o Bubble Sort o pior entre eles, seguido do Selection Sort e o Insertion Sort.

Comparando todos os algoritmos, o tempo de cada um para vetores pequenos é igual ou quase equivalente, já a partir de 100 mil elementos em um vetor os números começam a ter uma diferença griantante entre o Quick e Merge com os outros três algoritmos, não que o tempo dos dois não aumente, teve sim um aumento de tempo, mas não tão alto quanto os outros três, o aumento é minimo.

Portanto, pelo dados aprensentado neste estudo, o Quick e o Merge Sort são algoritmos bons para se trabalhar tanto com vetores grandes quanto com vetores pequenos, já os outros três só são viáveis de serem usados para vetores pequenos.

# 