Trabalho Exercício-Programa
Análise de tempo e consumo de memória de algoritmos de ordenação

Gabriel Estevam de Oliveira
Igor Zanelatto Favro
Raul Mendes Rosá

DESCRIÇÃO

A atividade proposta para a análise de tempo e consumo de memória de algoritmos de ordenação consiste em uma série de testes realizados em listas encadeadas com variadas instâncias. Com um número determinado de iterações, calcula-se o tempo e a memória gasta em cada uma e obtêm-se uma média. Em cada iteração cria-se uma nova lista preenchida com números aleatórios e o programa os rearranjará em ordem crescente. Foram abordados os seguintes algoritmos: Ordenação por Inserção, Ordenação por Seleção, Quicksort, Mergesort e Heapsort.

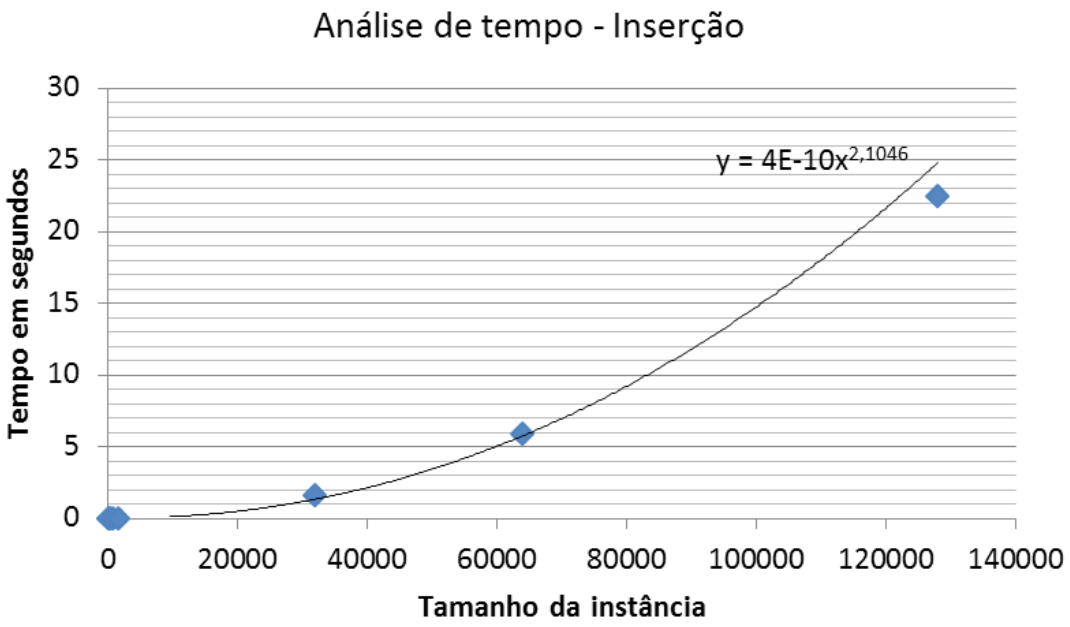
1 Ordenação por Inserção

Implementado em uma lista simplesmente encadeada e sem cabeça, o algoritmo por método de inserção consiste em procurar o elemento de conteúdo mínimo e colocar na primeira posição. Em seguida, procurar o elemento de conteúdo mínimo dos restantes e colocar na segunda posição. E assim, sucessivamente até que todos os elementos estejam em ordem crescente.

A seguir, a tabela com a quantidade média de memória utilizada em cada tamanho de instância e o respectivo tempo médio.

Tamanho da Instância	Memória Utilizada	Tempo médio
100	1656 bytes	0,0000078s
200	3256 bytes	0,0000319s
400	6456 bytes	0,0001254s
800	12856 bytes	0,0005081s
1600	25656 bytes	0,0023509s
32000	512056 bytes	1,5964024s
64000	1024056 bytes	5,8743175s
128000	2048056 bytes	22,4451157s
256000	-	Aprox. 96,4365849s
512000	-	Aprox. 414,7530145s
1024000	-	Aprox. 1783,763529s
2048000	-	Aprox. 7671,583368s
4096000	-	Aprox. 32993,83041s

Com o gráfico é possível perceber uma tendencia exponencial crescente com o tamanho da instância. Testes com instância acima de 128000 elementos custariam um longo tempo de execução, contudo, com a curva pode-se fazer uma estimativa.



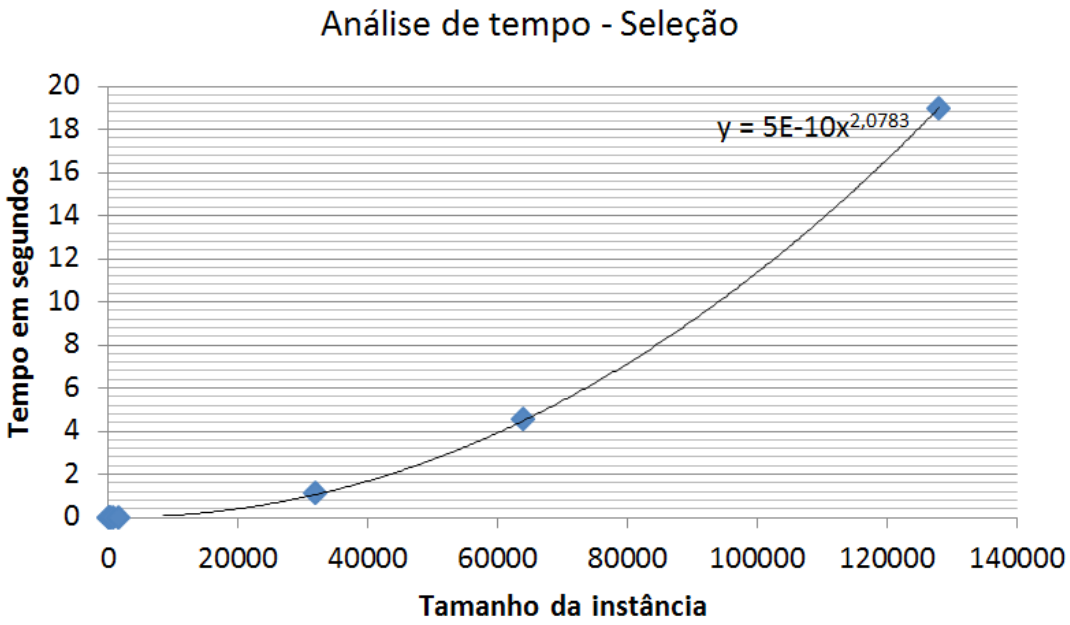
2 Ordenação por Seleção

Implementado em uma lista simplesmente encadeada e sem cabeça, o algoritmo por método de seleção consiste em procurar para cada posição o elemento que deveria estar na mesma. Começando da primeira até a ultima posição, o elemento procurado é o de conteúdo mínimo dos subsequentes à aquela posição.

A seguir, a tabela com a quantidade média de memória utilizada em cada tamanho de instância e o respectivo tempo médio.

Tamanho da Instância	Memória Utilizada	Tempo médio
100	1648 bytes	0,0000075s
200	3248 bytes	0,0000284s
400	6448 bytes	0,0001109s
800	12848 bytes	0,0004377s
1600	25648 bytes	0,0020816s
32000	512048 bytes	1,1260762s
64000	1024048 bytes	4,5481322s
128000	2048048 bytes	18,9761370s
256000	-	Aprox. 86,8790622s
512000	-	Aprox. 366,8983547s
1024000	-	Aprox. 1549,445854s
2048000	-	Aprox. 6543,453857s
4096000	-	Aprox. 27633,61383s

Assim como em Ordenação por Inserção, a Ordenação por Seleção também apresenta uma tendencia exponencial com o tamanho da instância, com o crescimento um pouco menos acentuado. Com instâncias acima de 128000 elementos apenas pode-se fazer uma estimativa.



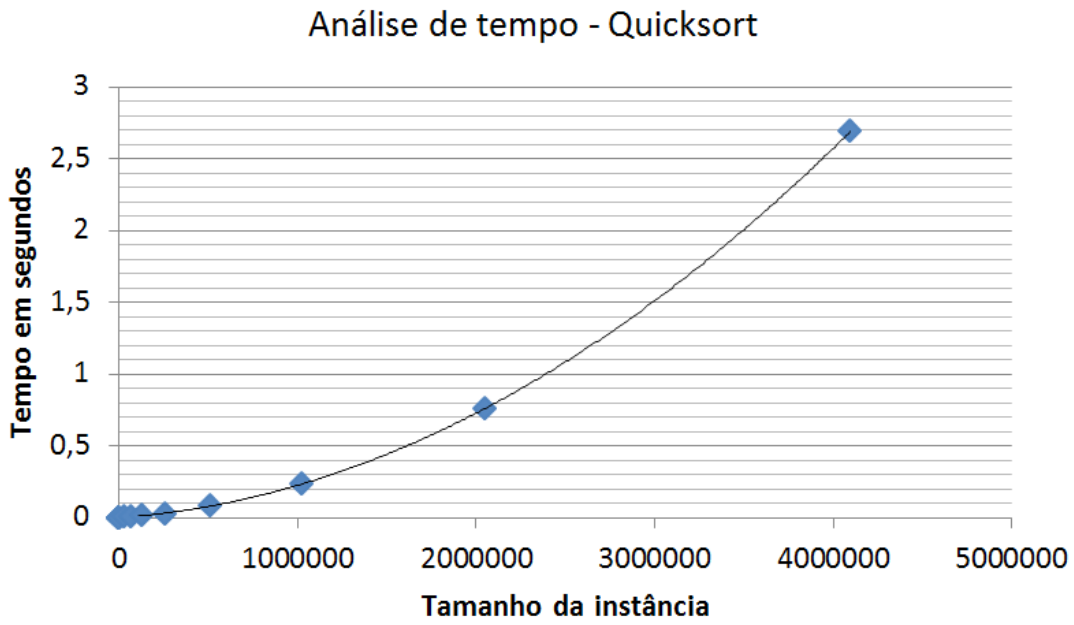
3 Quicksort

Implementado em uma lista duplamente encadeada, circular e sem cabeça, o algoritmo apresenta estrutura recursiva, dividindo a instância em duas novas instâncias (em alguns casos uma das partes pode ser vazia). A divisão é realizada por um método de separação que consiste em pegar o último elemento e rejeitar os outros de forma de todos os elemento anteriores sejam menores ou iguais e os seguintes sejam maiores. O processo é repetido até que em uma instância a primeira célula coincida com a última.

A seguir, a tabela com a quantidade média de memória utilizada em cada tamanho de instância e o respectivo tempo médio.

Tamanho da Instância	Memória Utilizada	Tempo médio
100	4673,14 bytes	0,0000042s
200	9340,67 bytes	0,0000093s
400	18673,62 bytes	0,0000206s
800	37342,02 bytes	0,0000452s
1600	74692,89 bytes	0,0000993s
32000	1599666,40 bytes	0,0027902s
64000	3447263,20 bytes	0,0057041s
128000	7280042,40 bytes	0,0123718s
256000	14960020,00 bytes	0,0295486s
512000	30320020,00 bytes	0,0782969s
1024000	61040020,00 bytes	0,2337374s
2048000	122480020,00 bytes	0,7595744s
4096000	245360020,00 bytes	2,6882144s

Com implementação em lista encadeada, o algoritmo de ordenação Quicksort apresenta o melhor desempenho de tempo entre os algoritmos implementados. Com ele foi possível realizar testes com instâncias de tamanho até 4096000.



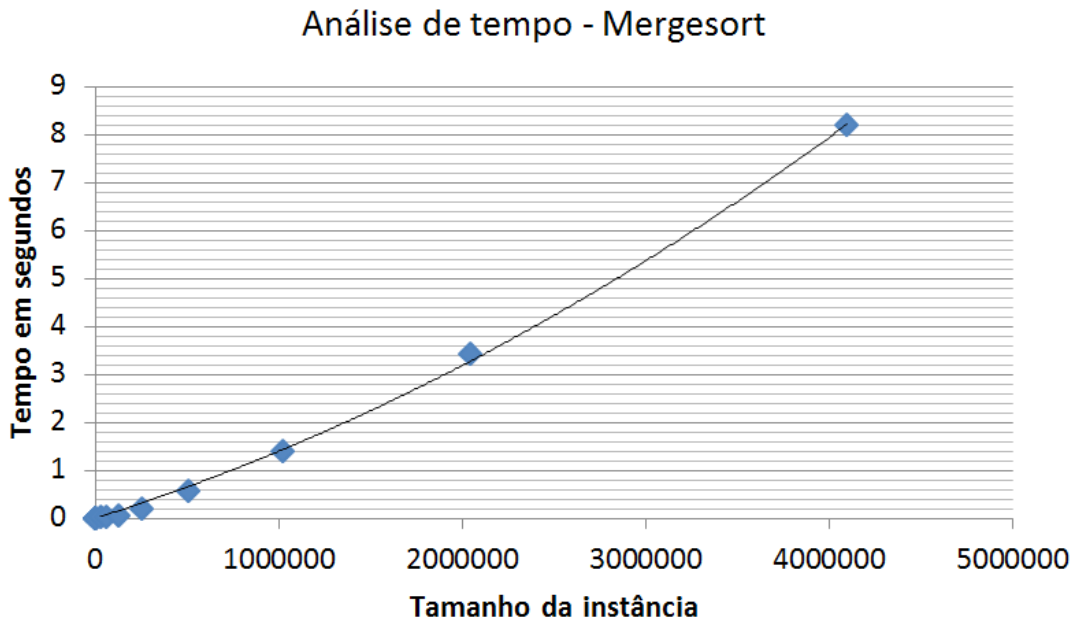
4 Mergesort

Implementado em uma lista duplamente encadeada, com cabeça e cauda, o algoritmo apresenta estrutura recursiva, dividindo a instância ao meio até que esta tenha apenas um elemento. Após cada divisão, uma função intercala as duas partes já ordenadas.

A seguir, a tabela com a quantidade média de memória utilizada em cada tamanho de instância e o respectivo tempo médio.

Tamanho da Instância	Memória Utilizada	Tempo médio
100	13924 bytes	0,0000069s
200	27924 bytes	0,0000155s
400	55924 bytes	0,0000335s
800	111924 bytes	0,0000727s
1600	223924 bytes	0,0001671s
32000	4479924 bytes	0,0085279s
64000	8959924 bytes	0,0201296s
128000	17919924 bytes	0,0519137s
256000	35839924 bytes	0,1848985s
512000	71679924 bytes	0,5726298s
1024000	143359924 bytes	1,3872661s
2048000	286719924 bytes	3,4069487s
4096000	573439924 bytes	8,1851602s

O algoritmo de ordenação Mergesort também apresenta um bom desempenho de tempo, muito mais rápido que os dois primeiros vistos (Inserção e Seleção), mas um pouco mais lento e com mais consumo de memória do que o Quicksort. Com o Mergesort também foi possível realizar testes com instâncias de tamanho até 4096000.



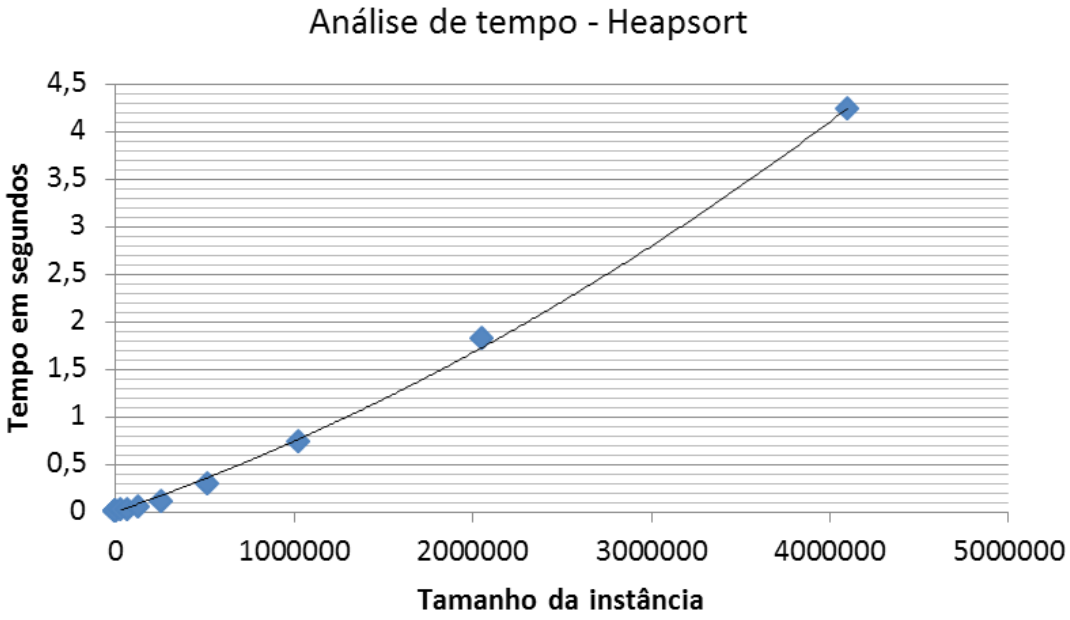
5 Heapsort

Implementado em uma lista duplamente encadeada e circular, o algoritmo usa o conceito chamado *maxheap* para obter o maior elemento e colocar na última posição da lista. A seguir, utiliza a função *maxheapfica* para deixar novamente a lista em forma de um *maxheap* e colocar o segundo maior na penúltima posição da lista. O processo é repetido até que toda a lista esteja em ordem crescente.

A seguir, a tabela com a quantidade média de memória utilizada em cada tamanho de instância e o respectivo tempo médio.

Tamanho da Instância	Memória Utilizada	Tempo médio
100	15584 bytes	0,0000057s
200	31184 bytes	0,0000131s
400	62384 bytes	0,0000293s
800	124784 bytes	0,0000658s
1600	249584 bytes	0,0001504s
32000	4991984 bytes	0,0061073s
64000	9983984 bytes	0,0144073s
128000	19967984 bytes	0,0362747s
256000	39935984 bytes	0,1006507s
512000	79871984 bytes	0,2867615s
1024000	159743984 bytes	0,7260371s
2048000	319487984 bytes	1,8069805s
4096000	638975984 bytes	4,2213284s

O algoritmo de ordenação Heapsort devido a complexidade de aplicar o conceito de *maxheap* exigiu mais esforços para implementação. Após diversas tentativas o código final presentou um bom desempenho. E com o Heapsort também foi possível realizar testes com instâncias de tamanho até 4096000.


Análise de tempo - Heapsort
Tempo em segundos
4,5
4
3,5
3
2,5
2
1,5
1
0,5
0
0
1000000
2000000
3000000
4000000
5000000
Tamanho da instância