Universidade Federal do Rio Grande do Norte – UFRN
Centro de Ensino Superior do Seridó – CERES
Departamento de Computação e Tecnologia – DCT
Bacharelado em Sistemas de Informação – BSI

Modelo de Referência para Escrita de Monografias e Relatórios do LabEPI

Vinicius Maia Marinho

Orientador: Prof. Dr. João Paulo de Souza Medeiros

Relatório Técnico apresentado ao Curso de Bacharelado em Sistemas de Informação como parte dos requisitos para aprovação na atividade de Estrutura de Dados.

Laboratório de Elementos do Processamento da Informação – LabEPI
Caicó, RN, 29 de maio de 2023

UFRN / Biblioteca Central Zila Mamede.

Catalogação da Publicação na Fonte.

Aluno, Vinicius Maia Marinho

Modelo de Referência para Escrita de Monografias e Relatórios do LabEPI. / Nome Vinicius Maia Marnho. – Caicó, RN, 2014.

20 f.: il.

Orientador: Prof. Dr. João Paulo de Souza Medeiros.

Relatório Técnico – Universidade Federal do Rio Grande do Norte. Centro de Ensino Superior do Seridó. Bacharelado em Sistemas de Informação.

1. Estrutura de dados. 2. Algorítmos de ordenação. 3. Relatório Técnico. I. Professor, João Paulo II. Universidade Federal do Rio Grande do Norte. III. Relatório Técnico para a disciplica de estrutura de dados.

RN/UF/BCZM CDU 004.7

Resumo

Este trabalho apresenta uma análise dos algoritmos de ordenação: distribution, insertion, merge, quick e selection. O objetivo é compreender o funcionamento de cada algoritmo e avaliar o tempo de execução deles.

Palavras-chave: Estrutura de dados. Algorítmos de ordenação. Relatório Técnico.

Abstract

This work presents an analysis of sorting algorithms: distribution, insertion, merge, quick and selection. The goal is to understand the operation of each algorithm and evaluate their execution time.

Keywords: Data structures; Order algorithms; Technical report.

Sumário

Lista de Algoritmos

Lista de Figuras

1. Introdução

"If knowledge can create problems,
it is not through ignorance that we can solve them."
Isaac Asimov

Este relatório tem como objetivo apresentar os resultados obtidos por meio da análise do tempo de execução de diferentes algoritmos de ordenação. A ordenação de elementos é uma tarefa fundamental no campo da computação, pois permite a organização dos dados de acordo com a necessidade, facilitando a busca, comparação e manipulação dessas informações.

Neste estudo, foram analisados os seguintes algoritmos de ordenação: selection-sort, insertion-sort, merge-sort, quick-sort e distribution-sort. Cada um desses algoritmos possui características distintas em relação ao tempo de execução e complexidade, o que nos permite compará-los e identificar suas eficiências em diferentes cenários.

O objetivo principal deste relatório é fornecer uma visão geral do desempenho de cada algoritmo em relação ao tempo de execução, com base em uma série de testes realizados em conjuntos de dados de diferentes tamanhos. Para isso, foram registrados os tempos de execução de cada algoritmo em diferentes situações, permitindo uma análise comparativa.

2. Selection Sort

"We can only see a short distance ahead,
but we can see plenty there that needs to be done."
Alan Mathison Turing

2.1 Introdução

O selection sort é do tipo de comparação in-place, o que quer dizer que ele não precisa de memória extra, ele ordena no mesmo vetor que foi passado. Ele percorre repetidamente a lista de elementos em busca do menor ou maior valor da parte não ordenada e o passando para a parte ordenada da lista. Esse processo de seleção é repetido até que toda a lista esteja ordenada.

Algoritmo 2.1. É possível calcular os graus de entrada e saída de cada nó da rede de forma iterativa com base na representação por lista de adjacência.

Embora seja fácil de entender e implementar, o Selection Sort têm um desempenho menos eficiente quando comparado a outros algoritmos de ordenação. Sua complexidade de tempo é O(n^2) onde n é o número de elementos na lista. Isso significa que o tempo de execução aumenta quadraticamente à medida que o tamanho da lista cresce, tornando-o menos adequado para grandes conjuntos de dados.

algoritmo selection-sort(v, n)

```
1: para i de 1 até |N − 1| faça
2:     m ← i
3:     para j de (i + 1) até n faça
4:         se v[m] > v[j] então
5:             m ← j
6:         fim se
7:     fim para
       swap(v[m],v[i])
8: fim para
```

□

2.2 Conclusão

Independente da maneira que o vetor está organizado ele rodará de maneira semelhante todas as vezes, assim, não tendo pior ou melhor caso, desta maneira, O(n^2), como é possível verificar na figura 2.1.

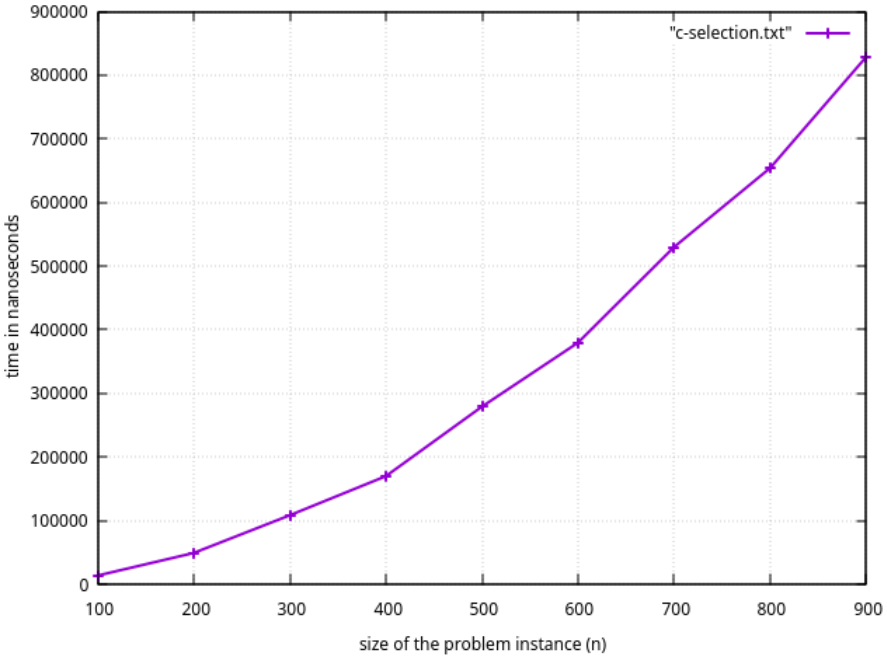

Figura 2.1: Tempo de execução médio do algoritmo Selection Sort

3. Insertion Sort

"Mathematical elegance is not a dispensable luxury
but a factor that decides between success and failure."
Edsger Wybe Dijkstra

3.1 Introdução

O Insertion Sort é um algoritmo de ordenação que utiliza apenas um vetor e se baseia na técnica "Diminuir para conquistar", o algoritmo inicialmente ordena uma parte do vetor e após isso vai aumentar o seu "grau de visão" alcançando os pontos desordenados e os ordenando.

Algoritmo 3.1. **algoritmo** selection-sort(v, n)

```
1: para i de 2 até |N| faça
2:     m ← i
3:     enquanto i > 1 & v[i − 1] > v[i] faça
3:         swap(v[m],v[i])
4:         i ← i − 1
5:     fim enquanto
6: fim para
```

□

3.2 Melhor Caso

O melhor caso deste algoritmo acontece quando o vetor já está ordenado, pois assim as trocas não acontecem, possuindo uma solução linear de análise assintótica O(n), como é possível verificar na figura 3.1.

3.3 Pior Caso

O pior caso acontece quando o vetor está em ordem decrescente, já que dessa maneira ele terá que realizar as trocas e rodar um número maior de vezes. Assim tendo uma solução quadrática de análise assintótica O(n^2), como é possível verificar na figura 3.2.

3.4 Caso Médio

O Caso médio acontece com vetores aleatórios e seu tempo de execução resulta em O(n^2), como é possível verificar na figura 3.1.

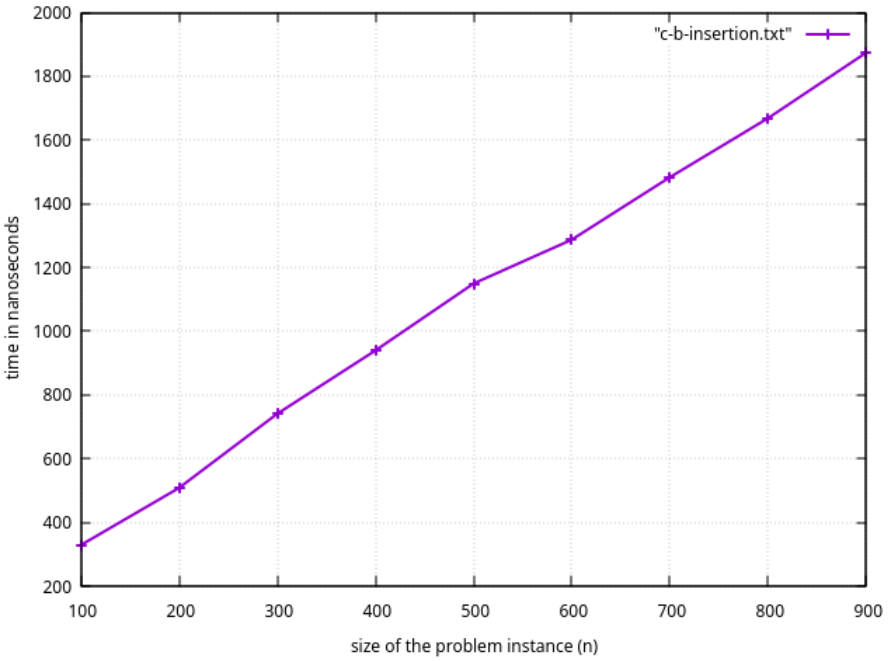

Figura 3.1: Melhor caso do algoritmo Insertion Sort

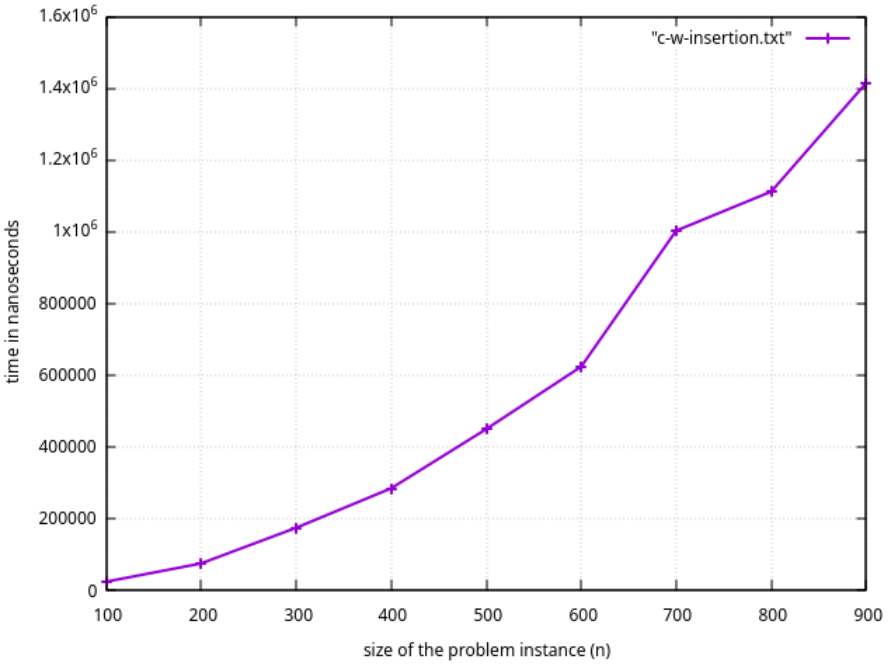

Figura 3.2: Pior caso do algoritmo Insertion Sort

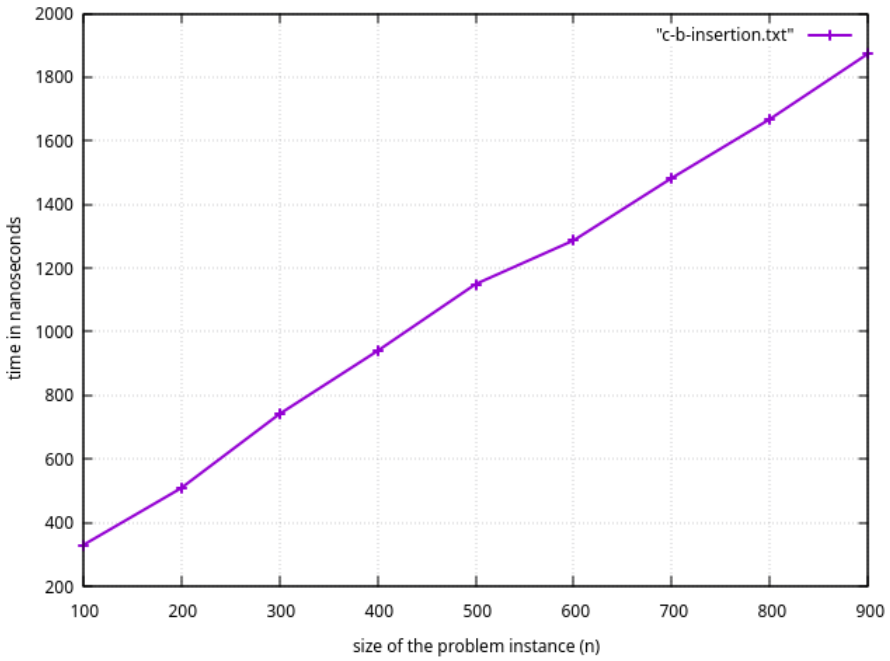

Figura 3.3: Caso médio do algioritmo Insertion Sort

3.5 Conclusão

No gráfico da figura 3.1 é possível visualizar o tempo de execução dos casos anteriormente apresentados.

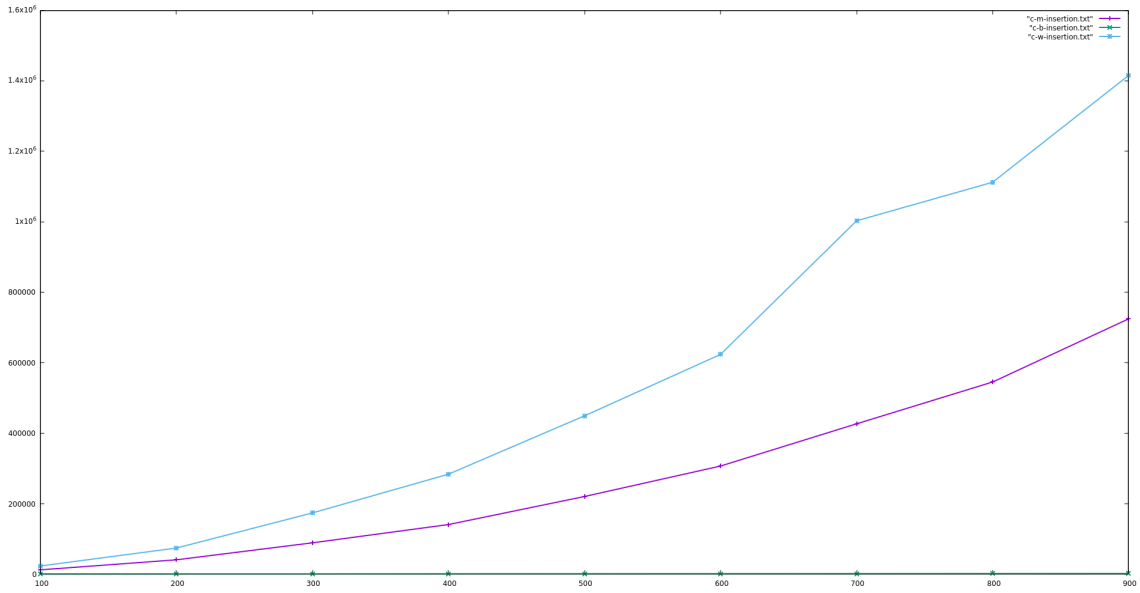

Figura 3.4: Tempo de execução em médio de comparação entre casos do quick sort

4. Quick Sort

"If we can really understand the problem,
the answer will come out of it,
because the answer is not separate from the problem."
Jiddu Krishnamurti

4.1 Introdução

O Quick Sort é um algoritmo de ordenação eficiente que utiliza a estratégia "dividir para conquistar". Ele seleciona um elemento pivô da lista, rearranja os elementos de forma que os elementos menores que o pivô fiquem antes dele, e os elementos maiores fiquem depois. Esse processo é realizado varias vezes nas sublistas resultantes até que a lista esteja completamente ordenada.

Algoritmo 4.1. **algoritmo** quick-sort(v, s, e)

```
1: se s < e então
2:     p ← partition(v, s, e)
       quick-sort(v,s,p-1)
       quick-sort(v,p-1,e)
3: fim se
algoritmo partition(v, s, e)
   d ← s − 1
2: para i de s para (e − 1) faça
       se v[i] <= v[e] então
4:         d ← d + 1
           swap(v[d],v[i])
       fim se
6: fim para
```

□

4.2 Melhor Caso

O melhor caso ocorre quando a escolha do pivô sempre divide a lista em duas partes iguais ou similares. Neste caso o tempo de execução é otimizado e a complexidade de tempo é de $O(n \log n)$.

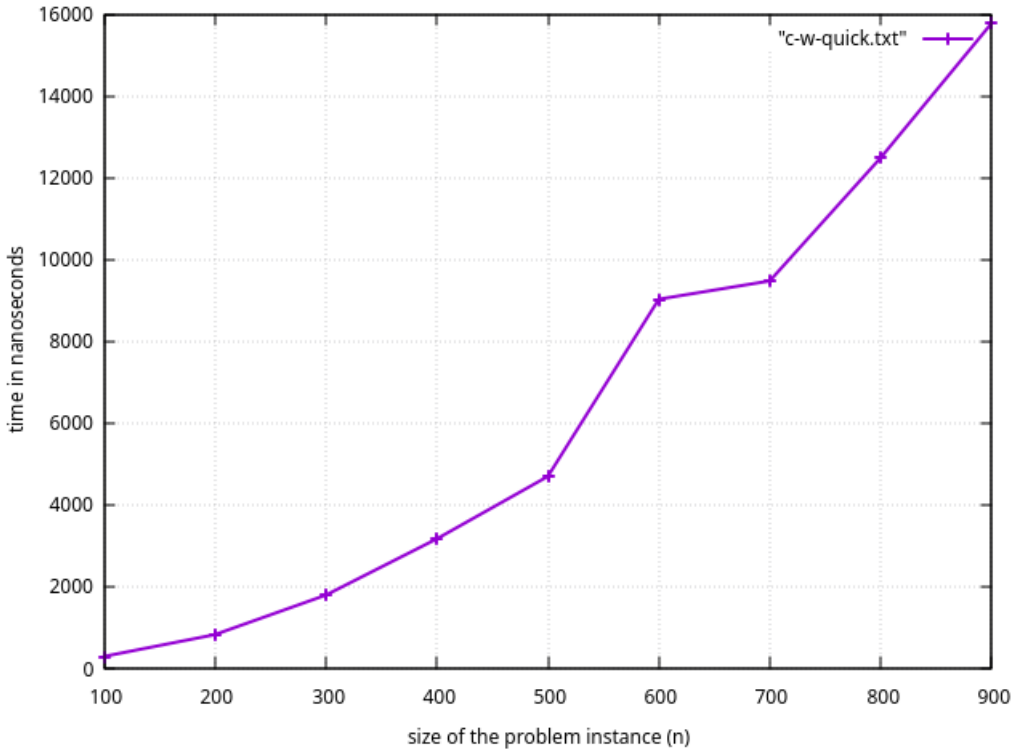

Figura 4.1: Tempo de execução do pior caso do Quick Sort

4.3 Pior Caso

O pior caso acontece quando a escolha do pivô leva a uma divisão desbalanceada das sublistas, acontecendo por exemplo, quando a escolha do pivô é o maior ou menor elemento da lista. Nesse caso o algoritmo se torna ineficiente e alcança uma complexidade de tempo de $O(n^2)$.

4.4 Caso Médio

O Caso médio acontece quando o pivô resulta em divisões balanceadas das sublistas de cada etapa de particionamento, como é possível verificar na figura 4.1.

4.5 Conclusão

Na figura 4.3 é possível visualizar o tempo de execução dos casos anteriormente apresentados e a diferença entre a execução do caso médio e do pior caso.

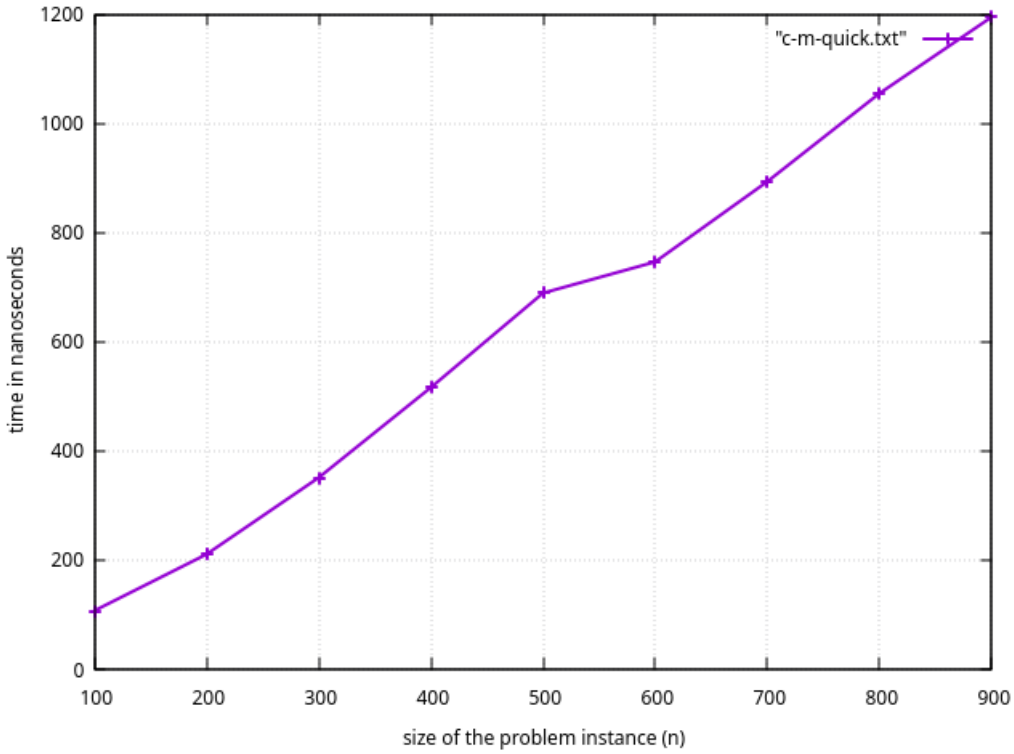

Figura 4.2: Tempo de execução médio do Quick Sort

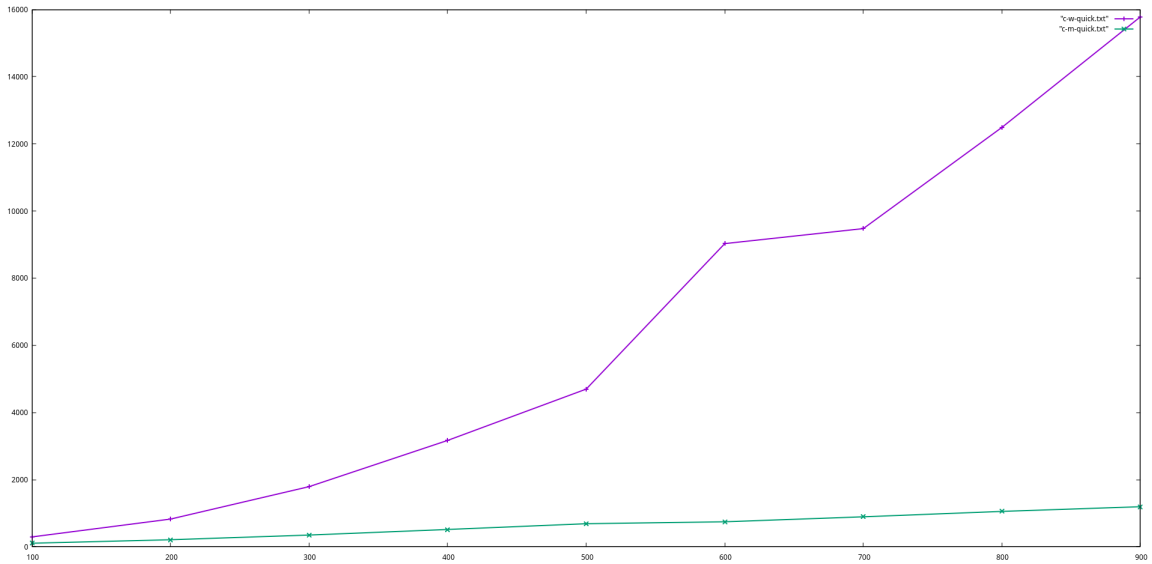

Figura 4.3: Tempo de execução em médio de comparação entre casos do quick sort

5. Distribution Sort

"If we can really understand the problem,
the answer will come out of it,
because the answer is not separate from the problem."
Jiddu Krishnamurti

5.1 Introdução

O Distribution Sort, também conhecido como Counting Sort, é um algoritmo de ordenação que se baseia no agrupamento dos elementos de acordo com seus dígitos ou posições. Ele utiliza a propriedade dos dígitos para realizar a ordenação de forma eficiente.

Algoritmo 5.1. **algoritmo** distribution-sort(v, n)
  $s \leftarrow min(v, n)$
  $b \leftarrow max(v, n)$
  para i de 1 para $(b - s + 1)$ **faça**
    $c[i] \leftarrow 0$
  fim para
  para i de 1 para n **faça**
    $c[v[i] - s + 1] \leftarrow c[v[i] - s + 1] + 1$
  fim para
  para i de 1 para $(b - s + 1)$ **faça**
    $c[i] \leftarrow c[i] + c[i - 1]$
  fim para
  para i de 1 para n **faça**
    $d \leftarrow v[i] - s + 1$
    $w[c[d]] \leftarrow v[i]$
    $c[d] \leftarrow c[d] - 1$
  fim para
  para i de 1 para n **faça**
    $v[i] \leftarrow w[i]$
  fim para

□

5.2 Conclusão

No gráfico abaixo é possível visualizar o tempo de execução dos casos anteriormente apresentados.

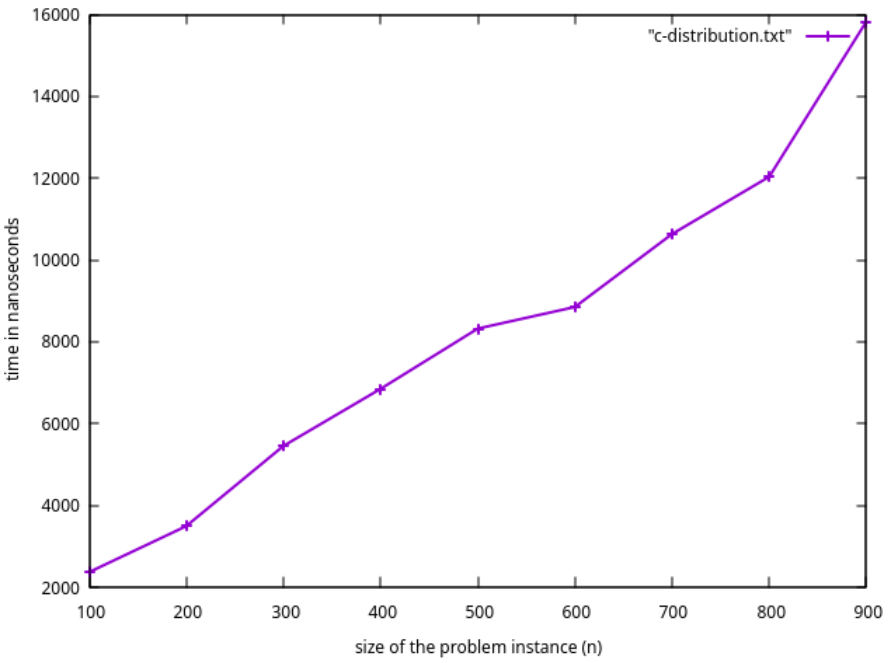

Figura 5.1: Tempo de execução médio do Distribution Sort

6. Conclusão

"If we can really understand the problem,
the answer will come out of it,
because the answer is not separate from the problem."
Jiddu Krishnamurti

6.1 Introdução

A fim de analisar e comparar o desempenho dos algoritmos de ordenação mencionados anteriormente - Selection Sort, Insertion Sort, Merge Sort, Quick Sort e Distribution Sort, é apresentado abaixo um gráfico que mostra a comparação do tempo de execução médio de todos eles.

O gráfico permite visualizar a relação entre o tamanho da lista de elementos a serem ordenados e o tempo necessário para completar o processo de ordenação. Os eixos vertical e horizontal representam, respectivamente, o tempo de execução em uma escala adequada e o tamanho da lista.

É importante ressaltar que o tempo de execução pode variar dependendo do ambiente de execução, da implementação do algoritmo e das características dos dados de entrada. Portanto, o gráfico fornecido tem como objetivo fornecer uma representação visual aproximada do desempenho dos algoritmos.

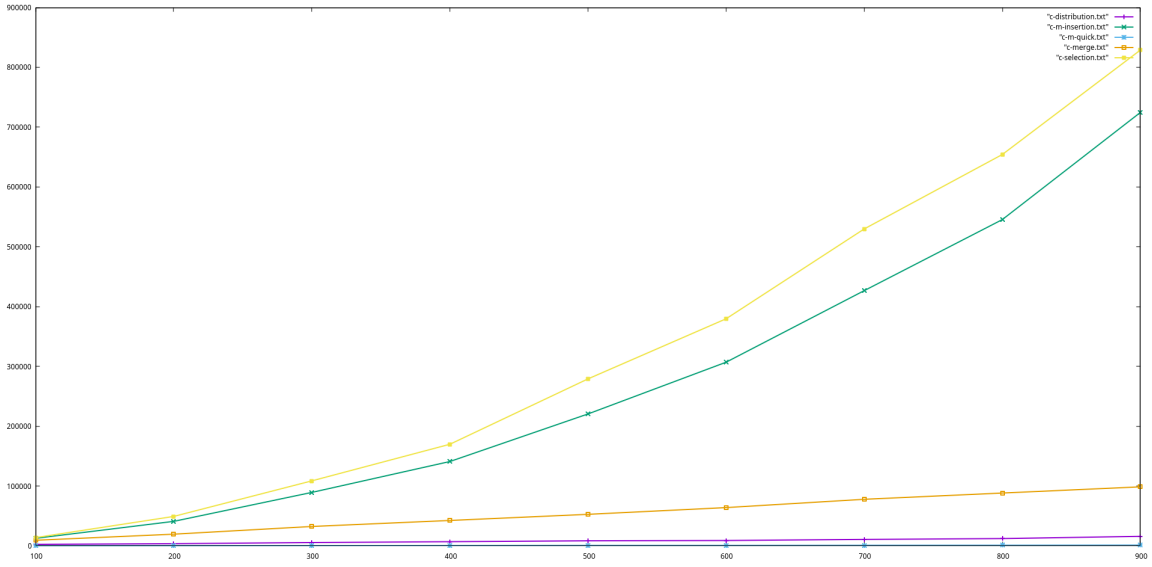

Figura 6.1: Comparação dos algorítmos de ordenação