**LUCAS TAVARES BATISTA**

**MURIEL SOUZA DA CRUZ**

**ANÁLISE DE ALGORITMOS**

**EXPERIMENTOS COM ALGORITMOS DE ORDENAÇÃO**

**PALMAS – TO**

**2019**

**LUCAS TAVARES BATISTA**

**MURIEL SOUZA DA CRUZ**

**ANÁLISE DE ALGORITMOS**

**EXPERIMENTOS COM ALGORITMOS DE ORDENAÇÃO**

**Trabalho elaborado e apresentado como requisito parcial na composição de grau, na disciplina de Análise de Algoritmos, turma 0704 – 2019/1, pelo Centro Universitário Luterano de Palmas (CEULP/ULBRA).**

**Profª. Responsável – Msc. Jack Gomes de Souza**

**PALMAS – TO**

**2019**

|  |
| --- |
| **TRABALHO**   * A tarefa está relacionada à implementação dos algoritmos de ordenação Bubble Sort, Insertion Sort e Selection Sort e à condução de experimentos com entradas de tamanhos diferentes, com o objetivo de analisar o comportamento dos algoritmos em relação a alguns fatores, como quantidade de comparações e trocas, uso de CPU e memória. Para conduzir os experimentos considere três conjuntos de dados (podem ter números gerados aleatoriamente) com tamanhos 100, 10.000 e 100.000 elementos. |

**METODOLOGIA**

Para a realização do presente trabalho utilizou-se como material um computador com processador Intel Core i5-2450M / frequência 2.50 GHz, memória RAM de 4GB DDR3 / frequência 665.1 MHz e um Hard Disk (HD) com 500GB de armazenamento.

Já para a execução dos códigos foram utilizadas a Linguagem Python 3.7 e seu ambiente de IDLE. Para verificação dos consumos de *Hardware* foi utilizado o Gerenciador de Tarefas/Monitor de Recursos fornecidos pelo Windows. A criação dos conjuntos de dados foi realizada através do site *mockaroo.com.*

A sequência metodológica utilizada foi a seguinte, primeiro escolheu-se um algoritmo. Após a escolha, ocorreu-se o teste com cada conjunto de dados e entre a execução de cada conjunto definiu-se que o processo *python* seria encerrado para execução do próximo conjunto de dados. Esse processo garante que resquícios de outra execução interfira no processamento de outro.

**RESULTADOS**

Conforme as tabelas a seguir é possível ver todos os resultados gerados no processamento dos conjuntos de dados, utilizando os algoritmos solicitados.

**Tabela 1** – Resultado dos Algoritmos para conjunto de CEM entradas.

|  |  |
| --- | --- |
| **Algoritmo** | **Resultados** |
| **Bubble Sort 1** | Quantidade de comparações com **100** entradas: 10100  Quantidade de trocas com **100** entradas: 2676  Tempo de execução em segundos com **100**entradas: 0.003269  Uso da CPU com **100** entradas: 2.9%  Uso da RAM com **100** entradas: 29.6MB |
| **Bubble Sort 2** | Quantidade de comparações com **100** entradas:  Quantidade de trocas com **100** entradas:  Tempo de execução em segundos com **100**entradas:  Uso da CPU com **100** entradas:  Uso da RAM com **100** entradas: |
| **Bubble Sort 3** | Quantidade de comparações com **100** entradas:  Quantidade de trocas com **100** entradas:  Tempo de execução em segundos com **100**entradas:  Uso da CPU com **100** entradas:  Uso da RAM com **100** entradas: |
| **Insertion Sort** | Quantidade de comparações com **100** entradas: 100  Quantidade de trocas com **100** entradas: 2676  Tempo de execução em segundos com **100**entradas: 4.894762  Uso da CPU com **100** entradas: 2.2%  Uso da RAM com **100** entradas: 27.5MB |
| **Selection Sort** | Quantidade de comparações com **100** entradas: 5050  Quantidade de trocas com **100** entradas: 305  Tempo de execução em segundos com **100**entradas: 0.012991  Uso da CPU com **100** entradas: 3.4%  Uso da RAM com **100** entradas: 23MB |

Na tabela 1 é possível visualizar todas as situações relacionadas ao cinco algoritmos em um conjunto de dados de cem entradas. Para o conjunto com cem entradas é possível verificar que o \*\*ALGORITMO NOME\*\* teve o menor número de trocas. Já em relação ao uso do processador, o \*\*ALGORITMO NOME\*\* foi o que usou o menor percentual de CPU enquanto o processo *python* executava o algoritmo. Em relação ao uso de memória RAM pelo processo que executa o código *python,* o \*\*ALGORITMO NOME\*\* foi o que usou menor quantidade de memória.

**Tabela 2** – Resultado dos Algoritmos para conjunto de DEZ MIL entradas.

|  |  |
| --- | --- |
| **Algoritmo** | **Resultados** |
| **Bubble Sort 1** | Quantidade de comparações com **10000** entradas: 999000  Quantidade de trocas com **10000** entradas: 213431  Tempo de execução em segundos com **10000**entradas: 0.344464  Uso da CPU com **10000** entradas: 6.1%  Uso da RAM com **10000** entradas: 51.1MB |
| **Bubble Sort 2** | Quantidade de comparações com **10000** entradas:  Quantidade de trocas com **10000** entradas:  Tempo de execução em segundos com **10000**entradas:  Uso da CPU com **10000** entradas:  Uso da RAM com **10000** entradas: |
| **Bubble Sort 3** | Quantidade de comparações com **10000** entradas:  Quantidade de trocas com **10000** entradas:  Tempo de execução em segundos com **10000**entradas:  Uso da CPU com **10000** entradas:  Uso da RAM com **10000** entradas: |
| **Insertion Sort** | Quantidade de comparações com **10000** entradas: 999  Quantidade de trocas com **10000** entradas: 213431  Tempo de execução em segundos com **10000**entradas: 21.104061  Uso da CPU com **10000** entradas: 2.6%  Uso da RAM com **10000** entradas: 62MB |
| **Selection Sort** | Quantidade de comparações com **10000** entradas: 499500  Quantidade de trocas com **10000** entradas: 5196  Tempo de execução em segundos com **10000**entradas: 0.101380  Uso da CPU com **10000** entradas: 3%  Uso da RAM com **10000** entradas: 22MB |

Na Tabela 2 é possível visualizar todas as situações relacionadas ao cinco algoritmos em um conjunto de dados de dez mil entradas. Para o conjunto com dez mil entradas é possível verificar que o \*\*ALGORITMO NOME\*\* teve o menor número de trocas. Já em relação ao uso do processador, o \*\*ALGORITMO NOME\*\* foi o que usou o menor percentual de CPU enquanto o processo *python* executava o algoritmo. Em relação ao uso de memória RAM pelo processo que executa o código *python,* o \*\*ALGORITMO NOME\*\* foi o que usou menor quantidade de memória.

**Tabela 3** – Resultado dos Algoritmos para conjunto de CEM MIL entradas.

|  |  |
| --- | --- |
| **Algoritmo** | **Resultados** |
| **Bubble Sort 1** | Quantidade de comparações com **100000** entradas: 81027002  Quantidade de trocas com **100000** entradas: 19879198  Tempo de execução em segundos com **100000**entradas: 21.711730  Uso da CPU com **100000** entradas: 32.7%  Uso da RAM com **100000** entradas: 23.8MB |
| **Bubble Sort 2** | Quantidade de comparações com **100000** entradas:  Quantidade de trocas com **100000** entradas:  Tempo de execução em segundos com **100000**entradas:  Uso da CPU com **100000** entradas:  Uso da RAM com **100000** entradas: |
| **Bubble Sort 3** | Quantidade de comparações com **100000** entradas:  Quantidade de trocas com **100000** entradas:  Tempo de execução em segundos com **100000**entradas:  Uso da CPU com **100000** entradas:  Uso da RAM com **100000** entradas: |
| **Insertion Sort** | Quantidade de comparações com **100000** entradas: 9001  Quantidade de trocas com **100000** entradas: 19879198  Tempo de execução em segundos com **100000**entradas: 150.783140  Uso da CPU com **100000** entradas: 32.6%  Uso da RAM com **100000** entradas: 67.4MB |
| **Selection Sort** | Quantidade de comparações com **100000** entradas: 40513501  Quantidade de trocas com **100000** entradas: 59315  Tempo de execução em segundos com **100000**entradas: 8.902644  Uso da CPU com **100000** entradas: 32%  Uso da RAM com **100000** entradas: 23MB |

Na tabela 1 é possível visualizar todas as situações relacionadas ao cinco algoritmos em um conjunto de dados de cem mil entradas. Para o conjunto com cem mil entradas é possível verificar que o \*\*ALGORITMO NOME\*\* teve o menor número de trocas. Já em relação ao uso do processador, o \*\*ALGORITMO NOME\*\* foi o que usou o menor percentual de CPU enquanto o processo *python* executava o algoritmo. Em relação ao uso de memória RAM pelo processo que executa o código *python,* o \*\*ALGORITMO NOME\*\* foi o que usou menor quantidade de memória.