relatório tabalho 1 de alg2

Matheus Ferreira Marquesini
Departamento de Informática
Universidade Federal do Paraná – UFPR
Curitiba, Brasil
matheus.marquesini@ufpr.br

I. Introdução

Esse relatório descreve uma implementação de código baseada em testes, destinada a testar algoritmos de classificação vetorial recursiva ((Merge Sort, quick Sort, heap Sort, Counting Sort, Tim Sort) implementados em linguagem C. O relatório está dividido em níveis. Primeiramente, tem-se o preenchimento dos vetores com valores inteiros aleatórios para a ordenação de vetores.

II. Iniciação dos vetores

Para a iniciação dos vetores para a realização dos testes, eu aloquei um vetor, que antes de cada chamada da função, é prenchidos por numeros inteiros aleatorios atraves da função rand() da biblioteca stdlib.h, onde esses valores aleatorios vao de 0 ate o tamanho do vetor.

III. Algoritmos de ordenação

A ordenação de vetores foi feita seguindo a seguinte ordem: Merge Sort, quick Sort, heap Sort, Tim Sort e Counting Sort. Esses algoritmos têm como entrada um vetor de inteiros indexado por [a..b], onde $b \leq$, e o tamanho do vetor é definido por n = b - a + 1. Portanto, todos os algoritmos têm como saída o vetor ordenado na forma crescente.m relação ao custo dos algoritmos de ordenação, a tabela - 1 (custo), retrata o custo dos algoritmos acima mencionados:

Tabela I
Tabela de relação de custo de cada algoritmo.

	Tempo de custo	
Algoritmo	Melhor caso	Pior caso
Merge Sort	$C(n * \log_2 n)$	$C(n + \log_2 n)$
Quick Sort	$C(n * log_2 n)$	$C(n^2/2)$
Heap Sort	$C(n * \log_2 n)$	$C(n + \log_2(n))$
Tim Sort	$C(n)$	$C(n + \log_2(n)$

Tendo em vista o custo de cada função sendo C, o custo em função de n, e n o tamanho do vetor, a tabela acima demostra as fórmulas para o cálculo do número de comparações entre elementos do vetor de cada algoritmo.

IV. Experimentos

A imagem - 1 primeiro teste, mostra os algoritmos executados em sequencia, com vetores de 20 posições, para fins de demostração de funcionalidade.

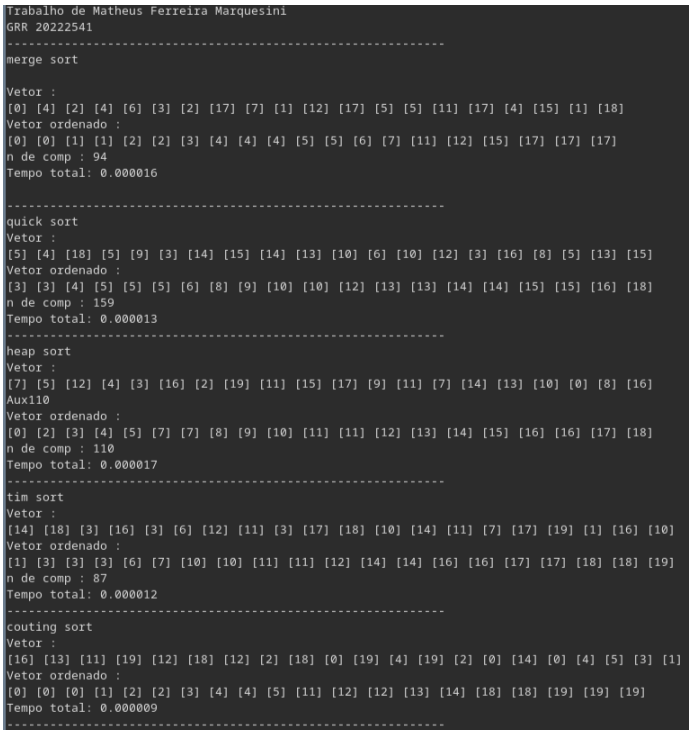


Imagem 1 - primeiro teste

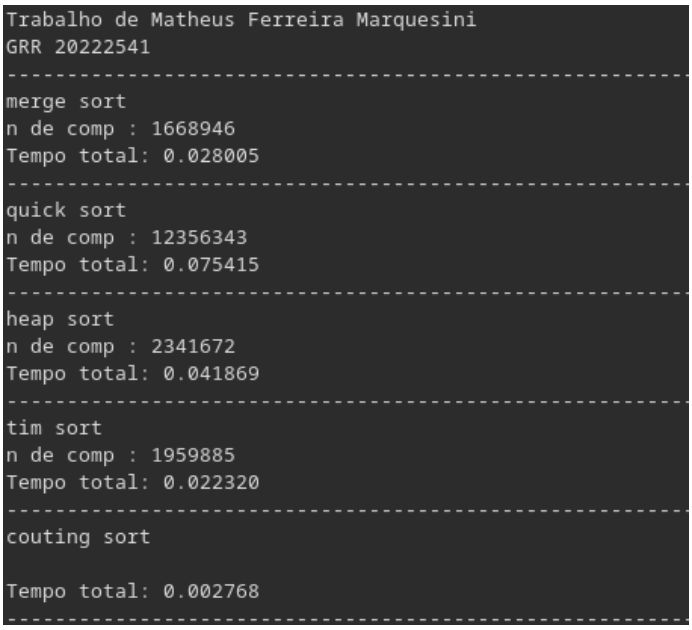


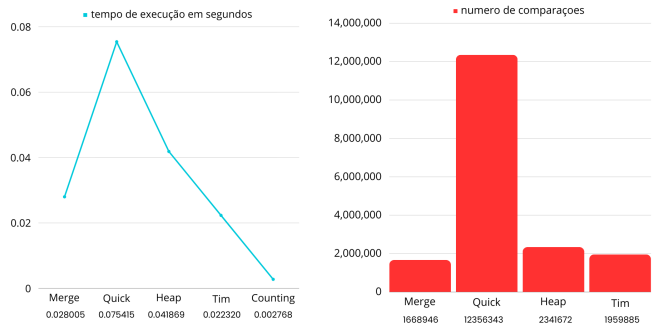
Imagem 2 - segundo teste

Tabela II
TEMPOS DE EXECUÇÃO PARA OS ALGORITMOS DE TAMANHO 100 MIL (EM ORDEM CRESCENTE)

Algoritmo de Ordenação	Tempo (s)
Counting Sort	0.002768
Tim Sort	0.022320
Merge Sort	0.028005
Heap Sort	0.041869
Quick Sort	0.075415

a imagem 2 e tabela 2, mostram os testes para numero de comparações e tempo de execução de cada algoritmo que foram citados anteriormente, para um vetor com indice de 100 mil de posiçoes e numeros aleatorios de 1 até o tamanho.

Grafico 1 - graficos de tempo de execução e numero de comparações (sem o counting sort)



Devido a relação do gráfico de barras estar em função do número de comparações, o algoritmo do Quick Sort, teve o número de comparações e tempo de execução muito elevado em comparação aos outros algoritimos, com quase 10 milhões de diferença para o segundo.

```
mfm22@atlas:~/Desktop/trabs-alg/trab_theus$ ./trab
Trabalho de Matheus Ferreira Marquesini
GRR 20222541
------------------------------------------------------------
merge sort
n de comp : 19951445
Tempo total: 0.322119
------------------------------------------------------------
quick sort
n de comp : 39072687
Tempo total: 0.267589
------------------------------------------------------------
heap sort
n de comp : 28424351
Tempo total: 0.507455
------------------------------------------------------------
tim sort
n de comp : 22658426
Tempo total: 0.258339
------------------------------------------------------------
couting sort

Tempo total: 0.095821
------------------------------------------------------------
mfm22@atlas:~/Desktop/trabs-alg/trab_theus$
```

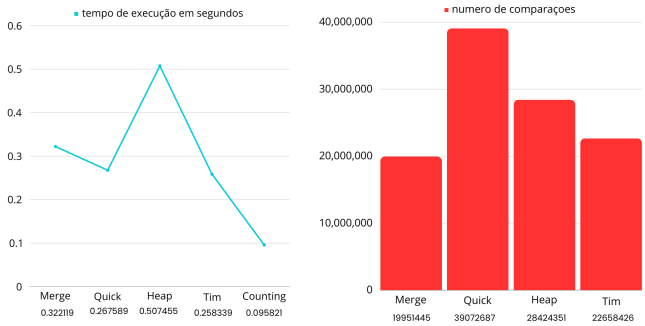
Imagem 3 - terceiro teste

Tabela III
TEMPOS DE EXECUÇÃO PARA OS ALGORITMOS DE TAMANHO 1 MILHÃO (EM ORDEM CRESCENTE)

Algoritmo de Ordenação	Tempo (s)
Counting Sort	0.095821
Tim Sort	0.258339
Quick Sort	0.267589
Merge Sort	0.322119
Heap Sort	0.507455

a imagem 3 e tabela 3, mostram os testes para numero de comparações e tempo de execução de cada algoritmo que foram citados anteriormente, para um vetor com indice de 1 milhão de posiçoes e numeros aleatorios de 1 até o tamanho.

Grafico 2 - graficos de tempo de execução e numero de comparações



Com base nos resultados do Grafico - 1, agora com 900 mil de tamanho a mais, o quick sort deu uma equilibrada a mais, apesar de ainda ser o com mais numeros de comparações e tempo de execução.

V. CONSIDERAÇÕES FINAIS

Considerando os esperimentos feitos acima, é importante destacar que, para cada algoritmo de ordenação de vetores, assumiu-se que o vetor estava preenchido com valores inteiros distintos. No que diz respeito à classificação dos vetores usando esses algoritmos, eles permaneceram dentro de um intervalo de custo esperado em termos de tempo de execução, bem melhores alguns dos testados no primeiro trabalho, especialmente o selection sort, que deixava os outros tempos de execução e comaprações, insignificantes. Uma propriedade para o algoritimo escolhido de custo interessante é a eficiência do TimSort em reduzir o número de comparações e trocas necessárias em arrays parcialmente ordenados.

REFERÊNCIAS

slides prod. Gregio Heap sort
slides prod. Gregio Quick sort
Counting sort
Tim sort
slides prod. Gregio Merge sort