CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS

LAED II

Relatório III

AUTOR:
Pedro Henrique Maia Duarte

Professor: THIAGO DE SOUZA RODRIGUES

Prática 4 – Implementação do TAD Heaps

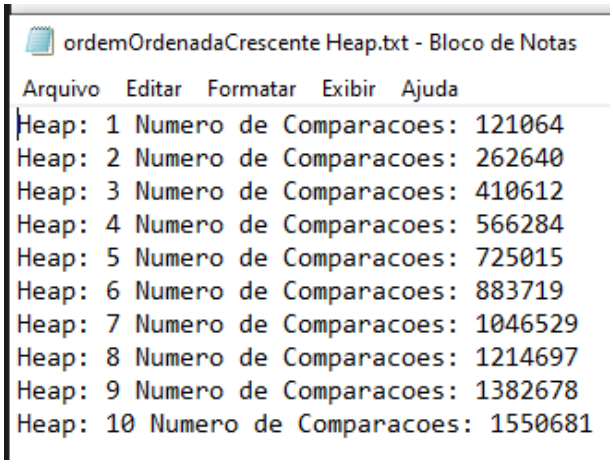
O **heap** é uma estrutura de dados, baseada em árvore, que é essencialmente uma árvore quase completa que satisfaz a propriedade heap: Se P é um nó pai de C, então a chave, o valor de P é maior que ou igual a (em uma heap máxima) ou menor que ou igual a (em uma heap mínima) chave de C. O nó no "topo" da heap (sem pais) é chamado de nó raiz. Em uma heap, o elemento de prioridade mais alta (ou mais baixa) é sempre armazenado na raiz.

O heap é uma implementação maximamente eficiente de um tipo de dados abstrato chamado de fila de prioridade e, de fato, as filas de prioridade são geralmente chamadas de "heaps", independentemente de como elas podem ser implementadas. Um heap é uma estrutura de dados útil quando é necessário remover repetidamente o objeto com a prioridade mais alta (ou mais baixa). Além disso, temos o o algoritmo Heapsort, que se consiste em ordenamento por seleção, com complexidade de O(n log(n)).

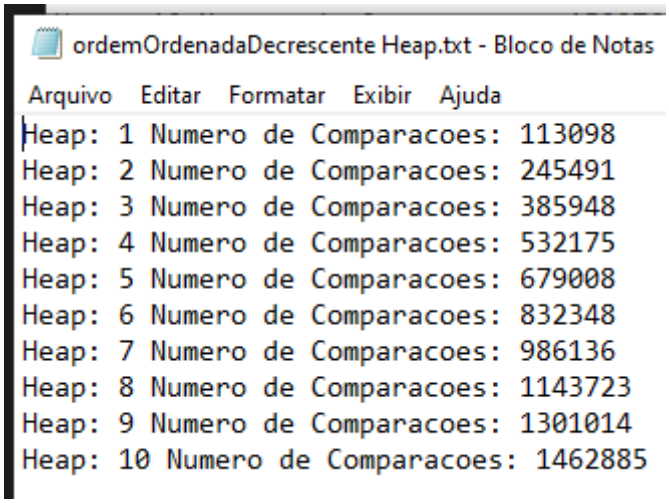
Sendo assim, implementaremos em linguagem Java, o algoritmo Heapsort, para ordenar um vetor de itens por inserção ordenada crescente, decrescente e inserção aleatória, com o intuito de analisar a quantidade de comparações realizadas em um heap.

temos os seguintes resultados, onde o número de elementos de cada árvore é dado pela seguinte forma:

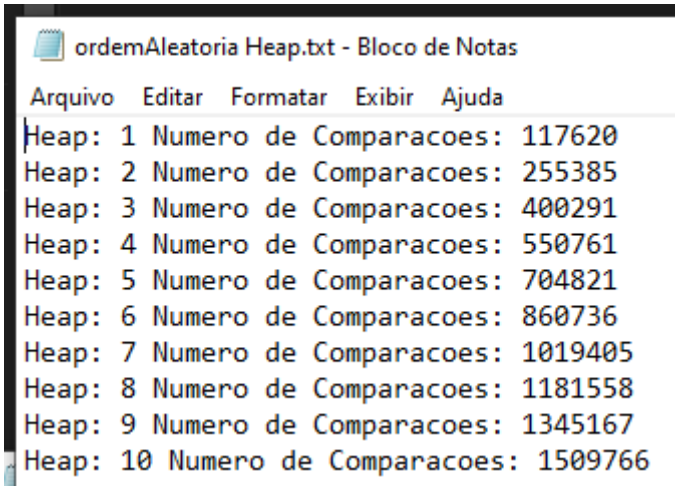
Heap:	1	2	3	4	5	6	7	8	9	
Quantidade de elementos:	10000	20000	30000	40000	50000	60000	70000	80000	90000	



Dados obtidos por inserção ordenada crescente em um Heap



Dados obtidos por inserção ordenada decrescente em um Heap



Dados obtidos por inserção aleatória em um Heap

Quando comparamos os elementos em um heap, esperamos que o número de comparações no pior caso, seja n-1 comparações, sendo n o número de elementos de um heap. Sendo assim, para testar o número de comparações realizadas, iremos utilizar o algoritmo Heapsort, para ordenar o heap.

Ademais, pretendemos analisar quantas comparações de elementos são realizadas em cada heap, e esperamos que a complexidade de comparações seja O(n log(n)), que é a complexidade do método Heapsort.

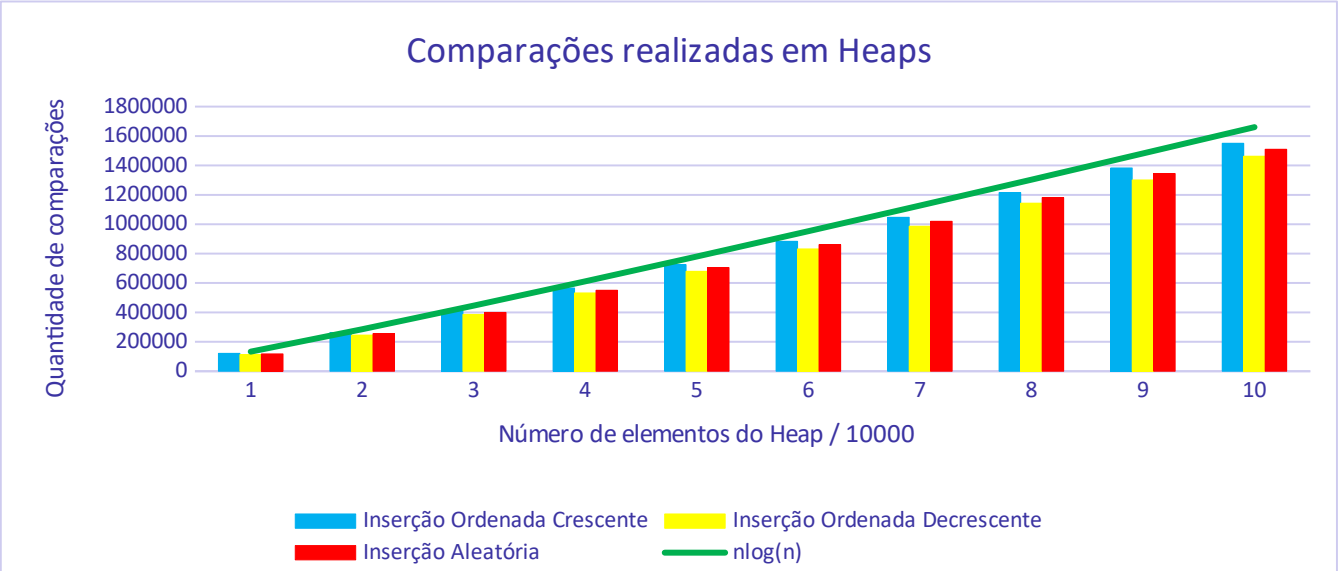
Sendo assim temos:

Heap	Inserção			Elementos no Heap (n)	nlog(n)
	Ordenada Crescente	Ordenada Decrescente	Aleatória		
	Número de Comparações	Número de Comparações	Número de Comparações		
1	121064	113098	117620	10000	132877,1238
2	262640	245491	255385	20000	285754,2476
3	410612	385948	400291	30000	446180,2464
4	566284	532175	550761	40000	611508,4952
5	725015	679008	704821	50000	780482,0237
6	883719	832348	860736	60000	952360,4928
7	1046529	986136	1019405	70000	1126654,711
8	1214697	1143723	1181558	80000	1303016,99
9	1382678	1301014	1345167	90000	1481187,364
10	1550681	1462885	1509766	100000	1660964,047

Onde, n por representar o número de elementos contidos em um heap, n será igual a:

$$n = Número\ do\ Heap * 10000.$$

Logo, temos o seguinte gráfico:



A partir da análise do gráfico, concluímos que o gráfico apresentou os resultados esperados, garantindo que o número de comparações realizadas, para um determinado número de elementos ‘n’, foi próximo de n log(n), como demonstrado pela linha verde do gráfico.

Sendo assim, temos que: $\forall\, n$, sendo n o número de elementos contidos em um heap, a complexidade de comparações realizadas é O(n log (n)) em Heaps.