UNIVERSIDADE FEDERAL DO CEARÁ

Nome: José Robertty e Maria Tassiane
Disciplina: Estrutura de Dados Avançada
Professor: Fábio Dias

Este presente trabalho foi pensado com o intuito da visualização dos tempos de execução e da visualização da complexidade dos diferentes algoritmos utilizados para a implementação da Lista de Prioridades.

Especificações da Máquina:

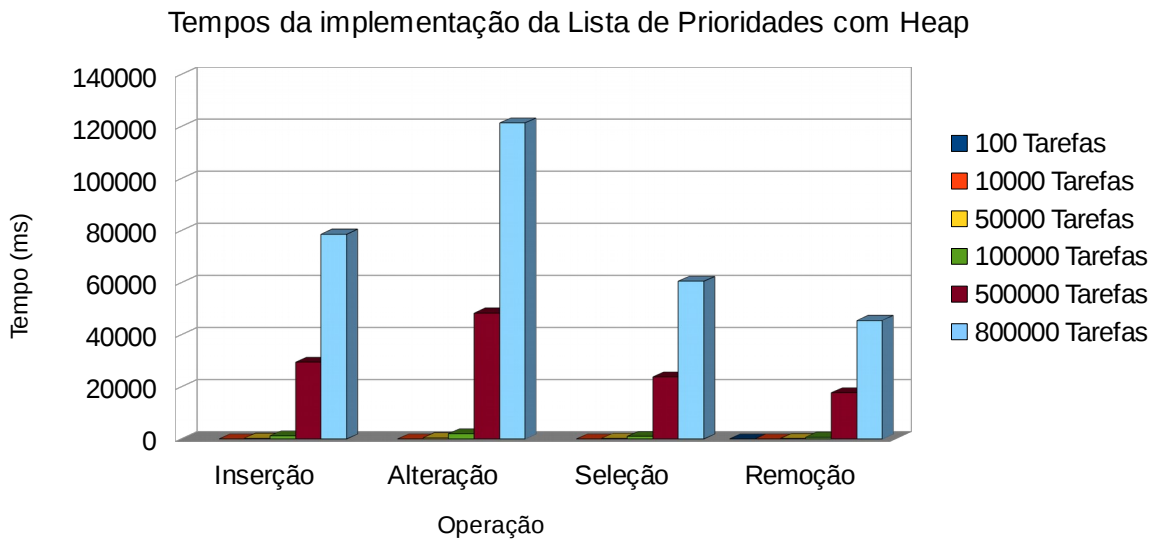
- Sistema operacional: Ubuntu 15.10
- Processador: Intel core i5-4210U 1.70GHz
- Memória: 7,7 GiB
- Tipo de sistema: 64-bit

Ao testarmos os algoritmos, temos os seguintes resultados para o Heap implementado para Lista de Prioridades, temos os tempos (em milissegundos) obtidos:

Operação/ Tarefas	**100**	**10000**	**50000**	**100000**	**500000**	**800000**	**TOTAL**
I	0	33	324	1262	29478	78678	109775
A	0	33	491	2014	48434	121384	172356
S	0	20	250	1003	23867	60577	85717
R	1	18	196	748	17830	45572	64365
TOTAL	1	104	1261	5027	119609	306211	432213

Tabela 1 – Tempos da implementação da Lista de Prioridades com Heap (I = Inserção, A = Alteração, S = Selecionar, R = Remover)

Gráfico 1



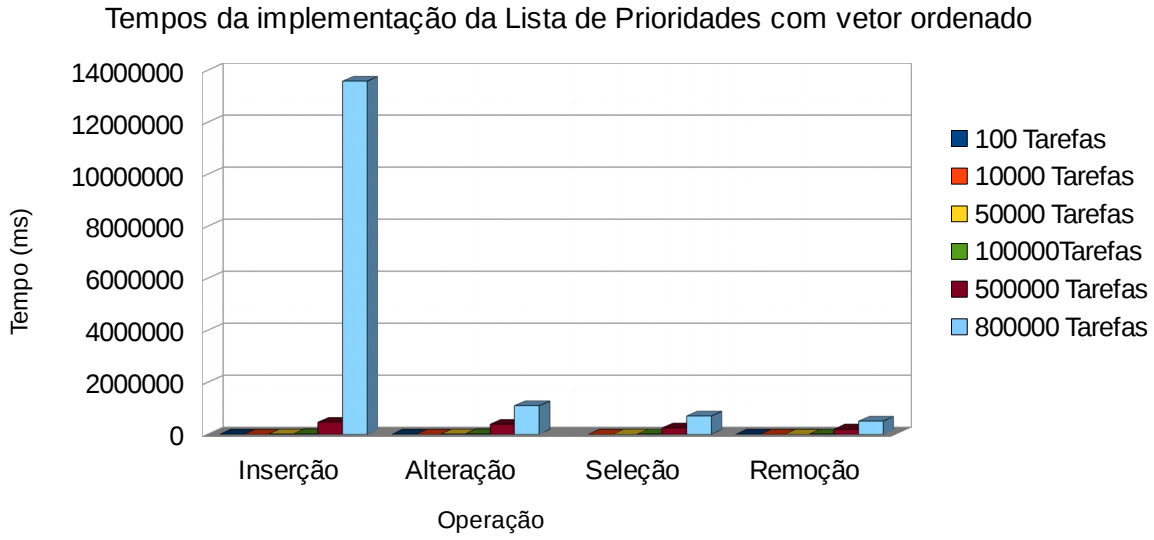
Na tabela 1 e no Gráfico 1, vemos que a estrutura de dados Heap gasta muito tempo nas tarefas onde se utiliza mais alteração e gasta pouco tempo nas tarefas onde se utiliza mais remoção e as tarefas que se utiliza mais seleção.

Agora vamos analisar a tabela da Lista de Prioridades implementada em um vetor ordenado, os tempos (em milissegundos) obtidos foram:

Operação/ Tarefas	**100**	**10000**	**50000**	**100000**	**500000**	**800000**	**TOTAL**
I	1	191	4547	18415	459380	13583305	14065839
A	1	154	3856	15359	383733	1098336	1501439
S	0	100	2442	9853	248537	707099	968031
R	1	74	1795	7215	193889	509070	712044
TOTAL	3	519	12640	50842	1285539	15897810	17247353

Tabela 2 – Tempos da implementação da Lista de Prioridades com vetor ordenado (I = Inserção, A = Alteração, S = Selecionar, R = Remover)

Gráfico 2



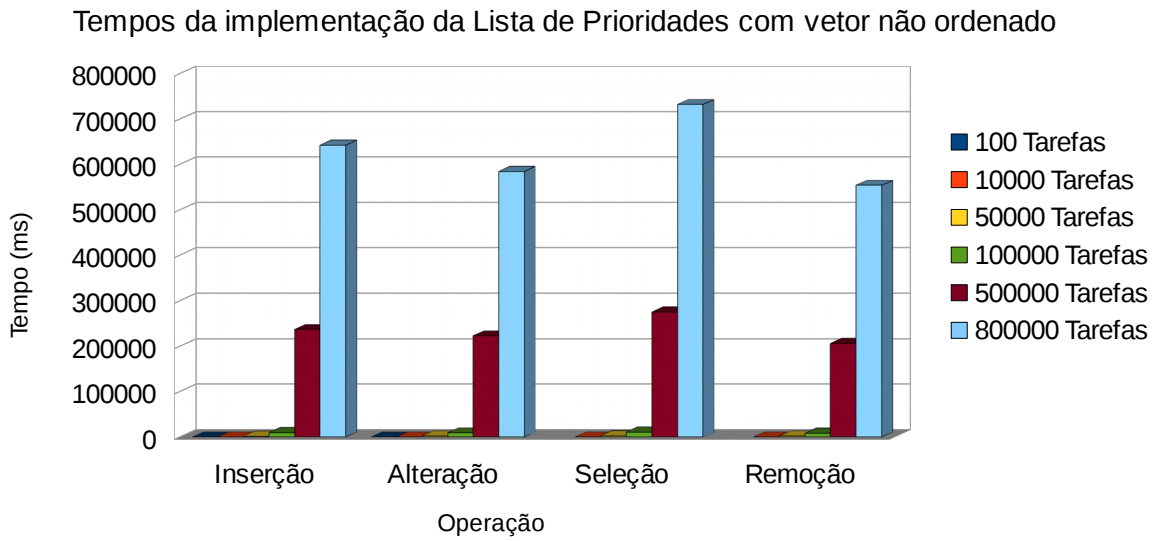
Na tabela 2 e no Gráfico 2, podemos observar que a Lista de Prioridades utilizando o vetor ordenado possui menores tempos nas tarefas que possuem mais remoção e nas tarefas que possuem mais seleção. E possui maiores tempos nas tarefas que exigem mais inserções e as que exigem mais alteração.

Por fim, iremos observar os tempos de execução das tarefas utilizando um vetor não ordenado. Segue a tabela dos tempos (em milissegundos).

Operação/ Tarefas	100	10000	50000	100000	500000	800000	TOTAL
I	1	120	1719	9435	236394	642211	889880
A	1	101	2312	9011	222801	584406	818632
S	0	119	2780	11100	275252	732183	1021434
R	0	96	2089	8370	206251	554287	771093
TOTAL	2	436	8900	37916	940698	2513087	3501039

Tabela 3 – Tempos da implementação da Lista de Prioridades com vetor não ordenado (I = Inserção, A = Alteração, S = Selecionar, R = Remover)

Gráfico 3



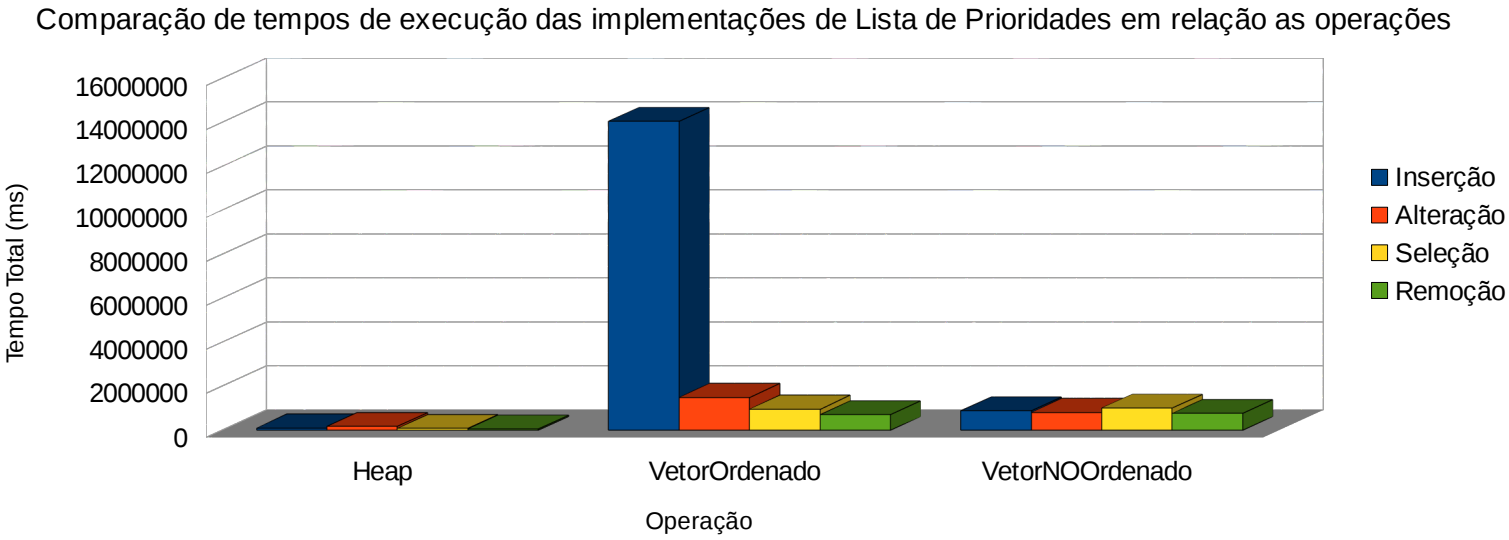
Na tabela 3 e Gráfico 3, vemos os tempos de execução de uma Lista de Prioridades implementada a partir de um vetor não ordenado. Podemos concluir que esta implementação é melhor para tarefas com mais remoções e é uma implementação ruim para tarefas que utilizem muita seleção.

Agora trabalhando com tempos totais (em milissegundos) em relação as operações iremos comparar os tempos de cada implementação.

Implementação/ Operação	I	A	S	R
Heap	109775	172356	85717	64365
VetorOrdenado	14065839	1501439	968031	712044
VetorNOOrdenado	889880	818632	1021434	771093

Tabela 4 – Comparação de tempos de execução das implementações de Lista de Prioridades em relação as operações (I = Inserção, A = Alteração, S = Selecionar, R = Remover)

Gráfico 4



Um fato relevante aqui é que o método construir também é um método que não foi computado seu tempo de forma agregada para não atrapalhar e confundir na comparação dos tempos das operações.

Analisando a tabela e o Gráfico vemos que a estrutura Heap é a melhor estrutura para implementar uma lista de Prioridades. A implementação do vetor ordenado é melhor que o vetor não ordenado nas tarefas onde acontecem mais seleções e remoções.

Agora iremos analisar cada operação:

- **Inserção:**

  Vimos que nas tarefas onde se utiliza mais inserções do que as outras operações o Heap tende a ser o melhor. A complexidade do Heap na inserção é O(log n) enquanto do vetor ordenado é O(n). Porém no vetor não ordenado é O(1), mas o fato das demais operações serem muito mais lentas que o Heap, ele acaba tendo um tempo maior.

  Fazendo uma comparação o tempo gasto nesta operação no Heap representa somente 12% do tempo gasto usando vetor não ordenado e representa impressionantes 0,7% do tempo gasto na implementação do vetor ordenado.

  Nas execuções onde acontece mais operações de inserção a implementação do vetor não ordenado é muito mais rápido que do vetor ordenado, fato que deve-se a complexidade desta operação no vetor ordenado.

- **Alteração:**

  Podemos observar que nas tarefas onde se utiliza mais alterações do que as outras operações o Heap é sempre superior. A complexidade do Heap na alteração é O(n log n) enquanto as implementações do vetor ordenado e do vetor não ordenado possuem complexidade no pior caso de $O(n^2)$ e O(n) respectivamente.

  Comparando o tempo gasto nesta operação no Heap representa 21% do tempo gasto usando vetor não ordenado e representa 11% do tempo gasto na implementação do vetor ordenado.

Vimos que nas execuções onde possuem mais este tipo de operação o tempo de execução do vetor não ordenado é muito superior a implementação do vetor ordenado. Isso deve-se muito a complexidade do vetor ordenado que é muito ruim em comparação com as demais implementações da Lista de Prioridades.

- **Seleção:**

  Vimos que nas tarefas onde se utiliza mais seleções do que as outras operações o Heap tende a ser o melhor. A complexidade do Heap e do vetor ordenado na seleção é O(1) enquanto do não ordenado é O(n).

  É bom lembrar que mesmo que a complexidade do Heap e a do vetor ordenado seja semelhantes os tempos de execuções são diferentes, isso deve-se ao fato das outras operações do vetor ordenado possuírem complexidade inferiores do que a implementação com o Heap.

  Fazendo uma comparação o tempo gasto nesta operação no Heap representa aproximadamente 8% do tempo gasto usando nas implementações com vetor não ordenado e do vetor ordenado.

  O fato das operações de inserção e alteração serem muito lentas na implementação do vetor ordenado faz com que se equilibre o tempo de execução de tarefas onde possuem a mais operações de seleção com o tempo de execução desse mesmo de tarefa com a implementação do vetor não ordenado.

- **Remoção:**

  Podemos observar que nas tarefas onde se utiliza mais remoções do que as outras operações o Heap é sempre superior. A complexidade do Heap na alteração é O(log n) enquanto as implementações do vetor ordenado e do vetor não ordenado possuem complexidade no pior caso de O(1) e O(n) respectivamente.

  Comparando o tempo gasto nesta operação no Heap representa somente 9% do tempo gasto usando vetor não ordenado e representa 8% do tempo gasto na implementação do vetor ordenado.

  Apesar da complexidade do vetor ordenado ser a melhor que as das demais implementações, o tempo do vetor ordenado é muito ruim em relação ao Heap, pois as demais operações possuem sempre implementações de complexidade piores ou iguais no vetor ordenado em comparação ao Heap.

Podemos concluir que na grande maioria dos casos é melhor implementar uma Lista de Prioridades utilizando o Heap do que utilizando as outras estruturas. E podemos observar que a implementação do vetor ordenado é de melhor que a melhor que a implementação do vetor não ordenado em nas operações onde acontecem mais operações de seleção e remoção e é inferior onde acontecem mais operações de inserção e alteração.