Relatório do TDE3

Gabriel Vidal Andreoli

[1]Escola Politécnica – Pontificia Universidade Católica do Paraná (PUCPR)

gabrielvidalandreoli@gmail.com

Abstract. *I dont really know what i should write here, i guess hello is apropriate, hello*

Resumo. *Este documento serve para a explicação do meu TDE3 da disciplina de Resolução de problemas estruturados em computação*

1. Inicio

Nesse trabalho escolhi o Bubble Sort do grupo A por já saber o seu funcionamento e já ter implementado ele anteriormente nas primeiras aulas de Resolução de problemas estruturados em computação. Do grupo B eu escolhi o Merge Sort por já ter uma ideia de como ele funcionava e o Shell Sort por ter considerado o seu funcionamento fácil.

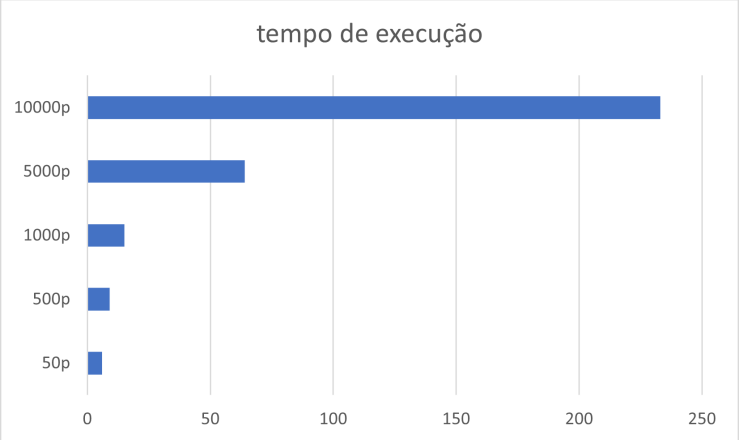
Em meu GitHub se encontra a pasta src onde está o código fonte do meu trabalho, o código está dividido em dois pacotes, o pacote 'A' onde está o código do bubble sort e seus testes e o pacote 'B' onde está o código tanto do Merge sort quanto do Shell sort e seus respectivos testes.

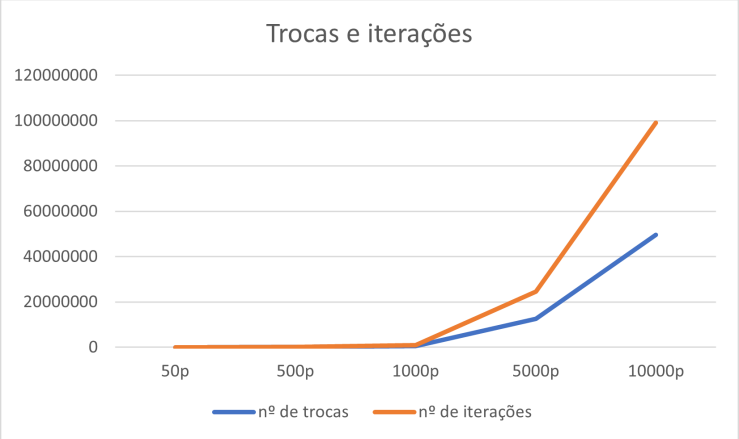
2. Bubble Sort

O Bubble Sort funciona da seguinte maneira, há um laço de repetição responsável por fazer o programa rodar uma quantidade x de vezes e outro que vai escaneando o vetor e realizando as comparações e quando necessário realiza as trocas, dentro do primeiro laço de repetição há também uma variável do tipo booleano que está por padrão como false e toda vez que alguma troca é realizada ele recebe como valor True, caso não seja realizada nenhuma troca um if irá parar a execução do programa.

Veja os resultados obtidos com a execução desse programa:

BubbleSort				
	Tamanho do vetor	tempo de execução	nº de trocas	nº de iterações
	50p	6ms	1343	2293
	500p	9ms	126434	237623
	1000p	15ms	500720	964632
	5000p	64ms	12496419	24674064
	10000p	233ms	49691051	99066092



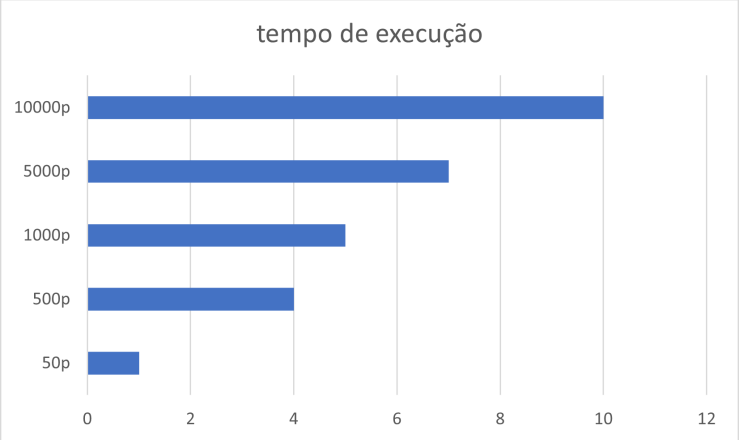


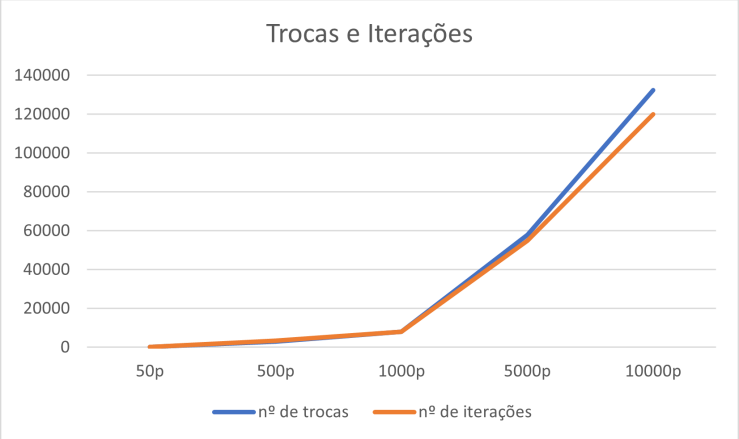
3. Shell Sort

O Shell Sort funciona de forma muito similar ao insertion sort, porém o espaçamento entre as comparações mudam a cada iteração do programa. Nesse caso o Shell Sort foi implementado da seguinte maneira: É pego o tamanho do vetor que vai passar pela ordenação e se pega o valor da metade do tamanho, esse valor vai servir como espaçamento, então o programa vai utilizar isso para ir comparando o índice n com o índice n- espaçamento, toda vez que necessário é realizado uma troca e após isso o espaçamento é reduzido pela metade, assim tendo certeza de que todos os índices serão comparados eventualmente.

Veja os resultados obtidos com a execução desse programa:

ShellSort				
	Tamanho do vetor	tempo de execução	nº de trocas	nº de iterações
	50p	1ms	154	208
	500p	4ms	2936	3514
	1000p	5ms	7901	8015
	5000p	7ms	57815	55017
	10000p	10ms	132367	120018



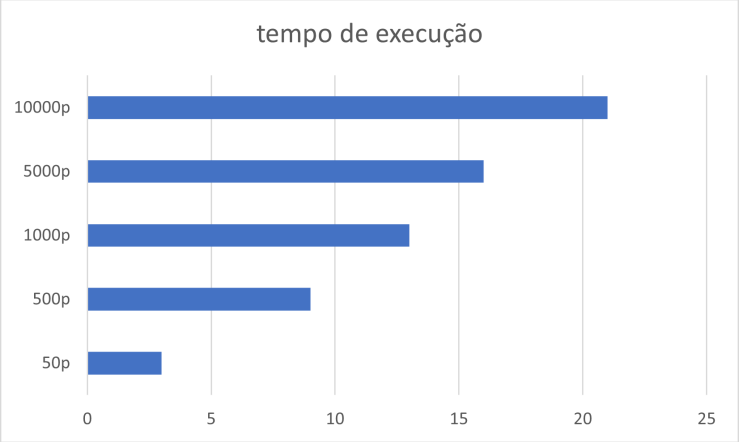


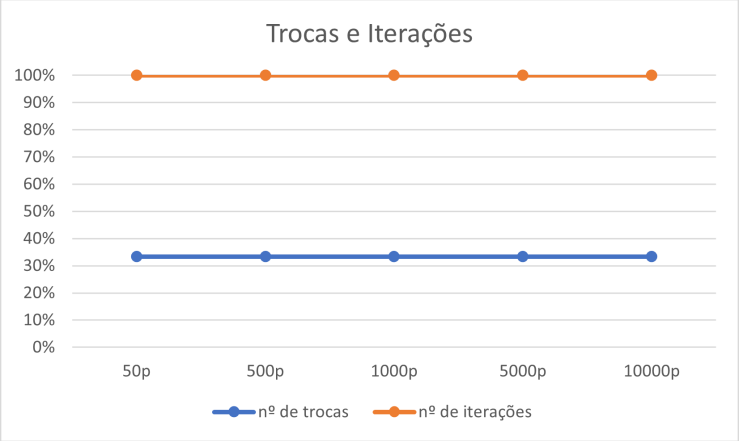
4. Merge Sort

O Merge Sort consiste em pegar o vetor e ir quebrando-o em subvetores cada vez menores, quando todos os vetores possuírem apenas 1 de tamanho é feita uma comparação e então eles começam a ser juntados de forma ordenada onde o menor valor fica à esquerda, e de forma repetitiva todos os subvetores são ordenados e juntados. A implementação desse código se dá da seguinte forma, há uma função recursiva com o nome sort que vai servir para chegar a segmentos cada vez mais específicos do vetor, então quando não tiver mais como segmentar o vetor é chamada a função merge que vai criando vários vetores com os valores do vetor original e então eles são comparados para serem inseridos no vetor original (esse vetor original não necessariamente é o vetor inserido como parâmetro, e sim o vetor que foi informado na chamada recursiva da função sort), isso é feito diversas vezes até que todas as chamadas recursivas já tenham sido realizadas e consequentemente ordenando o vetor.

Veja os resultados obtidos com a execução desse programa:

MergeSort				
	Tamanho do vetor	tempo de execução	nº de trocas	nº de iterações
	50p	3ms	286	572
	500p	9ms	4488	8976
	1000p	13ms	9976	19952
	5000p	16ms	61808	123616
	10000p	21ms	133616	267232





5. Conclusões

Com a realização desse trabalho, foi possível concluir que de todos os algoritmos de ordenação utilizados o mais lento é o bubble sort, ele possui um número de iterações e trocas muito acima dos outros algoritmos, além do seu tempo de execução também ser maior que o dos outros. Isso provavelmente se deve ao fato de que todos os índices são percorridos e comparados, resultando em uma grande perda de performance.

O Shell Sort e o Merge Sort são consideravelmente mais eficientes que o Buble Sort e pelos resultados obtidos o Shell Sort é o algoritmo mais eficiente dos 3. Ele possui um tempo de execução menor que o do Merge Sort e também realiza menos iterações, o que provavelmente influência nessa performance melhor.

Algo que também vale citar é que o Shell Sort sempre possui o mesmo número de iterações para cada tamanho de vetor, isso se deve a forma com que o código é estruturado, pois é sempre o (tamanho total / 2) e a cada repetição esse valor também é dividido por 2 até que esse valor seja 1. De forma resumida, o número de iterações do Shell Sort é sempre constante.

Os valores do número de trocas e iterações do Merge Sort também se provaram constantes ao longo dos testes, isso provavelmente se deve ao fato de que para cada tamanho de vetor, sempre irá ser criada a mesma quantidade de subvetores.