Universidade Federal da Paraíba (UFPB)
Curso: Sistemas de Informação
Disciplina: Avaliação de Desempenho de Sistemas
Professor: Marcus Carvalho

Prática 3 – Intervalo de confiança

Nesta atividade de laboratório é apresentado um problema de intervalo de confiança, envolvendo a análise de desempenho de três algoritmos de ordenação de arrays de números inteiros. Você deve executar o programa fornecido e avaliar o desempenho do mesmo, a partir das métricas gravadas pelo programa. Para responder às questões, você deve escrever um relatório com as suas análises.

Deve ser enviado pelo Google Classroom o relatório, de preferência em um arquivo do Google Docs contendo as respostas das questões com as análises, além de um arquivo *ZIP* com os dados coletados na medição dos programas (arquivos *.txt* gerados). É **fortemente recomendado** o uso de gráficos nos relatórios, para a exibição dos dados coletados e para ajudar na sua análise. Tabelas também devem ser usadas para exibir dados quando necessário.

No início do relatório, descreva a configuração da máquina na qual você está executando cada análise de desempenho, com informações sobre a CPU, Memória, Cache, Disco, etc.

Problema

O programa consiste na implementação de três algoritmos de ordenação para arrays de números inteiros: *quick sort*, *merge sort* e *counting sort*. A ideia desta atividade é comparar o desempenho dos três algoritmos para diferentes situações de valores de entrada para ordenação. Para executar o programa, deve-se passar três parâmetros de entrada. A linha de execução esperada é:

```
java –cp bin MedidorDeOrdenacao <ordenador(QUICK, MERGE, COUNTING)> <tamanho-da-entrada> <valor-maximo>
```

Os parâmetros de entrada esperados são os seguintes:

- **Ordenador**: um dos três valores (QUICK, MERGE, ou COUNTING) que indica qual o algoritmo de ordenação será usado na medição.
- **Tamanho da entrada (T)**: tamanho do array de inteiros a ser ordenado.
- **Valor máximo (MAX)**: o valor máximo possível dos números inteiros que irão compor o array de entrada. Os valores do array serão gerados aleatoriamente, sendo gerados uma quantidade **T** de valores inteiros no intervalo [0, MAX].

Questões

1. Queremos comparar o desempenho de cada algoritmo de ordenação, medindo o tempo para ordenar arrays de inteiros de certos tamanhos e com diferentes faixas de possíveis valores inteiros.
    a. Realize 30 medições para cada algoritmo (quick, merge, counting) e com base nos dados obtidos calcule, para cada um, quantas vezes cada um deve ser executado (tamanho da amostra) para que se tenha um intervalo de confiança com uma margem de erro menor ou igual a 2%, para um nível de confiança de 95%. Use como parâmetros de entrada um tamanho de entrada de 9300000 (9,3 milhões) e o valor máximo da entrada 930000 (930 mil). Exemplo:

    ```
    java -cp bin MedidorDeOrdenacao quick 9300000 930000
    java -cp bin MedidorDeOrdenacao merge 9300000 930000
    java -cp bin MedidorDeOrdenacao counting 9300000 930000
    ```

    b. Use o maior valor dos três tamanhos de amostra (n) obtidos na questão anterior, execute as ordenações n vezes para cada algoritmo, com a mesma entrada anterior. Qual a média e o intervalo de confiança, com nível de confiança de 95%, do tempo de ordenação obtidos para cada algoritmo?

    c. Com base no intervalo de confiança da resposta anterior, podemos afirmar **estatisticamente** que desses três algoritmos existem uns que são melhores que os outros em relação ao tempo de ordenação? Qual seria o melhor e o pior algoritmo para esta entrada? Justifique.

2. Queremos analisar agora como o tempo de ordenação de cada algoritmo varia ao mudarmos a entrada.
    a. Verifique como o tempo de ordenação aumenta ao aumentarmos o tamanho da entrada, para cada algoritmo. Mantenha fixo o valor máximo em 930000 (930 mil) e varie o tamanho da entrada em dois valores: 9300000 (9,3 milhões) e 93000000 (93 milhões). Analise o padrão de aumento do tempo de ordenação de cada algoritmo e indique qual o melhor e o pior em cada cenário, usando a análise de intervalos de confiança. O tamanho da amostra deve ser suficiente para ter uma margem de erro menor ou igual a 2%.

    b. Verifique como o tempo de ordenação aumenta ao aumentarmos a faixa de valores de entrada possíveis (ou seja, aumentando valor máximo da entrada). Mantenha o tamanho da entrada fixo em 9300000 (9,3 milhões) e varie o valor máximo de entrada em dois valores: 930000 (930 mil) e 93000000 (93 milhões). Analise o padrão de aumento do tempo de ordenação de cada algoritmo e indique qual o melhor e o pior em cada cenário, usando a análise de intervalos de confiança. O tamanho da amostra deve ser suficiente para ter uma margem de erro menor ou igual a 2%.

3. Com base nas questões anteriores e em outras análises que você pode fazer, quais algoritmos são os mais adequados para diferentes situações de tamanho da entrada e do valor máximo? Justifique com base nas análises de intervalos de confiança.

Especificações da máquina:

Processador 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz
RAM instalada 16,0 GB (utilizável: 15,7 GB)

Respostas:

Questão 1, Letra A:

ALGORITMO	QUANT. DE EXECUÇÕES	MARGEM DE ERRO
Counting	30	2.56%
Merge	30	2.79%
Quick	30	1.49%

Quantidade ideal para obter uma margem de erro menor ou igual a 2%:

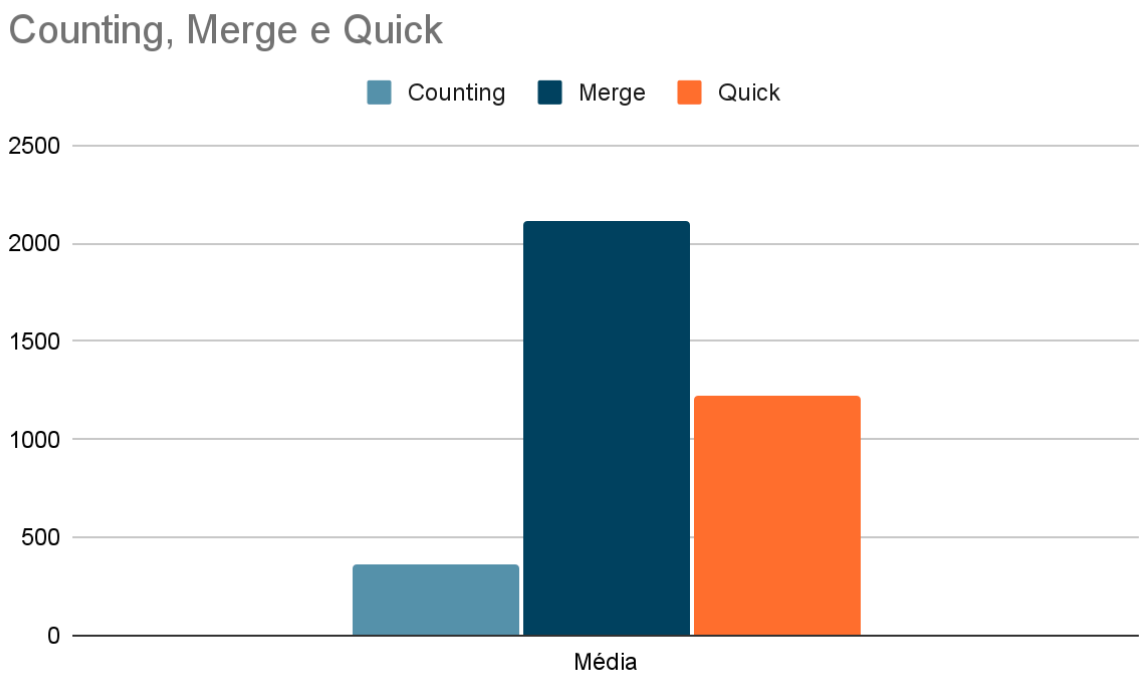
ALGORITMO	QUANT. DE EXECUÇÕES	MARGEM DE ERRO
Counting	100	2.03%
Merge	100	0.73%
Quick	30	1.49%

- **Com base nas medições, foi percebido que para atingir uma margem de erro menor ou igual a 2%, seria como foi supracitado.**

Questão 1, Letra B:

ALGORITMO	MÉDIA
Counting	**362.530**
Merge	**2,114.500**
Quick	**1,223.190**

Gráfico da Média:



Intervalo de confiança:

Algoritmo	**Média**	**Desvio padrão**	**Tamanho da amostra**	**Intervalo - erro**	**Intervalo - limite inferior**	**Intervalo - limite superior**	**Margem de erro**	**Tamanho da amostra para margem 5%**
Counting	362.530	37.472	100.000	7.344	355.186	369.874	2.03%	16.41739215
Merge	2,114.500	79.284	100.000	15.539	2,098.961	2,130.039	0.73%	2.160346706
Quick	1,223.190	157.425	100.000	30.855	1,192.335	1,254.045	2.52%	25.45272832

Questão 1, Letra C:

Com base no intervalo de confiança, pode-se afirmar que:

Counting é o mais eficiente para a entrada dada, pois tem o menor tempo médio de execução (362.530) e o intervalo de confiança mais estreito. Já o Merge é o menos eficiente para a entrada dada, pois tem o maior tempo médio (2,114.500) de execução e o intervalo de confiança mais largo. Por fim, o algoritmo Quick tem uma eficiência intermediária, com tempo médio de execução e intervalo de confiança entre o Counting e o Merge (1,223.190). Sendo assim, podemos afirmar estatisticamente que, para esta entrada, o Counting Sort é o melhor algoritmo em relação ao tempo de ordenação, enquanto o Merge Sort é o pior.

Questão 2, Letra A:

Gráfico com entrada de 9300000:

	Taxa de entrada	9,3 milhões	93 milhões
Counting	Média	362.530	2,730.470
Merge	Média	2,114.500	15,273.110
Quick	Média	1,223.190	12,911.640

Gráfico da entrada de 9,3 milhões:

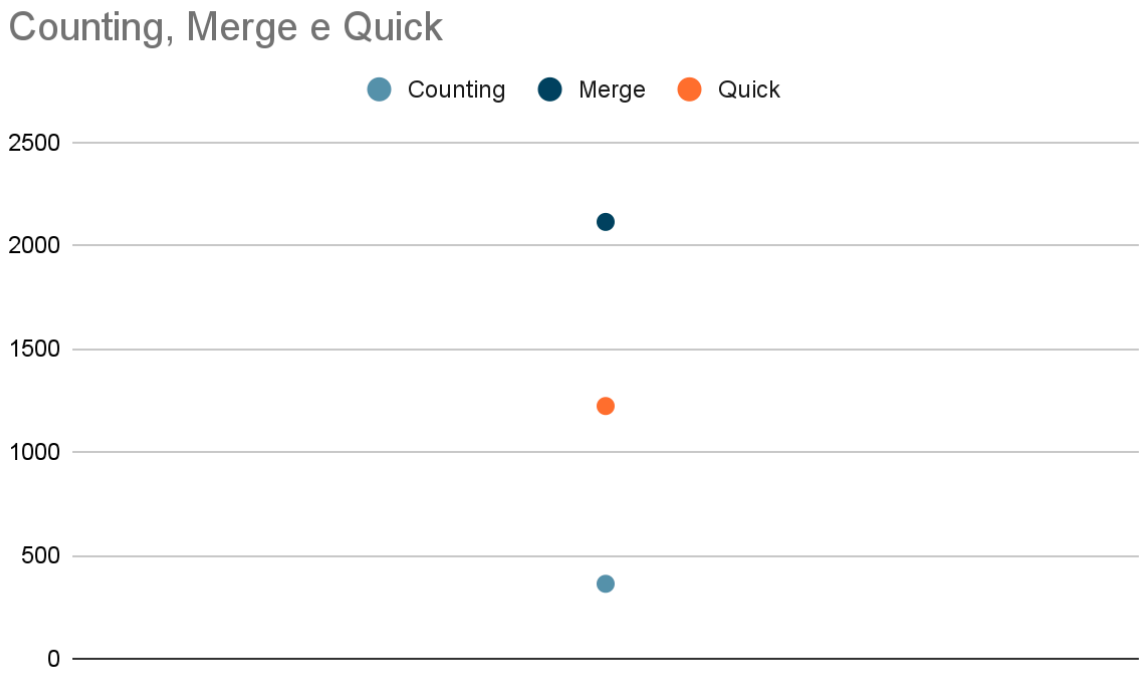
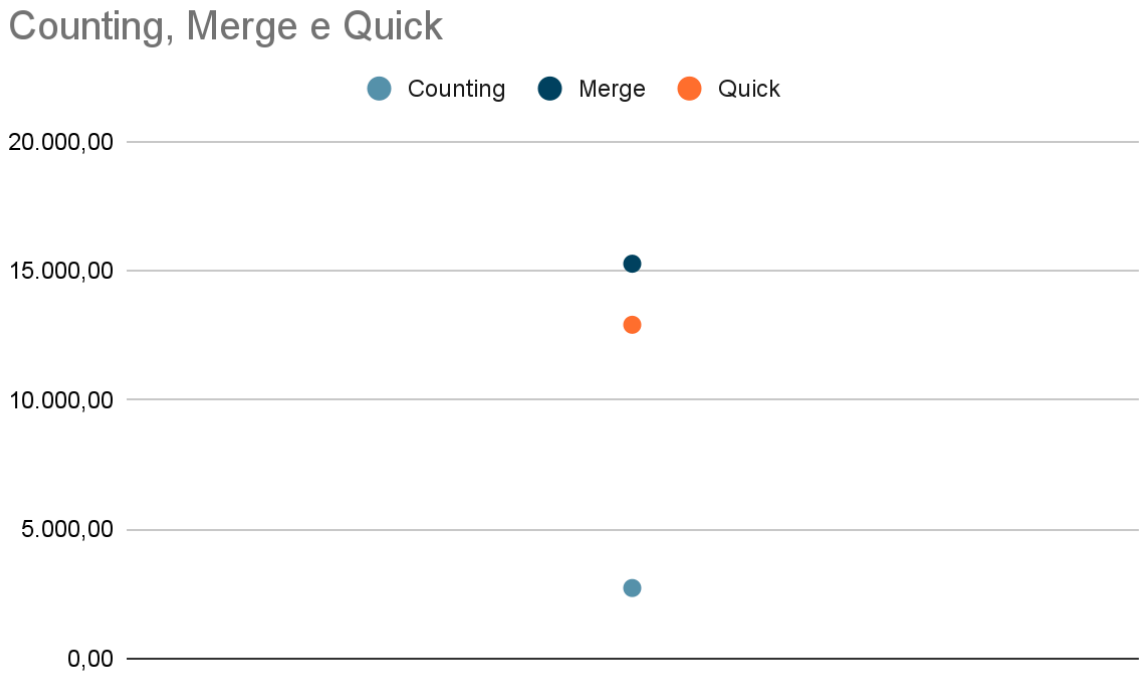


Gráfico da entrada de 93 milhões:



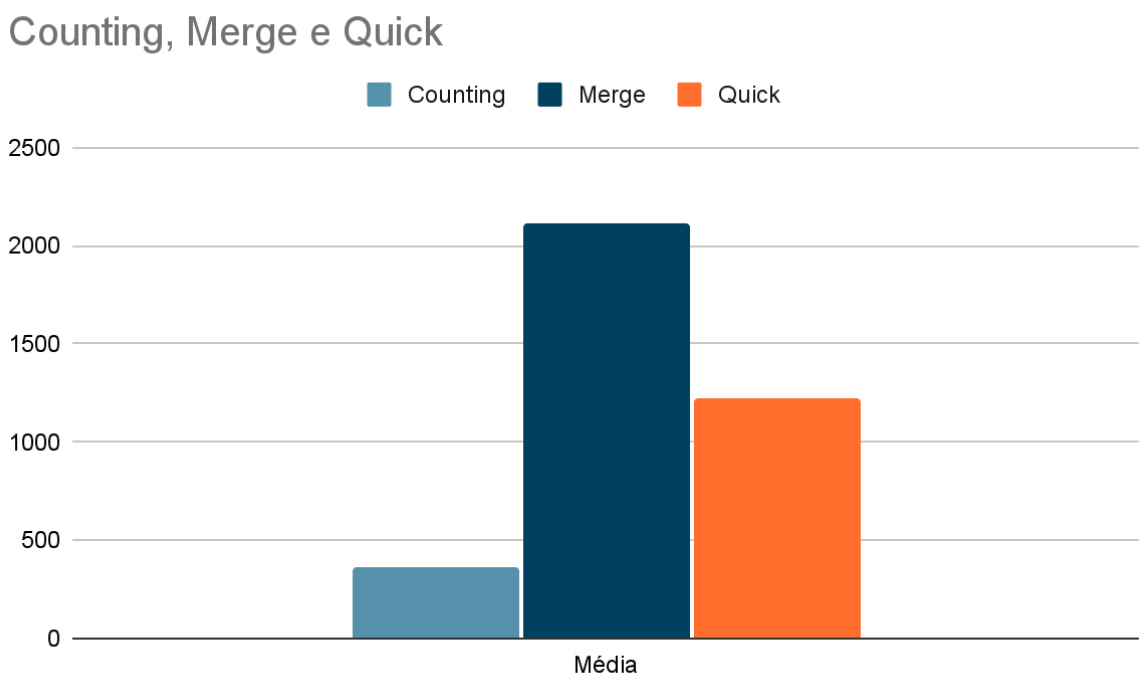
Com base nos dados e gráficos, podemos concluir que o Counting Sort é o algoritmo mais eficiente em termos de aumento de tempo de ordenação com o aumento do tamanho da entrada, enquanto Quick Sort é o menos eficiente.

Questão 2, Letra B:

Valor máximo da entrada sendo 930 mil:

ALGORITMO	MÉDIA
Counting	362.530
Merge	2,114.500
Quick	1,223.190

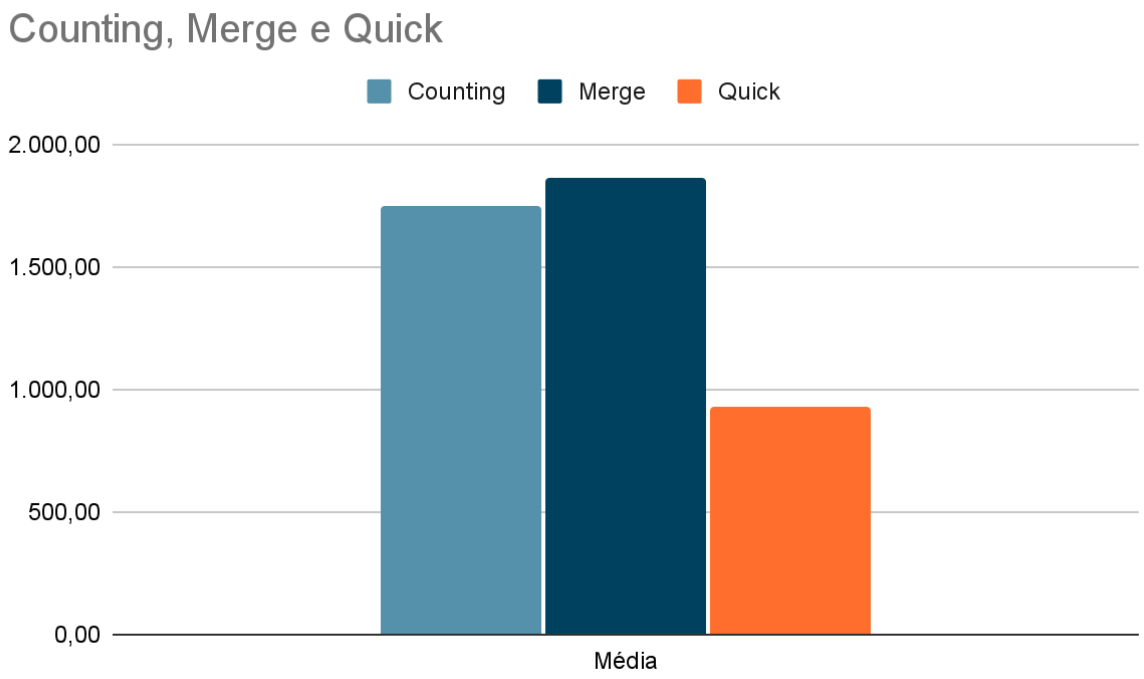
Gráfico:



Valor máximo da entrada sendo 93 milhões:

ALGORITMO	MÉDIA
Counting	1,748.700
Merge	1,866.480
Quick	931.330

Gráfico:



Com o valor máximo da entrada de 930 mil, percebe-se que o melhor algoritmo é o Counting Sort, sendo o pior o Merge. Analogamente, usando o valor máximo de entrada em 93 milhões, o cenário muda: O Quick, que antes estava atrás do Counting agora vem sendo o melhor e o Merge, o pior. Rodando os algoritmos 100 vezes cada, percebe-se que só o Quick(0,92%) e Counting(1,61%) estavam dentro da margem de erro esperada, que era igual ou menor a 2%.

Questão 3:

Com base nos gráficos, podemos afirmar que o Counting é o melhor algoritmo em comparação com os outros algoritmos na maioria dos casos. Quando se tem um valor fixo de entrada muito alto, nesse caso o quick é superior. Analisando, pode-se ver também que o algoritmo Quick é melhor que o Merge em quase todos os casos, exceto quando se tem uma taxa de entrada muito alta.