Universidade Federal do Agreste de Pernambuco
Bacharelado em Ciência da Computação

AED1
Prof.: Igor Medeiros Vanderlei
Aluno: Rodrigo Leandro dos Santos

Métodos de Ordenação

1 Introdução

Um Problema muito comum é precisar que determinados dados sejam armazenados obedecendo uma determinada ordem. Temos basicamente duas maneiras de fazer está ordenação, a primeira maneira é inserindo os elementos já respeitando está ordem estabelecida previamente, e a segunda maneira seria utilizar algum algoritmo para ordenar a lista já formada.

Por ser algo realmente comum, algumas linguagens de programação já trazem isso implementado, mas para as que não trazem temos diversas maneias de fazer esta ordenação, porém alguns métodos são inviáveis para um numero elevado de dados, pois para que faça sentido usar um método de ordenação no sentido computacional temos que seguir duas regras básicas, ele te que ser eficiente tanto em termo de tempo (deve ser rápido) como em termo de espaço (deve ocupar pouca memória durante a execução). As ordenações funcionam da seguinte maneira:

* Entrada → é um vetor cujos elementos precisam ser ordenados;

* Saída → é o mesmo vetor com seus elementos na ordem especificada;

2 Metodologia

Para realizar a sequência de testes foram utilizadas 6 listas, contendo números inteiros aleatórios, que foram gerados utilizando a classe "Java.util.Randow"da linguagem Java, as 6 listas contém respectivamente 1000, 10.000, 50.000, 100.000, 500.000 e 1.000.000 de elementos. O tempo de execução foi calculado utilizando a função "time"do terminal linux.

Os testes foram realizados em um desktop, utilizando o windows 10, com o processador ryzen 3200g, 16 gb ram, hd 1tb. Esse desktop estava rodando uma maquina virtual com o sistema operacional linux ubuntu, configurações da maquina virtual: 2gb ram, hd 30gb. Todos os testes foram feitos na máquina virtual e sobre as mesmas condições.

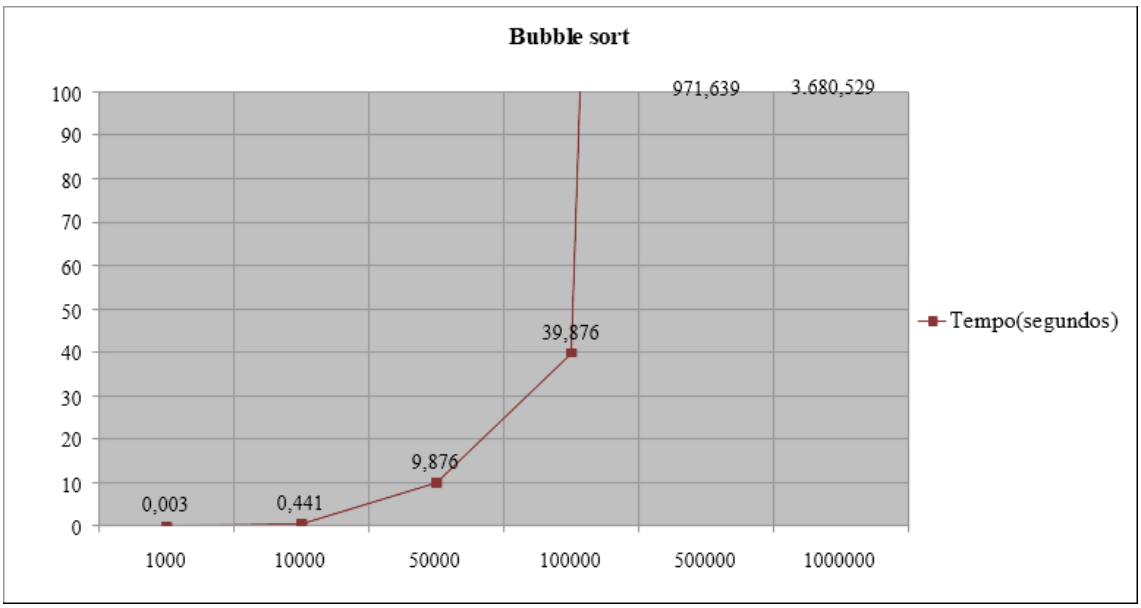
3 Resultados

3.1 Bubble sort

O algoritmo de "ordenação bolha", ou "bubble sort", recebeu este nome por causa do pensamento que os elementos maiores são mais leves, e sobem como bolhas até suas posições corretas. O seu funcionamento está baseado em uma serie de comparações entre seus elementos, por exemplo se começarmos as comparações do final para o inicio vamos começar as comparações comparando o ultimo com seu antecessor, se eles estiverem fora da ordem desejada esses dois elementos são trocados de posição, ficando em ordem correta em seguida independente se houve ou não troca após a primeira comparação, o penúltimo elemento é comparado com o antepenúltimo,O processo continua até que o segundo elemento seja comparado com o primeiro. Assim garantimos que o primeiro elemento estará da posição correta ao fim da 1 rodada, onde teremos n-1 rodadas porque se por exemplo tivermos um vetor de 10 elementos e ordenamos 9, o elemento 10 já está na posição correta.

Segue a seguir a tabela com o tempo de execução do método em minutos e segundos para cada número de elementos a serem ordenados, e logo apos a tabela, temos o gráfico do método de ordenação considerando o tempo apenas em segundos.

	Bubble
1000	0m 0,003s
10.000	0m 0,441s
50.000	0m 9,876s
100.000	0m 39,876s
500.000	16m 11,639s
1.000.000	61m 20,529s

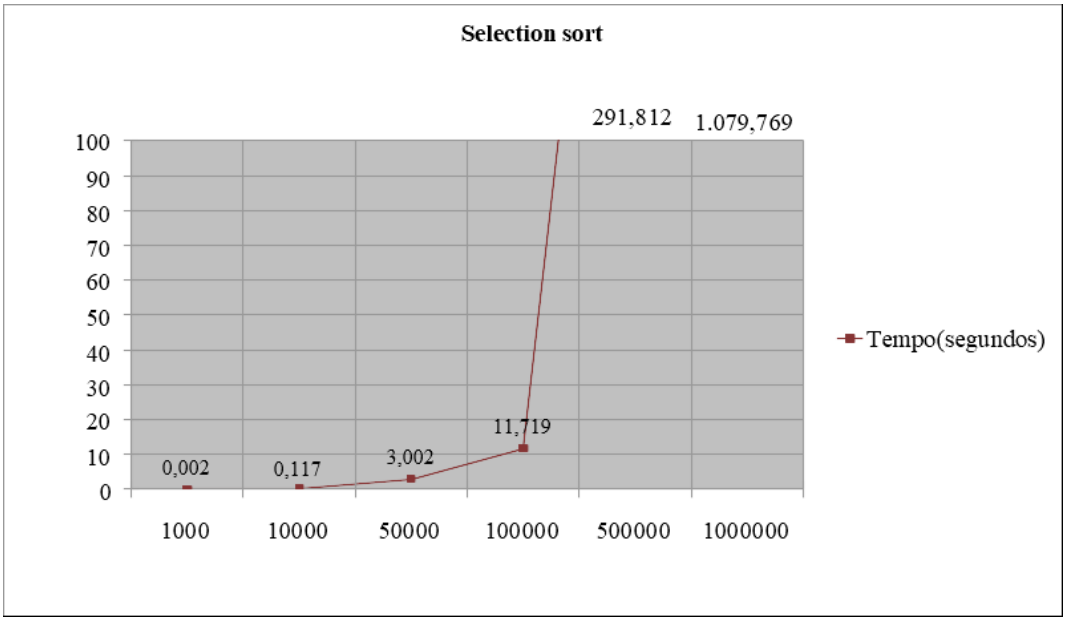


3.2 Selection sort

O algoritmo de ordenação por seleção ou Selection sort, funciona da seguinte maneira, ele vai trabalhar com o vetor em duas partes, a parte do vetor que está está ordenada, e a parte desordenada, a parte ordenada começa sem nenhum elemento, em seguida ele procura o menor elemento(ou maior) do vetor e troca ele com a menor posição,agora temos um elemento na parte ordenada, pois garantimos que esse elemento está na posição correta, na próxima rodada nós vamos repetir o passo de encontrar o menor elemento, mas agora só vamos procurar na parte desordenada e vamos trocar ele com o primeiro elemento da própria parte desordenada, e assim temos a parte ordenada incrementada, repetimos esse processo n-1 vezes.

Segue a seguir a tabela com o tempo de execução do método em minutos e segundos para cada número de elementos a serem ordenados, e logo apos a tabela, temos o gráfico do método de ordenação considerando o tempo apenas em segundos.

	Selection
1000	0m 0,002s
10.000	0m 0,117s
50.000	0m 3,002s
100.000	0m 11,719s
500.000	4m 51,812s
1.000.000	17m 59,769

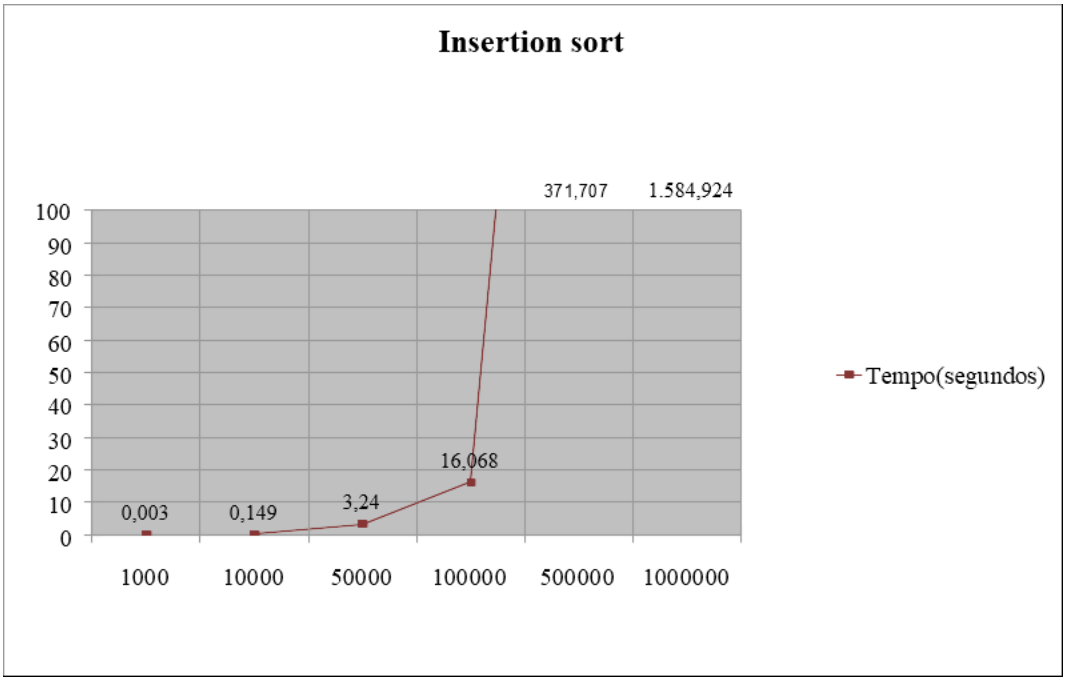


3.3 Insertion sort

A ordenação por inserção ou Insertion sort, também usa uma divisão imaginaria do vetor, a parte ordenada e a parte desordenada, a parte ordenada começa com um elemento, pois uma lista de um elemento já está ordenada, então vamos pegando um a um os elementos da parte desordenada e vamos inserindo na parte ordenada respeitando a ordem estabelecida até que o vetor esteja ordenado.

Segue a seguir a tabela com o tempo de execução do método em minutos e segundos para cada número de elementos a serem ordenados, e logo apos a tabela, temos o gráfico do método de ordenação considerando o tempo apenas em segundos.

	Insertion
1000	0m 0,003s
10.000	0m 0,149s
50.000	0m 3,240s
100.000	0m 16,068s
500.000	6m 11,707s
1.000.000	26m 24,924 s



3.4 Heap sort

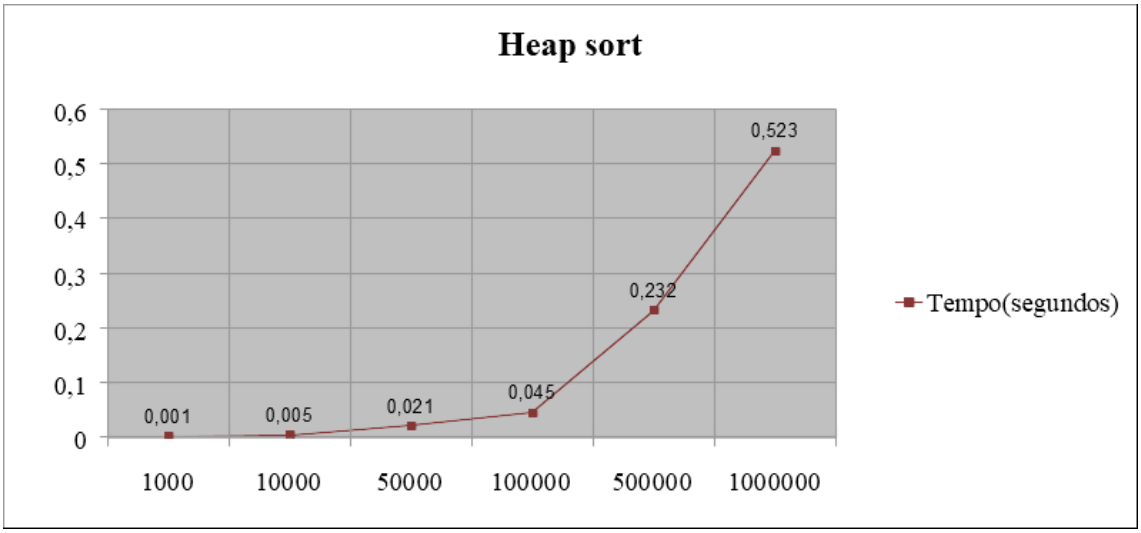
Heap sort é uma técnica de ordenação baseada em comparação com base na estrutura de dados binários Heap. É semelhante ao selection sort, onde primeiro encontramos o elemento mínimo e colocamos o elemento mínimo no início. Repetimos o mesmo processo para os demais elementos.

Um Heap Binário é uma Árvore Binária Completa onde os itens são armazenados em uma ordem de modo que o valor do pai seja maior (ou menor) que os valores em seus dois nós filhos. O primeiro é chamado heap máximo e o último é chamado heap mínimo. O heap é sempre preenchido da esquerda para direita.

O processo de Ordenação acontece da seguinte maneira, primeiro é montada o heap seguindo suas regras, em seguida trocamos o valor do maior elemento com o ultimo do elemento do heap, simulando a retirada do elemento do heap; em seguida reorganisamos o heap e repetimos o processo até o vetor está ordenado.

Segue a seguir a tabela com o tempo de execução do método em minutos e segundos para cada número de elementos a serem ordenados, e logo apos a tabela, temos o gráfico do método de ordenação considerando o tempo apenas em segundos.

	Heap
1000	0m 0,001s
10.000	0m 0,005s
50.000	0m 0,021s
100.000	0m 0,045s
500.000	0m 0,232s
1.000.000	0m 0,523s

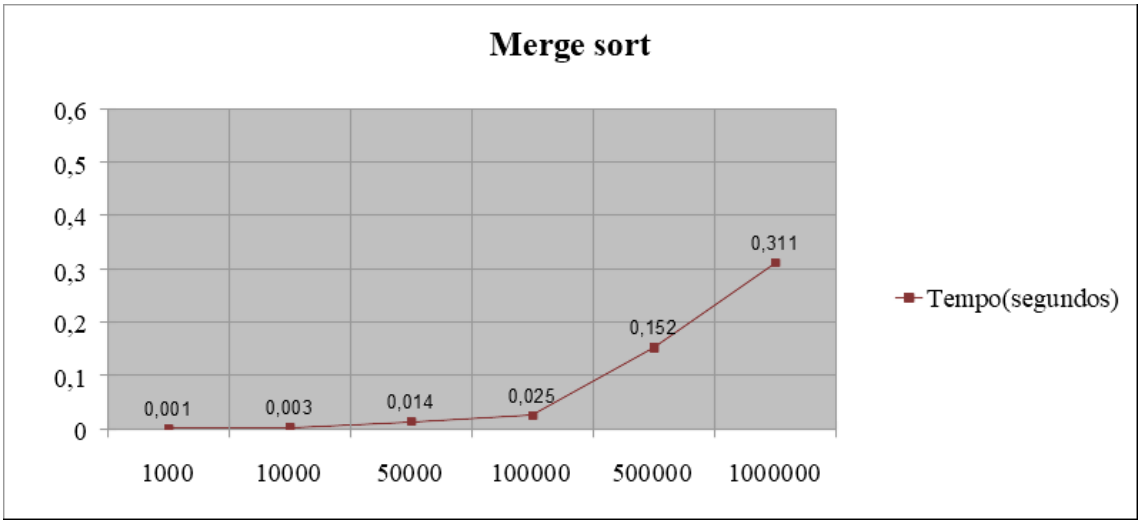


3.5 Merge sort

o Merge Sort é um algoritmo de divisão e conquista. Ele divide o vetor de entrada em duas metades, essas metades em outras metades, e repete o processo até termos listas unitárias, com os as listas unitárias voltamos a juntar as partes que foram separadas, mas juntamos ordenando as listas até voltarmos a ter uma única lista mas desta vez ordenada.

Segue a seguir a tabela com o tempo de execução do método em minutos e segundos para cada número de elementos a serem ordenados, e logo apos a tabela, temos o gráfico do método de ordenação considerando o tempo apenas em segundos.

	Merge
1000	0m 0,001s
10.000	0m 0,003s
50.000	0m 0,014s
100.000	0m 0,025s
500.000	0m 0,152s
1.000.000	0m 0,311s

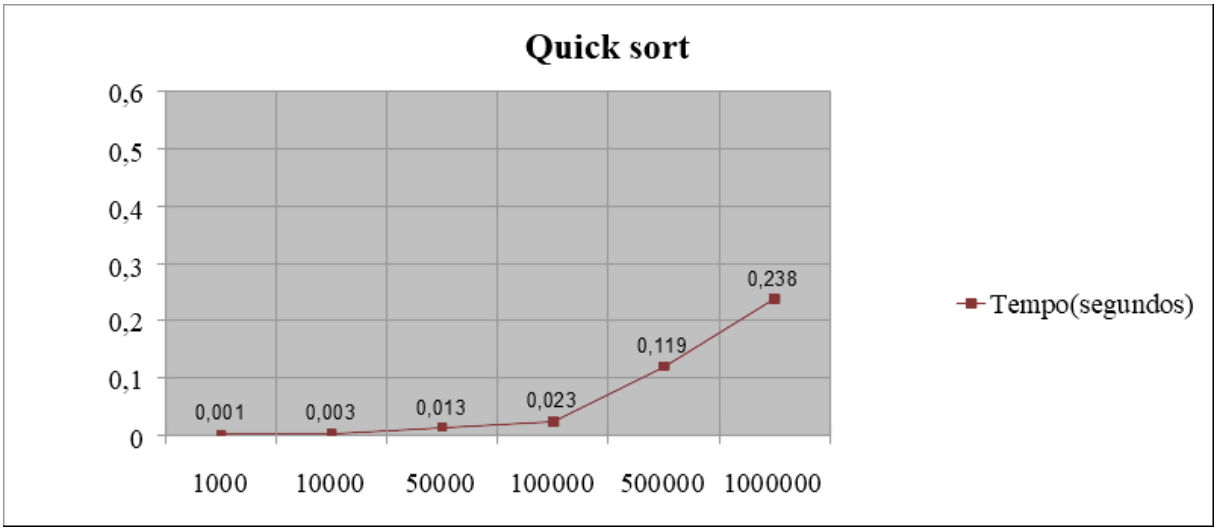


3.6 Quick sort

O quick sort é um método que usa o estilo de divisão e conquista. Ele escolhe um elemento como pivô, e na primeira rodada separamos todos os elementos maiores que o pivô de um lado do vetor e todos os menores que ele do outro lado do vetor. Com isso garantimos que o pivô esta no local certo, agora repetimos todo o processo para as duas listas geradas(as dos dois lados do pivô), e vai repetindo o processo para as listas geradas pelos pivôs até o vetor está ordenado.

Segue a seguir a tabela com o tempo de execução do método em minutos e segundos para cada número de elementos a serem ordenados, e logo apos a tabela, temos o gráfico do método de ordenação considerando o tempo apenas em segundos.

	Quick
1000	0m 0,001s
10.000	0m 0,003s
50.000	0m 0,013s
100.000	0m 0,023s
500.000	0m 0,119s
1.000.000	0m 0,238s

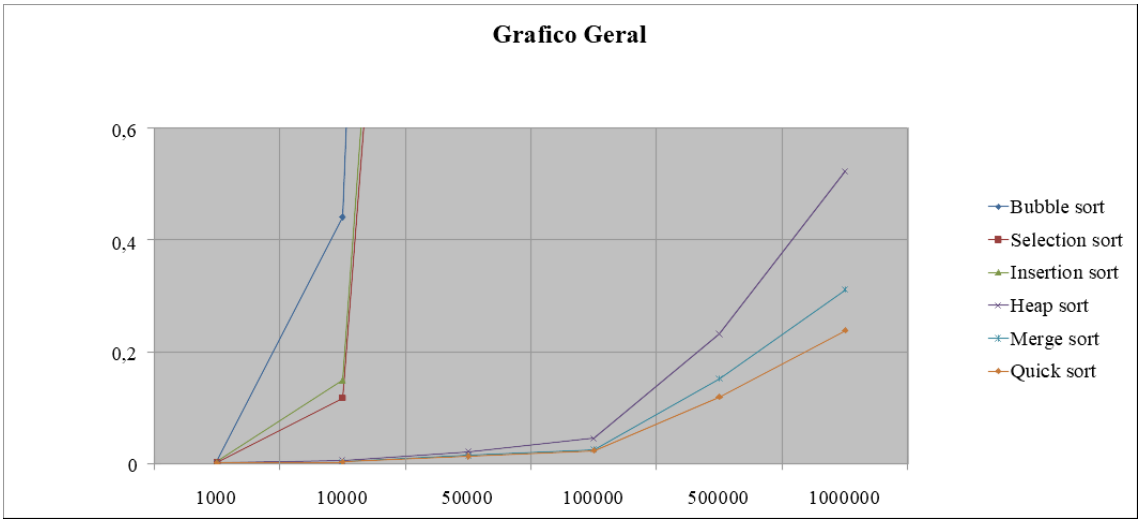


4 Considerações finais

A seguir temos a tabela contendo o tempo de execução de todos os métodos em minutos e segundos para cada número de elementos a serem ordenados, e em seguida temos a mesma tabela só que em segundos.

	Bubble	Selection	Insertion	Heap	Merge	Quick
1000	0m 0,003s	0m 0,002s	0m 0,003s	0m 0,001s	0m 0,001s	0m 0,001s
10.000	0m 0,441s	0m 0,117s	0m 0,149s	0m 0,005s	0m 0,003s	0m 0,003s
50.000	0m 9,876s	0m 3,002s	0m 3,240s	0m 0,021s	0m 0,014s	0m 0,013s
100.000	0m 39,876s	0m 11,719s	0m 16,068s	0m 0,045s	0m 0,025s	0m 0,023s
500.000	16m 11,639s	4m 51,812s	6m 11,707s	0m 0,232s	0m 0,152s	0m 0,119s
1.000.000	61m 20,529s	17m 59,769	26m 24,924 s	0m 0,523s	0m 0,311s	0m 0,238s

	Bubble	Selection	Insertion	Heap	Merge	Quick
1000	0,003	0,002	0,003	0,001	0,001	0,001
10.000	0,441	0,117	0,149	0,005	0,003	0,003
50.000	9,876	3,002	3,24	0,021	0,014	0,013
100.000	39,876	11,719	16,068	0,045	0,025	0,023
500.000	971,639	291,812	371,707	0,232	0,152	0,119
1.000.000	3.680,529	1.079,769	1.584,924	0,523	0,311	0,238



Analisando a tabela e o gráfico podemos observar algumas coisas que eram esperadas e outras que nem tanto, a primeira coisa que podemos observar é que para uma pequena quantidade de dados(1000), todos os métodos tem tempos aceitáveis, porém já da pra notar uma superioridade pelos métodos de maior complexidade(heap, merge, quick) sobre os mais simples(insertion, selection, bubble), mas até em tão nada muito perceptível. A partir de 10.000 elementos os métodos mais simples apresentam um aumento já bem significativo, principalmente o método de ordenação bubble sort que já não fica muito interessante seu uso em consideração aos outros métodos, a partir dos 100.000 elementos os outros dois dos métodos simples também já ficam inviáveis. Os métodos complexos seguem com resultados bem parecidos até os 100.000 elementos onde após isso eles começam a apresentar uma diferença com o Quick sort se destacando e o heap sort tendo o pior tempo entre eles. Portanto podemos concluir que escolher qual método usar vai depender muito da atividade a ser utilizada tendo um método que vai se destacar para cada situação, como para cada exemplo o quick se destaca na eficiência com um grande número de elementos.