Matrícula: 1721101043
Nome: Obenson Maurice
Data : 13 de dezembro de 2023
Disciplina: Pesquisa e ordenação de dados(POD)
Professor: Guilherme Dal Bianco
Curso : Ciência Da Computação.

Trabalho de Recuperação

Este relatório visa comparar o desempenho em termos de tempo de execução e número de trocas entre os métodos RadixSort e HeapSort, em relação aos algoritmos abordados no trabalho TP1. O projeto consistiu na implementação de algoritmos de ordenação para conjuntos de tamanhos variados (n = 100, 1000, 5000, 10000, 50000, 100000). Neste caso realizamos os testes com muitos dados e chegando colocar 1000000 em que o processamento não me deu resultado. Desse fato, enquanto que o valor é maior o custo do algoritmo fica alto. Neste trabalho cada algoritmo está sendo representado numa cor diferente que vai ser mostrada nos gráficos seguintes.

Técnicas usadas para desenvolver o trabalho:

Usamos a biblioteca Time.h da linguagem C de uma forma para conseguir contar o tempo e o número de trocas entre os dados em que foi executado em algumas vezes com o tamanho diferente de “n” que representa a entrada. Assim gerou um arquivo CSV com as saídas em termo de trocas e tempo.

Analiticamente temos umas situações que valem a pena destacar

Bubble Sort:

O Bubble Sort demonstrou desempenho inferior em todos os cenários, com tempos de execução consideravelmente mais longos à medida que o tamanho do conjunto de dados aumenta.

Selection Sort:

Assim como o Bubble Sort, o Selection Sort apresenta desempenho limitado, com tempos de execução superiores em comparação com outros algoritmos, especialmente para conjuntos de dados maiores.

Insertion Sort:

O Insertion Sort exibiu um desempenho moderado, sendo mais eficiente que o Bubble e o Selection Sort, mas ainda assim superado por algoritmos mais avançados em termos de eficiência.

QuickSort:

O QuickSort mostrou-se significativamente mais eficiente, com tempos de execução notavelmente baixos em todos os conjuntos de dados. Este algoritmo é conhecido por sua eficácia em lidar com conjuntos grandes.

HeapSort:

O HeapSort também apresentou bom desempenho, especialmente em conjuntos de dados maiores. Sua eficiência contribui para torná-lo uma escolha sólida para ordenação.

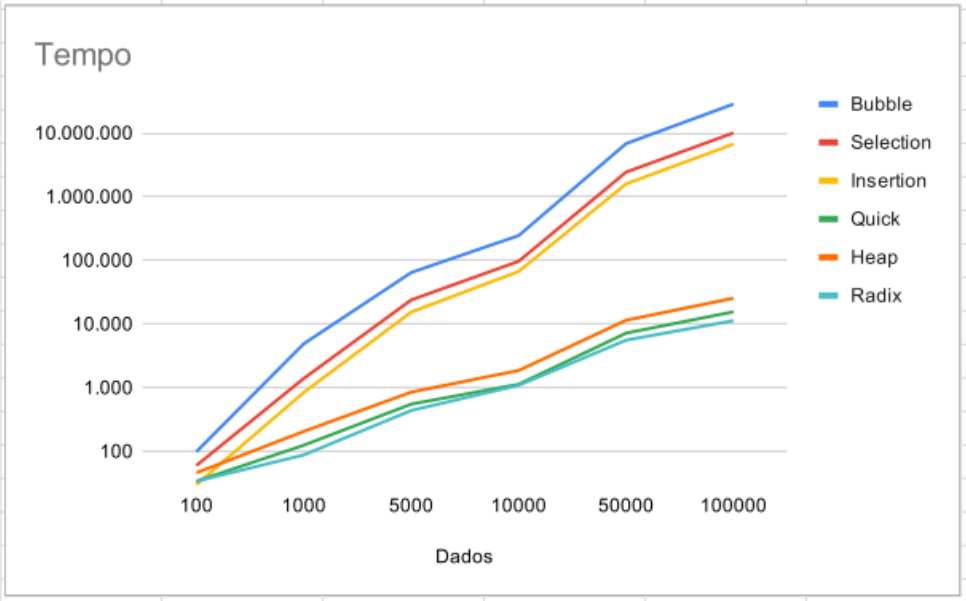
RadixSort:

O Radix Sort demonstrou ser altamente eficiente, com tempos de execução notavelmente baixos, mesmo em conjuntos de dados consideravelmente grandes. Este algoritmo é particularmente eficaz para ordenação de inteiros.

Considerações finais:

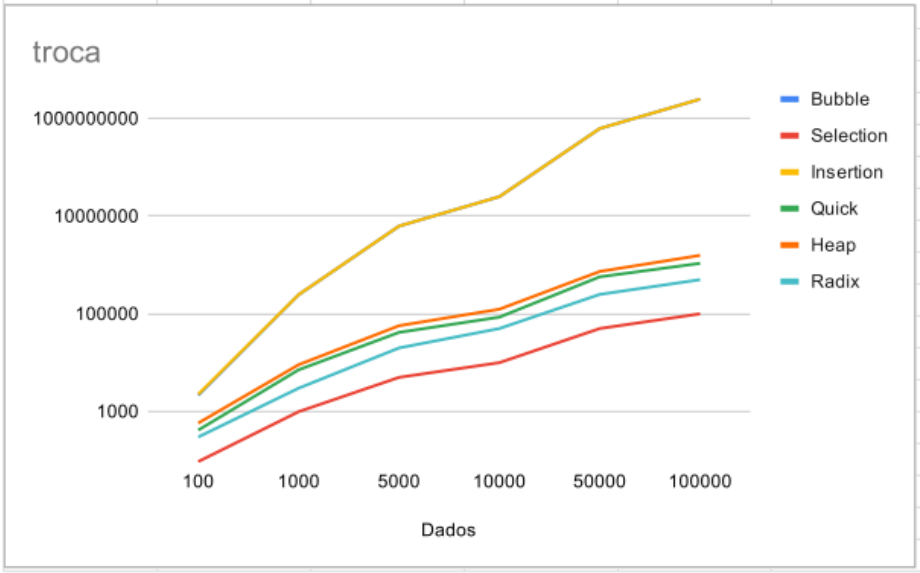
Os algoritmos mais eficientes para diferentes tamanhos de conjuntos de dados são QuickSort, HeapSort e RadixSort. Enquanto o QuickSort é eficaz em geral, o HeapSort e o RadixSort têm vantagens específicas em cenários particulares. Para conjuntos pequenos, algoritmos quadráticos podem ser viáveis, mas para conjuntos grandes, a escolha recai entre HeapSort, QuickSort (ambos com complexidade O(n log n)), e RadixSort (complexidade O(n)). A decisão entre QuickSort e RadixSort depende da natureza dos dados. O Radix é preferível quando a comparação envolve números com dígitos proporcionais ao tamanho do vetor; caso contrário, a escolha é o QuickSort. Mesmo para conjuntos grandes já ordenados, se a necessidade é inserir novos valores, o Insertion é a melhor opção, pois é O(n). Basta dizer que a Insertion Sort é especialmente adequado para inserções em conjuntos que já estão parcialmente ou totalmente ordenados, proporcionando um desempenho mais eficiente nesses casos específicos.

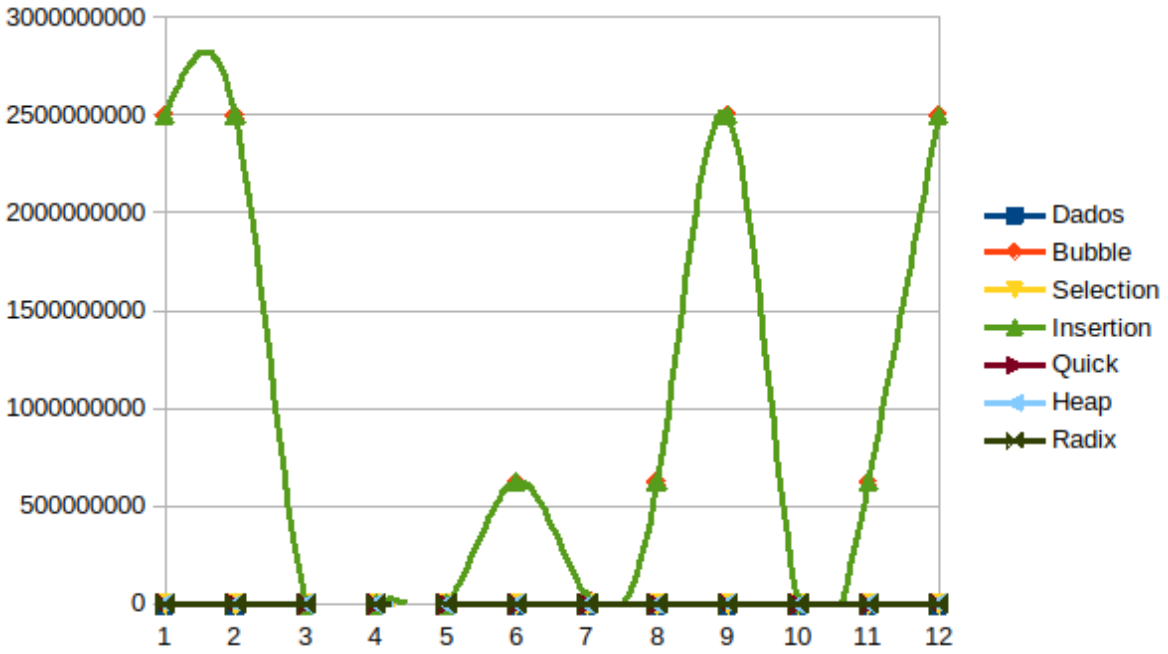
Dados	Bubble	Selection	Insertion	Quick	Heap	Radix
100	99	60	30	34	46	34
1000	4.910	1.400	844	125	206	88
5000	64.433	23.916	15.431	553	853	439
10000	244.947	96.777	66.924	1.131	1.864	1.092
50000	6.823.825	2.431.014	1.588.285	7.200	11.469	5.586
100000	28.484.833	10.044.081	6.707.764	15.569	25.472	11.302



https://github.com/obens/sorting

Dados	Bubble	Selection	Insertion	Quick	Heap	Radix
100	2157	94	2256	415	579	300
1000	246104	990	247103	7128	9074	3000
5000	6255802	4992	6260801	41659	57069	20000
10000	25105880	9992	25115879	85465	124245	50000
50000	624000186	49991	624050185	572271	737334	250000
100000	2496264079	99985	2496364078	1081734	1575422	500000




3000000000
2500000000
2000000000
1500000000
1000000000
500000000
0
1
2
3
4
5
6
7
8
9
10
11
12
Dados
Bubble
Selection
Insertion
Quick
Heap
Radix