UNESP - Faculdade de Ciências - Câmpus de Bauru

Trabalho 1

Alunos
Eric Trofino
Ian Marques Breda
Lucas Yuki Nishimoto
Rafael Piccolomini de Lima

Prof. Dr. João Paulo Papa

Bauru, Junho de 2023

Sumário

1 Algoritmos utilizados e suas características

(Obs: foram usados, para todos os algoritmos, vetores de tamanho aleatório, de 90.000 até 110.000 posições, e de valores aleatórios entre 1 e 10.000. Ao final do relatório é indicado um exemplo dos tempos que foram medidos e uma das situações)

1.1 Merge Sort

Complexidade de tempo: O(n log n) Desempenho esperado: O tempo de execução é independente da ordem do vetor de entrada. Resultados obtidos: O Merge Sort mostrou um desempenho semelhante para os três tipos de vetores, desde que tivessem o mesmo tamanho. Isso ocorre porque o algoritmo divide o vetor em subvetores menores, independentemente da ordem dos elementos.

1.2 Bubble Sort

Complexidade de tempo: O(n^2) Desempenho esperado: Vetores aleatórios e ordenados crescentemente podem ter desempenho melhor do que vetores ordenados decrescentemente. Resultados obtidos: O Bubble Sort demonstrou ser o algoritmo mais lento entre os três. Para vetores aleatórios, ele precisava percorrer o vetor várias vezes, resultando em um tempo de processamento mais longo. No entanto, para vetores ordenados crescentemente, ele teve um desempenho um pouco melhor, pois as trocas de elementos foram reduzidas. Já para vetores ordenados decrescentemente, o Bubble Sort foi o mais lento, pois exigia muitas trocas repetidas.

1.3 Quick Sort

Complexidade de tempo: O(n log n) em média, podendo chegar a O(n^2) no pior caso. Desempenho esperado: Desempenho semelhante para todos os tipos de vetores, mas o pior caso pode ocorrer para vetores ordenados. Resultados obtidos: O Quick Sort demonstrou um desempenho semelhante para todos os tipos de vetores. Para vetores aleatórios, ele mostrou um bom desempenho e foi capaz de dividir o vetor em partes menores eficientemente. No entanto, para vetores ordenados crescentemente e decrescentemente, o Quick Sort pode ter um desempenho pior devido à possibilidade de partições desequilibradas. Escolher o pivô de forma inteligente pode ajudar a melhorar o desempenho.

1.4 Heap Sort

Complexidade de tempo: O(n log n) Desempenho esperado: Desempenho semelhante para todos os tipos de vetores. Resultados obtidos: O Heap Sort também mostrou um desempenho semelhante para todos os tipos de vetores. Assim como o Merge Sort, o Heap Sort é um algoritmo eficiente que não depende da ordem inicial dos elementos. Ele requer uma construção inicial do heap, seguida por remoções sucessivas da raiz do heap para obter os elementos em ordem crescente. Portanto, o tempo de execução do Heap Sort foi aproximadamente o mesmo para vetores aleatórios, ordenados crescentemente e ordenados decrescentemente, desde que tenham o mesmo tamanho.

1.5 Selection Sort

Complexidade de tempo: O(n^2) Desempenho esperado: Desempenho semelhante para todos os tipos de vetores. Resultados obtidos: O Selection Sort também mostrou um desempenho menos eficiente quando comparado a outros algoritmos mais avançados. No entanto, o tempo de processamento foi semelhante para os três tipos de vetores - aleatório, ordenado crescentemente e ordenado decrescentemente. Isso ocorre porque o algoritmo sempre busca o menor (ou maior) elemento restante e o coloca em sua posição correta, independentemente da ordem inicial dos elementos.

2 Conclusões

Após avaliar o desempenho dos algoritmos de ordenação - Bubble Sort, Selection Sort, Heap Sort, Quick Sort e Merge Sort - podemos concluir o seguinte: O Bubble Sort e o Selection Sort apresentaram um desempenho menos eficiente em comparação com outros algoritmos mais avançados, como o Heap Sort, Quick Sort e Merge Sort. Ambos possuem uma complexidade de tempo quadrática ($O(n^2)$), resultando em tempos de processamento mais longos para grandes conjuntos de dados. O Heap Sort, Quick Sort e Merge Sort demonstraram desempenho eficiente e semelhante para todos os tipos de vetores avaliados. O Heap Sort e o Merge Sort têm uma complexidade de tempo de O(n log n), enquanto o Quick Sort tem uma complexidade média de O(n log n) e pior caso de $O(n^2)$ para vetores ordenados decrescentemente. A escolha adequada do pivô no Quick Sort pode melhorar seu desempenho. O Merge Sort, em particular, mostrou um desempenho consistente e eficiente para todos os tipos de vetores, independentemente da ordem inicial dos elementos. Ele divide o vetor em subvetores menores, ordena-os de forma independente e, em seguida, mescla-os para obter o vetor final ordenado. Portanto, para um desempenho mais eficiente, recomenda-se utilizar o Heap Sort, Quick Sort ou Merge Sort em vez do Bubble Sort ou Selection Sort. No entanto, é importante considerar o caso específico

3 Tempos medidos

3.1 Merge Sort

Tempo gasto (random) 0 segundos e 12997 micros

Tempo gasto (crescente) 0 segundos e 9000 micros

Tempo gasto (decrescente) 0 segundos e 8002 micros

3.2 Bubble Sort

Tempo gasto (random) 24 segundos e 24027334 micros

Tempo gasto (crescente) 0 segundos e 1444 micros

Tempo gasto (decrescente) 32 segundos e 31998700 micros

3.3 Quick Sort

Tempo gasto (random) 0 segundos e 12999 micros

Tempo gasto (crescente) 0 segundos e 256001 micros

Tempo gasto (decrescente) 1 segundos e 330001 micros

3.4 Heap Sort

Tempo gasto (random) 0 segundos e 18000 micros

Tempo gasto (crescente) 0 segundos e 14000 micros

Tempo gasto (decrescente) 0 segundos e 14002 micros

3.5 Selection Sort

Tempo gasto (random) 7 segundos e 6903999 micros

Tempo gasto (crescente) 7 segundos e 6998000 micros

Tempo gasto (decrescente) 7 segundos e 7245000 micros