Relatório - Métodos de ordenação

Igor Filipi Cardoso - AQ3022587

Neste relatório irei mostrar a implementação e comparação de cinco métodos de ordenação, evidenciando a quantidade de comparações e trocas de cada, e claro, o tempo com que cada um conseguiu ordenar diversas listas.

Os algoritmos de ordenação que serão utilizados são o:

- Bubble Sort (Versão Otimizada)
- Selection Sort
- Insert Sort
- Quick Sort
- Merge Sort

Todos os algoritmos de ordenação foram implementados pelo professor Edinilson Rossi, com pequenas alterações e adaptações feitas por mim, para que encaixassem no relatório e funcionassem utilizando listas de 1 milhão de elementos.

Algoritmos feitos em C, utilizando o Visual Studio Code como IDE e local de teste.

Especificação do computador utilizado para os testes:

Processador: Ryzen 5 5600
Clock: 3.5GHz
Memória Ram: 32GB
Sistema Operacional: Windows 11 Home (64 bits)

Lembrando que, computadores e ambientes diferentes têm resultados diferentes, no quesito tempo, o intuito desse relatório é principalmente comparar os algoritmos de ordenação, em suas comparações e trocas. O tempo é importante, porém cada computador terá um resultado diferente no tempo.

As primeiras listas foram testadas com 1000 elementos, sendo uma lista com números ordenados, uma com números em ordem aleatória e a última com eles ao contrário.

1000 Elementos			
Algoritmos	Ordem Crescente (Melhor Caso)		
	Trocas	Comparações	Tempo(Sec)
BubbleSort	0	999	0
SelectionSort	0	499500	0,002
InsertSort	0	999	0
QuickSort	488	1178	0
MergeSort	5044	5044	0

Analisando a tabela com os resultados, é visível que, na lista já ordenada, com apenas 1000 elementos, todos algoritmos são muito rápidos, e em sua maioria fazem poucas comparações, tendo uma exceção claro, o Selection Sort, faz o incrível número de 499500 comparações, e por conta de tantas comparações ele acaba perdendo no tempo. Em relação aos outros, o tempo foi tão pequeno que mesmo 15 casas decimais de precisão utilizando variáveis do tipo Double não foi suficiente para mensurar o tempo que demoraram.

O QuickSort teve algumas trocas, poucas porém teve, isso se deve ao pivô que foi escolhido ser o número do meio. Sendo assim ele acaba reorganizando a lista de certa forma. Se o pivô fosse o primeiro ou o último número não teria essas trocas, porém utilizar esse método não funciona com um Array grande, com 1 milhão.

O Merge reorganizou a lista já organizada, assim tendo o mesmo tanto de trocas e comparação, uma característica que iremos ver que se repete sempre.

1000 Elementos			
Algoritmos	Ordem Decrescente (Pior Caso)		
	Trocas	Comparações	Tempo(Sec)
BubbleSort	499500	499500	0,003
SelectionSort	500	499500	0,003
InsertSort	499500	999	0,002
QuickSort	992	1693	0
MergeSort	4932	4932	0

Temos então a tabela com o array no pior caso, a lista ao contrário, o bubble faz muitas trocas e o mesmo tanto de comparações, e por conta disso demora um pouco em relação aos seus irmãos recursivos que simplesmente foram tão rápidos que não cabe nas 15 casas decimais. O Insert faz muitas comparações porém poucas trocas, assim acaba de alguma forma empatando com o Bubble. O Insert teve 1 comparação a menos do que o tamanho da lista, algo que vai se repetir também. Ele se sai um pouco mais rápido que o Bubble e o Selection mas não chega perto dos métodos recursivos.

O Quick foi muito rápido e teve poucas comparações e trocas, no momento se mostrando um dos melhores.

Como havia dito anteriormente, o Mergesort faz suas 4932 comparações e 4932 trocas, tendo o mesmo número novamente, de trocas e comparações.

1000 Elementos			
Algoritmos	Ordem Aleatória (Caso Médio)		
	Trocas	Comparações	Tempo(Sec)
BubbleSort	247993	499490	0,002
SelectionSort	990	499500	0,003
InsertSort	247993	999	0,001
QuickSort	1875	2554	0
MergeSort	8729	8729	0,001

Com o array aleatório, as mesmas coisas acontecem, porém conseguimos marcar alguns padrões.

O Selection Sort faz 499500 comparações não importando qual é a ordem dos números, e sempre faz poucas trocas. Por sua vez o Insert faz muitas trocas e poucas comparações, sendo sempre 1 a menos do que o tamanho da lista.

O Merge pela primeira vez sai dos 0 segundos e chega a 0,001. Porém ainda sim é muito mais rápido que os métodos não recursivos. O Merge novamente, tem o mesmo número de comparações e trocas.

O Quick se destaca novamente, tendo poucas comparações e poucas trocas, assim organizando a lista em menos tempo.

100000 Elementos			
Algoritmos	Ordem Crescente (Melhor Caso)		
	Trocas	Comparações	Tempo(Sec)
BubbleSort	0	99999	0,001
SelectionSort	0	4999950000	23,16
InsertSort	0	99999	0,001
QuickSort	51580	122422	0,001
MergeSort	853904	853904	0,019

100000 Elementos			
Algoritmos	Ordem Aleatória (Caso Médio)		
	Trocas	Comparações	Tempo(Sec)
BubbleSort	2500554453	4999937754	23,561
SelectionSort	99982	4999950000	23,15
InsertSort	2500554453	99999	11,59
QuickSort	320822	398306	0,007
MergeSort	1536457	1536457	0,024

100000 Elementos			
Algoritmos	Ordem Decrescente (Pior Caso)		
	Trocas	Comparações	Tempo(Sec)
BubbleSort	4999950000	4999950000	22,7
SelectionSort	50000	4999950000	22,83
InsertSort	4999950000	99999	23,17
QuickSort	103090	176196	0,002
MergeSort	815024	815024	0,019

Com a lista de 100000 itens, já ordenados, podemos ver que o Merge e o Insert seguem os padrões deles, o Insert faz o tamanho do array menos um, de comparações e por a lista está ordenada não faz nenhuma troca. O selection sort, por mais que não tenha feito nenhuma troca ele fez muitas comparações, por isso o tempo de execução dele foi muito maior que os outros.

No caso médio fica visível a diferença de um algoritmo de ordenação recursivo com um mais simples. O tempo que os métodos mais simples gastam é muito maior, apesar disso, o melhor entre os mais simples, sem a recursão, foi o Insert Sort que organizou em apenas 11 segundos e fez menos comparações que todos, porém fez muitas mais trocas que os outros, se igualando no quesito troca com o bubble sort.

No pior caso, mostra que os métodos mais simples de ordenação ficam muito para trás dos algoritmos recursivos, onde o Quick e o Merge, fazem menos comparações e menos trocas comparados aos seus irmãos não recursivos.

O Merge continua seu padrão, onde suas trocas e comparações são as mesmas. Ele fica pouco atrás do Quick Sort no tempo e na contagem de comparações e trocas.

1000000 Elementos			
Algoritmos	Ordem Crescente (Melhor Caso)		
	Trocas	Comparações	Tempo(Sec)
BubbleSort	0	999999	0,006
SelectionSort	0	499999500000	2316,97
InsertSort	0	999999	0,005
QuickSort	491871	1181627	0,024
MergeSort	10066432	10066432	6,92

1000000 Elementos			
Algoritmos	Ordem Aleatória (Caso Médio)		
	Trocas	Comparações	Tempo(Sec)
BubbleSort	250204127769	499999334975	2415,21
SelectionSort	999960	499999500000	2316,47
InsertSort	250204127769	999999	1159,53
QuickSort	3495274	4462506	0,082
MergeSort	18673684	18673684	6,94

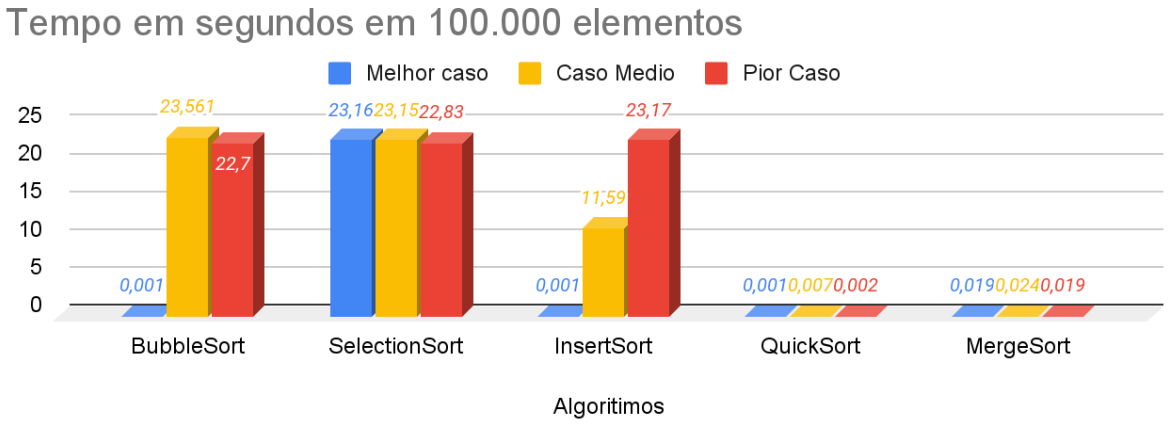
1000000 Elementos			
Algoritmos	Ordem Decrescente (Pior Caso)		
	Trocas	Comparações	Tempo(Sec)
BubbleSort	499999500000	499999500000	2266,15
SelectionSort	500000	499999500000	2285,37
InsertSort	499999500000	999999	2316,82
QuickSort	1004837	1709693	0,025
MergeSort	9884992	9884992	6,91

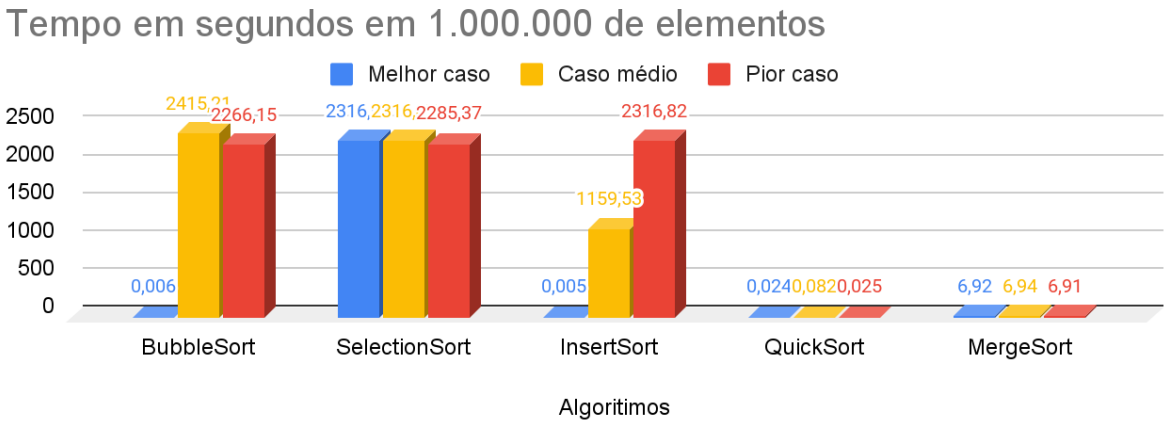
Com um array maior temos uma clara diferença entre todos os métodos. No melhor caso possível, o Bubble e o Insert não se saem mal, ficam a frente do Mergesort, que reorganizou todo o array, porém o Selection, por conta de sempre fazer o mesmo número de comparações independente de estar ordenado ou não ele demorou 2 mil segundos para conferir o array de 1 milhão de itens.

Com a ordem aleatória os métodos não recursivos acabam sendo inviáveis, onde o bubble e o Selection demoraram 2 mil horas e o Insert foi um pouco melhor, fazendo 1 mil segundos.

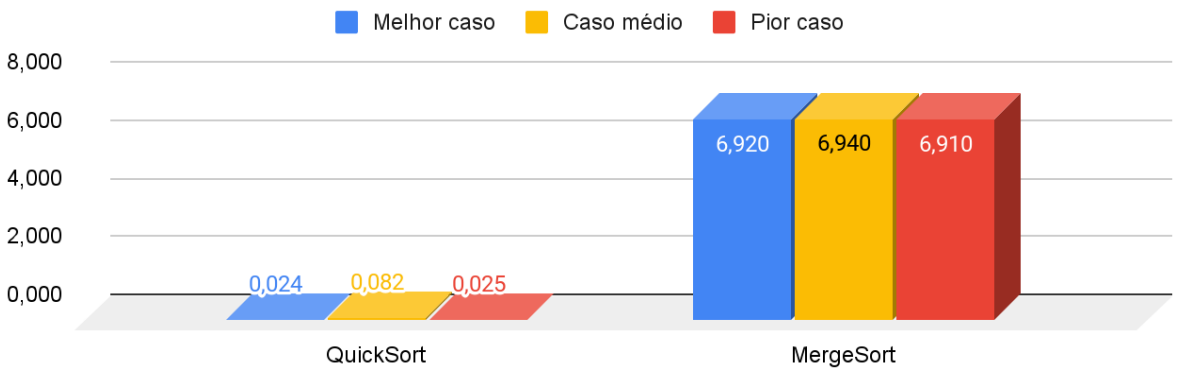
O Merge faz em quase 7 segundos, onde quando comparado aos métodos não recursivos é um resultado muito bom, já quando comparado ao Quick foi um péssimo resultado, o Quick fez menos comparações e trocas, conseguindo ordenar o array de 1 milhão em 0,082 segundos, algo impressionante.

No pior caso possível, todos os métodos não recursivos ficaram muito parecidos no tempo, até o Insert que estava indo bem agora ficou no mesmo barco. O Merge se saiu muito bem, praticamente igual ao caso médio.

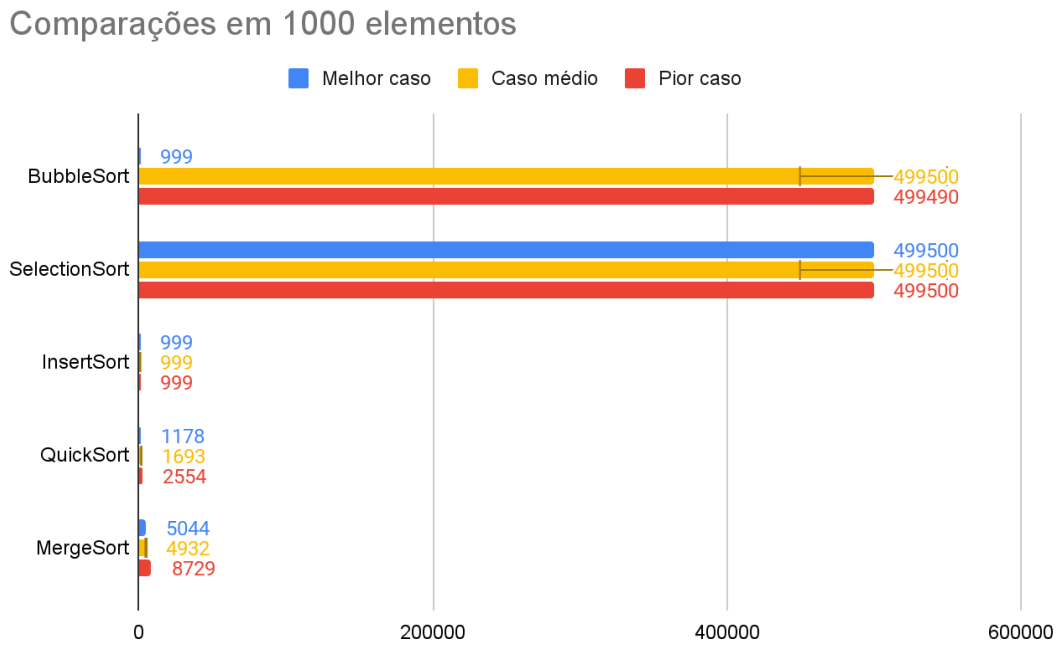


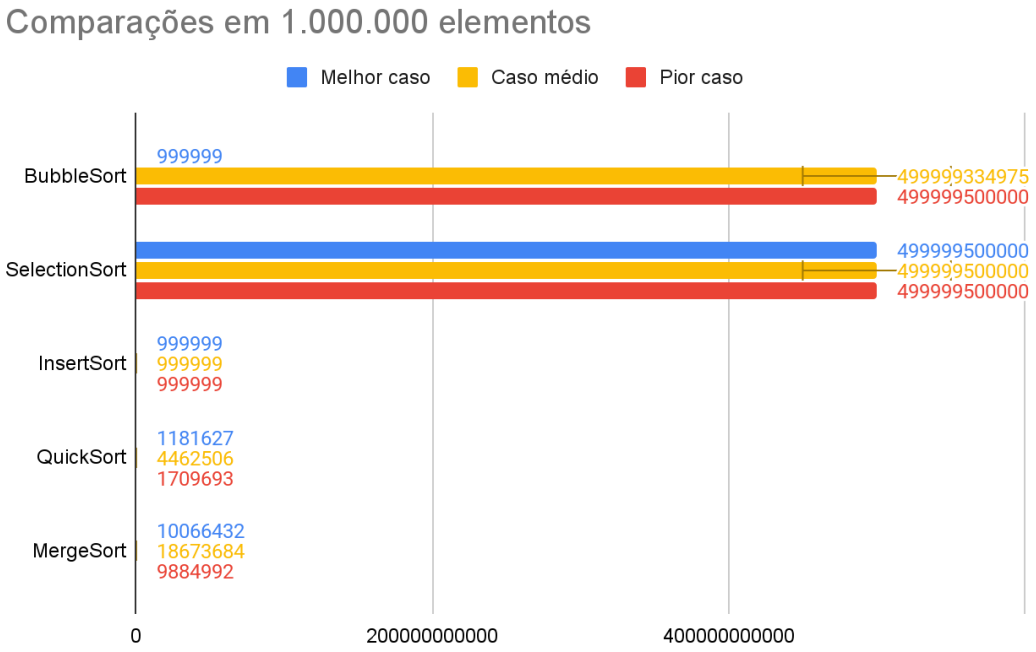


Em uma simples comparação podemos ver que, ambos os gráficos seguem o mesmo padrão, ao aumentar o tamanho da lista o tempo também aumenta, mas como também é visível existe uma proporcionalidade entre o tempo e a quantidade de elementos. Podemos ver o quão incrível é isso observando o Selection Sort, que foi de 23,16 segundos no melhor caso para 2316 segundos. Quick e Merge foram tão rápidos que praticamente nem aparecem no gráfico.

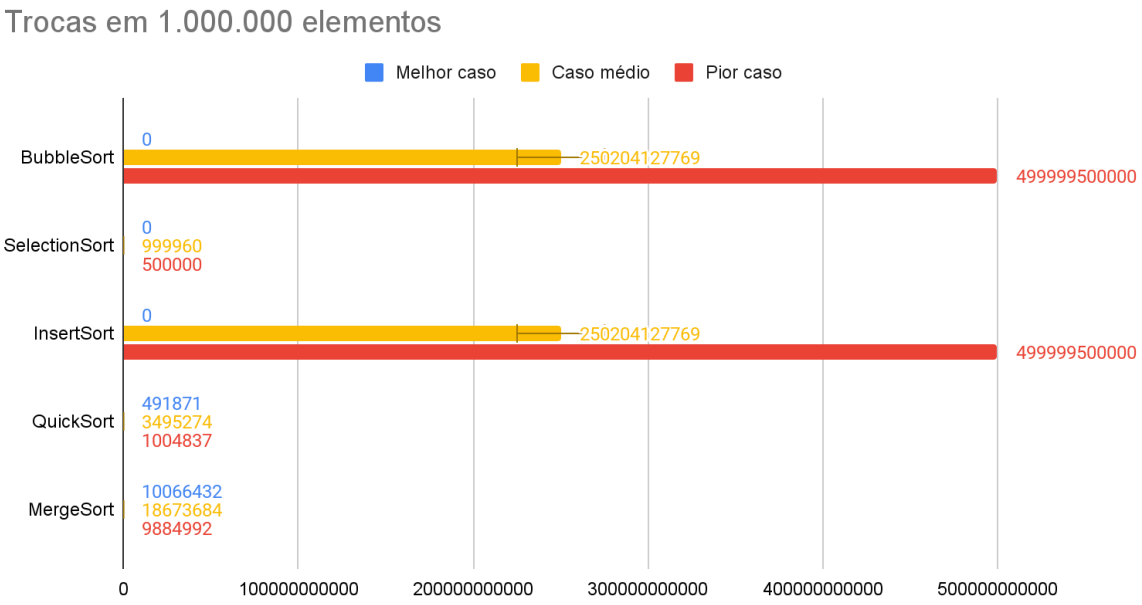


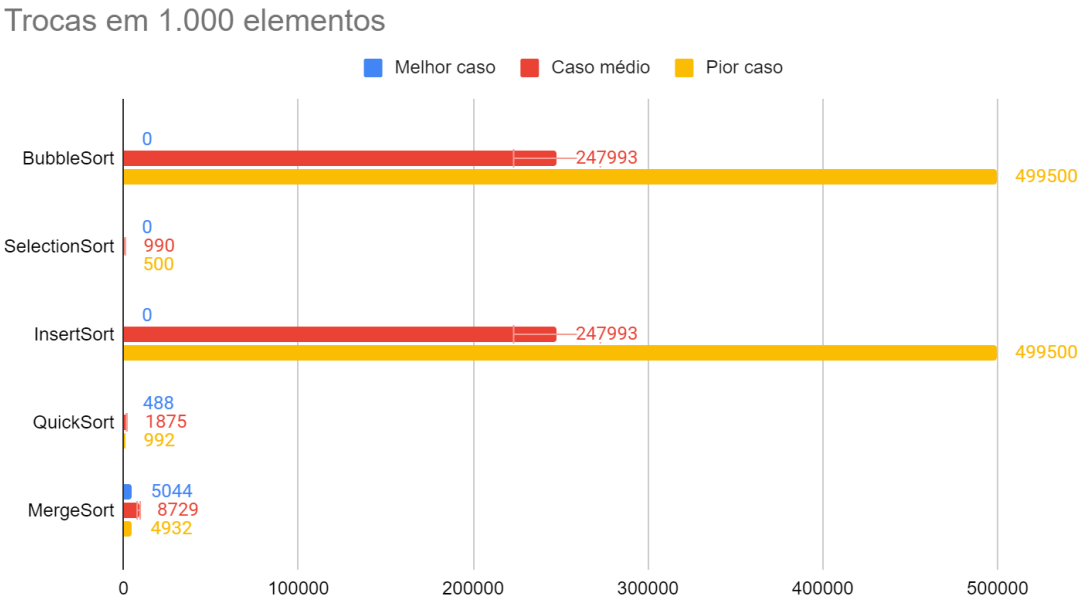
Em um gráfico com apenas eles é visível a diferença, onde o Merge faz todos resultados parecidos e com um tempo muito menor comparado aos outros métodos, mas quando fica lado a lado com o Quick fica muito aparente que o Quick é muito mais rápido(Pelo menos quando se utiliza o número do meio como Pivô).





Em relação às comparações, podemos ver que temos o padrão, onde o bubble e o selection fazem muitas comparações, e o Insert faz menos comparações que o Quick e o Merge, porém como vimos anteriormente demora mais.





Com as trocas é visível que, o bubble e o Insert, tanto no array de 1 milhão quanto no de 1 mil se sobressaem sobre os outros, de uma forma ruim, realizando o mesmo número de trocas, ambos se saem mal, fazendo trocas de mais.

Conclusão

O método Bubble Sort, o mais simples entre todos, se saiu mal, tendo muitas trocas e comparações, sendo sua única exceção quando o array já está ordenado, assim fazendo nenhuma troca e poucas comparações.

O Selection Sort, ele tem sua vantagem e desvantagem de sempre ter o mesmo número de comparações, assim quase sempre tendo o mesmo tempo de execução, mesmo com o array no melhor caso.

O Insert Sort, ele tem sempre o mesmo número de comparações, o tamanho da lista menos 1, porém o número de trocas se iguala ao bubble sort, assim acabando sendo mais lento por conta da quantidade de trocas.

O Merge Sort, ele é um método recursivo que sempre tem o mesmo número de comparações e trocas, resolve o problema praticamente sempre no mesmo tempo, mesmo com ela ordenada ou não.

O Quick Sort, é o mais rápido sem dúvida, ele consegue ter menos comparações e trocas, e mesmo com o array de 1 milhão de posições ele conseguiu fazer em menos de 1 segundo, assim se sobressaindo sobre todos os outros. Se o pivô escolhido fosse outro, sem ser o número do meio, os resultados seriam diferentes. Optei por fazer com o pivô central para o programa não crashar com 1 milhão de elementos.