Relatório Atividade Prática 01 - ED2

Alunos: Hellen Guterres França e Iaze Guilherme Soares Carneiro Santos

Objetivos

Aplicar os algoritmos de ordenação aprendidos em sala de aula para a sua otimização de acordo com os problemas apresentados nos enunciados.

Usamos o método System.nanotime() para medir o tempo de execução dos algoritmos de ordenação.

Problema 1

Foi implementada uma versão do MergeSort com um corte para subvetores de tamanho menor. Esse corte define que a ordenação será feita com o InsertSort.

O MergeSort é um algoritmo que utiliza a abordagem de divisão e conquista. Ele começa dividindo o vetor inteiro em subvetores cada vez menores até que sejam unitários. Depois, compara seus valores para ordena-los e mescla eles até estarem completamente ordenados. No caso desse problema, na hora que o subvetor atingir um certo tamanho, iremos mudar o algoritmo de ordenaçao para o InsertSort.

Para aumentar a eficiência de 10 a 15% testamos alguns valores de corte. Além disso, adicionamos uma condição antes de mesclar os subvetores para verificar se eles já estão ordenados.

Aqui temos alguns testes de vetores com seu tamanho e tamanho definido para ocorrer o corte

```
Tamanho do vetor: 30
Tamanho do corte: 0
Duration: 3.2113 ms
```

```
Tamanho do vetor: 30
Tamanho do corte: 15
Duration: 0.0766 ms
```

Problema 2

Uma versão modificada do algoritmo selection sort foi implementada. O selection sort normalmente “divide o vetor em dois”: parte ordenada e parte não ordenada, com a parte ordenada começando vazia. Enquanto percorre a parte não ordenada, procura nela o menor elemento e posiciona ao final da parte ordenada. Repetindo o processo tem-se um vetor ordenado de forma crescente.
No algoritmo modificado apresentado, isso acontece de forma diferente. Tem-se o processo habitual do algoritmo selection sort com um detalhe adicional: a ordenação também ocorre de trás para frente, ou seja, um processo parecido com o de ordenar o vetor escolhendo o menor e colocando no final do vetor ordenado acontece quando também percorremos o vetor de trás pra frente procurando o maior elemento e posicionando ao final do vetor original que podemos enxergar como uma terceira divisão ordenada do vetor, ou seja, estamos colocando no início dessa terceira divisão, se enxergarmos ela como um vetor, o maior elemento da divisão não ordenada. Assim como no selection sort normal, repetir esse processo resulta em um vetor ordenado.
Vale ressaltar que o processo de encontrar o maior e o menor elemento ocorre no mesmo laço for. Ainda, o laço que procura o maior e o menor procura apenas na parte não ordenada do vetor.
Já que o processo de ordenação começa pelo início e pelo fim do vetor ao mesmo tempo, o laço externo executa apenas n/2 vezes.

Cálculo do tempo de execução:

```
Tamanho da entrada: 100 itens
Tempo de execução com o selectionSort modificado: 0.6986 milissegundos
Tempo de execução com o selectionSort usual: 0.6392 milissegundos
```

Ao final, obtém-se um vetor ordenado.

```
Vetores antes da ordenação:
Select modificado        |        Select Habitual
      9931               |              9931
      5748               |              5748
      4369               |              4369
      7190               |              7190
      121               |              121
      4571               |              4571
      5518               |              5518
      3280               |              3280
      1449               |              1449
      6754               |              6754
Tamanho da entrada: 10 itens
Tempo de execução com o selectionSort modificado: 0.3874 milissegundos
Tempo de execução com o selectionSort usual: 0.252601 milissegundos
Vetores depois da ordenação:
Select modificado        |        Select Habitual
      121               |              121
      1449               |              1449
      3280               |              3280
      4369               |              4369
      4571               |              4571
      5518               |              5518
      5748               |              5748
      6754               |              6754
      7190               |              7190
      9931               |              9931
```

Problema 3

O QuickSort é um algoritmo de ordenação que segue a ideia de dividir o problema em problemas menores. Temos um vetor arbitrário pivo, que nesse caso usamos a mediana de tres, e vamos organizando o vetor de acordo com as comparações feitas com ele. Os valores menores que o pivô ficam a esquerda e os maiores ficam a direita, recursivamente partindo o vetor em subvetores. O bubbleSort segue a ideia de ao percorrer o vetor comparar em pares qual é o maior e levá-lo para o fim do vetor. Para esse problema, testamos valores de tamanho do subvetor que melhoram o tempo de execução. Quando o subvetor tiver determinado tamanho, a ordenaçao passa a ser feita com o BubbleSort.

```
Tamanho do vetor: 100
Tamanho do corte: 50
Duration: 2.6413 ms
```

```
Tamanho do vetor: 100
Tamanho do corte: 25
Duration: 0.4527 ms
```

```
Tamanho do vetor: 100
Tamanho do corte: 20
Duration: 3.9173 ms

Tamanho do vetor: 100
Tamanho do corte: 25
Duration: 0.3717 ms
```

```
Tamanho do vetor: 100
Tamanho do corte: 30
Duration: 2.9313 ms

Tamanho do vetor: 100
Tamanho do corte: 25
Duration: 0.5492 ms
```

```
Tamanho do vetor: 1000
Tamanho do corte: 500
Duration: 8.4652 ms

Tamanho do vetor: 1000
Tamanho do corte: 250
Duration: 2.1423 ms
```

```
Tamanho do vetor: 1000
Tamanho do corte: 200
Duration: 9.1512 ms

Tamanho do vetor: 1000
Tamanho do corte: 250
Duration: 2.9834 ms
```

```
Tamanho do vetor: 10000
Tamanho do corte: 5000
Duration: 148.1237 ms

Tamanho do vetor: 10000
Tamanho do corte: 2500
Duration: 139.7861 ms
```

```
Tamanho do vetor: 10000
Tamanho do corte: 5000
Duration: 172.0938 ms

Tamanho do vetor: 10000
Tamanho do corte: 2000
Duration: 113.3063 ms
```

```
Tamanho do vetor: 10000
Tamanho do corte: 5000
Duration: 141.9041 ms

Tamanho do vetor: 10000
Tamanho do corte: 1000
Duration: 109.768 ms
```

```
Tamanho do vetor: 100000
Tamanho do corte: 50000
Duration: 21374.983 ms

Tamanho do vetor: 100000
Tamanho do corte: 10000
Duration: 15369.1901 ms
```

```
Tamanho do vetor: 100000
Tamanho do corte: 50000
Duration: 20628.8253 ms

Tamanho do vetor: 100000
Tamanho do corte: 25000
Duration: 15984.1084 ms
```

```
Tamanho do vetor: 100000
Tamanho do corte: 50000
Duration: 20449.9373 ms

Tamanho do vetor: 100000
Tamanho do corte: 20000
Duration: 15774.1392 ms
```

```
Tamanho do vetor: 100000
Tamanho do corte: 50000
Duration: 20667.0131 ms

Tamanho do vetor: 100000
Tamanho do corte: 10000
Duration: 15244.8604 ms
```

Observamos que para 100, o valor de corte funciona melhor para o 25
Para o 1000, o 250
para o 10.000, o 2000
para o 100.000, o os valores de 25000 ate 10000 nao tem muita variação
para o 1.000.000, não foi possivel testar pois a maquina nao rodou

Problema 4

O heapsort consiste na construção de uma estrutura (heap) e logo em seguida a ordenação do vetor com base na estrutura construída. O heap é uma árvore binária que guarda os valores do vetor seguindo uma lógica. Se for um min-heap, para todo nó da árvore, se ele possui filhos, os filhos são sempre maiores que ele. Já no max-heap, todos os nós filhos são sempre menores que o pai. No algoritmo implementado, são construídas duas estruturas heap: um max e um min heap, e em seguida ordena-se o vetor pela esquerda e pela direita alternadamente. A seguir, o tempo de execução do algoritmo:

```
Tamanho da entrada: 100 itens
Tempo de execução com o heapsort modificado: 0.207799 milissegundos
Tempo de execução com o heapsort usual: 0.9207 milissegundos
```

```
Tamanho da entrada: 10000 itens
Tempo de execução com o heapsort modificado: 11.320499 milissegundos
Tempo de execução com o heapsort usual: 6.5163 milissegundos
```

Ao final, um vetor ordenado:

```
Vetores antes da ordenação:
heap modificado        |        heap Habitual
      9931              |              9931
      5748              |              5748
      4369              |              4369
      7190              |              7190
      121               |             121
      4571              |              4571
      5518              |              5518
      3280              |              3280
      1449              |              1449
      6754              |              6754
Tamanho da entrada: 10 itens
Tempo de execução com o heapsort modificado: 0.0267 milissegundos
Tempo de execução com o heapsort usual: 0.3622 milissegundos
Vetores depois da ordenação:
heap modificado        |        heap Habitual
      121               |             121
      1449              |              1449
      3280              |              3280
      4369              |              4369
      4571              |              4571
      5518              |              5518
      5748              |              5748
      6754              |              6754
      7190              |              7190
      9931              |              9931
```