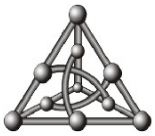


UNIVERSIDADE FEDERAL DE MATO GROSSO DO SUL
Faculdade de Computação

Implementação Algorítmica

Atividade 1 — Algoritmos de Ordenação

1 Descrição

Ordenação é uma operação básica em Computação. Diversos algoritmos de ordenação têm sido propostos ao longo da história, mesmo quando computadores ainda não existiam. A partir de meados do século passado, os algoritmos começaram a ser transformados em programas e executados em computadores. Esta atividade pede que você implemente os algoritmos de ordenação que estudamos e realize experimentos com os mesmos.

Em particular, você deve implementar os seguintes algoritmos de ordenação: BUBBLESORT, INSERTIONSORT, MERGESORT, HEAPSORT, QUICKSORT e COUNTINGSORT.

Dessa forma, você deve implementar 6 algoritmos de ordenação listados acima.

2 Programa, entrada e saída

Considere os seguintes parâmetros para execução dos experimentos, que são fornecidos como entrada:

- **`inc`** é o tamanho inicial de um vetor de entrada;
- **`fim`** é o tamanho final;
- **`stp`** é o intervalo entre dois tamanhos; e
- **`rpt`** é o número de repetições a serem realizadas.

Para realizar experimentos com esses algoritmos, você deve construir 4 tipos de conjuntos de dados de entrada, conforme a descrição a seguir:

1. **Vetor aleatório**: cada conjunto de entrada A deve conter números inteiros não negativos escolhidos (pseudo)aleatoriamente no intervalo $[0, n^2]$. Um tal conjunto de números A deve ter n números. Os valores de n variam, para cada conjunto construído, de acordo com os parâmetros **`inc`** (valor inicial), **`fim`** (valor final) e **`stp`** (intervalo entre dois tamanhos). Por exemplo, se **`inc`** $= 100$, **`fim`** $= 1000$ e **`stp`** $= 10$, então os tamanhos dos vetores de entrada A que devem ser construídos são $n = 100, 110, 120, \ldots, 990, 1000$.

   Um conjunto A com n elementos assim gerado é chamado de **caso de teste**. Para cada caso de teste, você deve executar os seis algoritmos mencionados na Seção 1 para ordenar esse conjunto A de n números inteiros. Para cada n, você deve repetir o processo acima um determinado número **`rpt`** de vezes, obtendo então a média dos tempos medidos.

A saída deste experimento consiste de uma primeira linha contendo o rótulo **`[[RANDOM]]`**, especificando o conjunto de dados sendo usado, uma segunda linha contendo os rótulos da quantidade de elementos n e a identificação de cada um dos algoritmos. Cada linha a seguir contém o valor da quantidade de elementos n de um caso de teste e as médias dos tempos gastos da execução de cada algoritmo.

2. **Vetor reverso**: cada conjunto de entrada A deve conter números inteiros não negativos, arranjados em ordem decrescente. Um tal conjunto de números A deve ter n números: $n, n-1, n-2, \ldots, 1$. Os valores de n variam, para cada conjunto construído, de acordo com os parâmetros **`inc`** (valor inicial), **`fim`** (valor final) e **`stp`** (intervalo entre dois tamanhos), assim como descrito no caso anterior (**vetor aleatório**).

   Para cada caso de teste, você deve executar os seis algoritmos mencionados na seção 1 para ordenar esse conjunto A de n números inteiros. Neste caso, **não é necessário repetir a execução dos algoritmos**, isto é, cada caso de teste é executado uma única vez, obtendo então os tempos medidos.

   A saída deste experimento consiste de uma primeira linha contendo o rótulo **`[[REVERSE]]`**, especificando o conjunto de dados sendo usado, uma segunda linha contendo os rótulos da quantidade de elementos n e a identificação de cada um dos algoritmos. Cada linha a seguir contém o valor da quantidade de elementos n de um caso de teste e os tempos gastos da execução de cada algoritmo.

3. **Vetor ordenado**: cada conjunto de entrada A deve conter números inteiros não negativos, arranjados em ordem crescente. Um tal conjunto de números A deve ter n números: $1, 2, 3, \ldots, n$. Os valores de n variam, para cada conjunto construído, de acordo com os parâmetros **`inc`** (valor inicial), **`fim`** (valor final) e **`stp`** (intervalo entre dois tamanhos), assim como descrito nos dois casos anteriores.

   Para cada caso de teste, você deve executar os seis algoritmos mencionados na seção 1 para ordenar esse conjunto A de n números inteiros. Neste caso, **não é necessário repetir a execução dos algoritmos**, isto é, cada caso de teste é executado uma única vez, obtendo então os tempos medidos.

   A saída deste experimento consiste de uma primeira linha contendo um rótulo **`[[SORTED]]`**, especificando o conjunto de dados sendo usado, uma segunda linha contendo os rótulos da quantidade de elementos n e a identificação de cada um dos algoritmos. Cada linha a seguir contém o valor da quantidade de elementos n de um caso de teste e os tempos gastos da execução de cada algoritmo.

4. **Vetor quase ordenado**: cada conjunto de entrada A deve conter n números inteiros não negativos no intervalo $[0, n^2]$, escolhidos (pseudo)aleatoriamente, e "quase" ordenado crescentemente. Para obter um vetor dessa forma, você deve arranjar o vetor crescentemente, escolher 10% dos seus elementos e então embaralhá-los. Um tal conjunto de números A deve ter n números. Os valores de n variam, para cada conjunto construído, de acordo com os parâmetros **`inc`** (valor inicial), **`fim`** (valor final) e **`stp`** (intervalo entre dois tamanhos), assim como descrito nos dois casos anteriores.

Para cada caso de teste, você deve executar os seis algoritmos mencionados na seção 1 para ordenar esse conjunto A de n números inteiros. Neste caso, **não é necessário repetir a execução dos algoritmos**, isto é, cada caso de teste é executado uma única vez, obtendo então os tempos medidos.

A saída deste experimento consiste de uma primeira linha contendo o rótulo **`[[NEARLY SORTED]]`**, especificando o conjunto de dados sendo usado, uma segunda linha contendo os rótulos da quantidade de elementos n e a identificação de cada um dos algoritmos. Cada linha a seguir contém o valor da quantidade de elementos n de um caso de teste e os tempos gastos da execução de cada algoritmo.

2.1 Exemplo de entrada e saída

Um exemplo de execução, para casos de teste com n variando de acordo com os parâmetros **`inc`** $= 1000$, **`fim`** $= 20000$ e **`stp`** $= 1000$, é mostrado a seguir. O parâmetro relativo ao número de repetições foi adotado como **`rpt`** $= 10$. As médias dos tempos de execução (no caso do experimento aleatório) e os tempos de execução (no caso dos outros experimento) dos algoritmos são mostrados em segundos.

```
[[RANDOM]]
     n     Bubble    Insertion     Merge        Heap       Quick      Counting
-------------------------------------------------------------------------------
  1000    0.001349    0.000657    0.000265    0.000179    0.000095    0.000125
  2000    0.004430    0.002466    0.000519    0.000401    0.000202    0.000137
  3000    0.009835    0.005533    0.000769    0.000615    0.000314    0.000124
  4000    0.017442    0.009807    0.001041    0.000851    0.000428    0.000126
  5000    0.027168    0.015133    0.001312    0.001092    0.000548    0.000134
  6000    0.039032    0.021433    0.001582    0.001339    0.000668    0.000151
  7000    0.053161    0.029577    0.001865    0.001587    0.000788    0.000149
  8000    0.069235    0.038900    0.002134    0.001848    0.000906    0.000159
  9000    0.087501    0.049461    0.002438    0.002121    0.001038    0.000169
 10000    0.108018    0.060918    0.002717    0.002364    0.001163    0.000187
 11000    0.130686    0.072739    0.003011    0.002638    0.001307    0.000180
 12000    0.155325    0.086640    0.003284    0.002897    0.001414    0.000188
 13000    0.182383    0.102143    0.003576    0.003171    0.001558    0.000229
 14000    0.214547    0.121038    0.003947    0.003509    0.001741    0.000213
 15000    0.243412    0.136780    0.004170    0.003721    0.001811    0.000256
 16000    0.281862    0.159071    0.004527    0.004020    0.001960    0.000276
 17000    0.315802    0.179878    0.004797    0.004346    0.002088    0.000232
 18000    0.363552    0.210642    0.005256    0.004680    0.002331    0.000241
 19000    0.401789    0.225040    0.005392    0.004972    0.002350    0.000244
 20000    0.443572    0.248571    0.005777    0.005260    0.002510    0.000250
```

```
[[REVERSE]]
     n    Bubble   Insertion    Merge        Heap       Quick      Counting
-------------------------------------------------------------------------------
  1000    0.000107    0.000121    0.000022    0.000015    0.000007    0.000010
  2000    0.000425    0.000487    0.000049    0.000046    0.000015    0.000020
  3000    0.000978    0.001102    0.000067    0.000052    0.000023    0.000014
  4000    0.001704    0.001975    0.000093    0.000072    0.000032    0.000012
  5000    0.002677    0.003026    0.000112    0.000093    0.000039    0.000013
  6000    0.004017    0.004415    0.000138    0.000113    0.000048    0.000014
  7000    0.005348    0.006190    0.000200    0.000162    0.000058    0.000034
  8000    0.007416    0.008080    0.000189    0.000156    0.000068    0.000018
  9000    0.009365    0.010001    0.000215    0.000182    0.000077    0.000020
 10000    0.012034    0.012198    0.000231    0.000201    0.000089    0.000017
 11000    0.015342    0.015851    0.000269    0.000263    0.000097    0.000018
 12000    0.019094    0.017795    0.000279    0.000247    0.000102    0.000018
 13000    0.023595    0.020952    0.000304    0.000272    0.000109    0.000019
 14000    0.028797    0.024271    0.000329    0.000296    0.000118    0.000019
 15000    0.033354    0.028214    0.000365    0.000313    0.000130    0.000025
 16000    0.040336    0.031296    0.000382    0.000334    0.000131    0.000023
 17000    0.045078    0.035136    0.000404    0.000363    0.000148    0.000021
 18000    0.054619    0.040024    0.000464    0.000392    0.000170    0.000022
 19000    0.061757    0.047838    0.000574    0.000442    0.000193    0.000023
 20000    0.073058    0.050572    0.000629    0.000468    0.000182    0.000024
```

```
[[SORTED]]
     n    Bubble   Insertion    Merge        Heap       Quick      Counting
-------------------------------------------------------------------------------
  1000    0.000108    0.000000    0.000022    0.000017    0.000007    0.000010
  2000    0.000552    0.000003    0.000055    0.000036    0.000014    0.000012
  3000    0.001045    0.000001    0.000067    0.000056    0.000021    0.000013
  4000    0.001845    0.000001    0.000095    0.000103    0.000052    0.000025
  5000    0.003045    0.000002    0.000123    0.000108    0.000038    0.000014
  6000    0.004040    0.000002    0.000145    0.000143    0.000048    0.000014
  7000    0.005502    0.000002    0.000160    0.000157    0.000060    0.000017
  8000    0.007079    0.000003    0.000189    0.000189    0.000064    0.000015
  9000    0.008757    0.000003    0.000206    0.000190    0.000072    0.000016
 10000    0.010742    0.000003    0.000230    0.000217    0.000077    0.000017
 11000    0.013653    0.000004    0.000258    0.000252    0.000087    0.000017
 12000    0.016020    0.000004    0.000278    0.000264    0.000101    0.000018
 13000    0.018336    0.000004    0.000325    0.000345    0.000107    0.000019
 14000    0.022547    0.000005    0.000328    0.000323    0.000128    0.000019
 15000    0.024587    0.000005    0.000351    0.000334    0.000118    0.000020
 16000    0.028075    0.000005    0.000397    0.000359    0.000128    0.000021
 17000    0.033364    0.000006    0.000463    0.000436    0.000235    0.000052
 18000    0.036224    0.000006    0.000446    0.000428    0.000152    0.000022
 19000    0.038899    0.000006    0.000448    0.000439    0.000163    0.000023
 20000    0.043674    0.000007    0.000472    0.000462    0.000172    0.000023
```

```
[[NEARLY SORTED]]
     n    Bubble   Insertion    Merge        Heap       Quick      Counting
-------------------------------------------------------------------------------
  1000    0.000109    0.000018    0.000022    0.000018    0.000007    0.000010
  2000    0.000436    0.000083    0.000045    0.000036    0.000016    0.000017
  3000    0.000977    0.000178    0.000069    0.000058    0.000024    0.000012
  4000    0.001718    0.000315    0.000092    0.000079    0.000034    0.000013
  5000    0.002683    0.000487    0.000115    0.000102    0.000044    0.000013
  6000    0.003866    0.000731    0.000140    0.000125    0.000055    0.000014
  7000    0.005268    0.001016    0.000164    0.000147    0.000062    0.000015
  8000    0.006935    0.001323    0.000189    0.000180    0.000075    0.000016
  9000    0.008736    0.001700    0.000213    0.000194    0.000081    0.000016
 10000    0.010746    0.002041    0.000240    0.000218    0.000094    0.000017
 11000    0.013051    0.002411    0.000262    0.000242    0.000099    0.000017
 12000    0.015461    0.002870    0.000286    0.000266    0.000107    0.000018
 13000    0.018250    0.003442    0.000315    0.000294    0.000123    0.000022
 14000    0.021121    0.004009    0.000338    0.000316    0.000131    0.000019
 15000    0.024185    0.004550    0.000365    0.000340    0.000137    0.000020
 16000    0.027491    0.005107    0.000388    0.000369    0.000153    0.000021
 17000    0.031025    0.005735    0.000415    0.000391    0.000160    0.000021
 18000    0.034784    0.006482    0.000438    0.000413    0.000176    0.000022
 19000    0.038924    0.007034    0.000465    0.000439    0.000197    0.000023
 20000    0.043165    0.007990    0.000488    0.000468    0.000201    0.000046
```

Você deve escolher os valores dos parâmentros **`inc`**, **`fim`**, **`stp`** e **`rpt`** para fazer os experimen-

tos. Faça testes para saber os valores adequados de acordo com o computador em que você irá executar.

3 Entrega

A atividade deve ser feita, **preferencialmente**, em duplas. Na primeira linha do código-fonte, coloque o nome completo de ambos como comentário. **Apenas um dos discentes deve submeter o trabalho no AVA**.

Instruções para entrega do seu trabalho:

1. **O que entregar?**

   Um arquivo compactado a ser entregue deve conter o seguinte:

   - Programa desenvolvido (código-fonte);
   - Tabelas dos tempos de execução dos algoritmos (em texto), de acordo com a formatação apresentada na Seção 2.1; e
   - Um relatório em `.pdf` contendo gráficos gerados a partir das tabelas dos tempos de execução dos algoritmos. Exemplos relativos às tabelas da Seção 2.1 sobre o experimento "vetor aleatório", usando o software `gnuplot`, são mostrados na Figura 1. Você pode utilizar qualquer outro software de sua preferência para gerar os gráficos. No relatório, faça uma discussão sobre os resultados obtidos.

   Compacte todos esses arquivos com o compactador de sua preferência e entregue um único arquivo (com extensão `.tgz`, `.bz2`, `.zip`, `.rar`, ...).

2. **Forma de entrega**

   A entrega será realizada diretamente no Sistema (AVA/UFMS), na disciplina de Implementação Algorítmica – T01. Você pode entregar a atividade quantas vezes quiser até às **23 horas e 59 minutos** do dia **12 de setembro de 2024**. A última versão entregue é aquela que será corrigida. Encerrado o prazo, trabalhos não serão mais aceitos.

3. **Linguagem de programação**

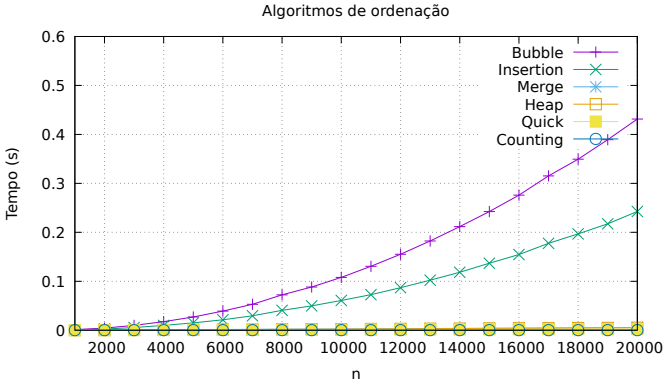
   O programa deve ser implementado em uma das seguintes linguagens: C/C++, Python ou Java.

4. **Erros**

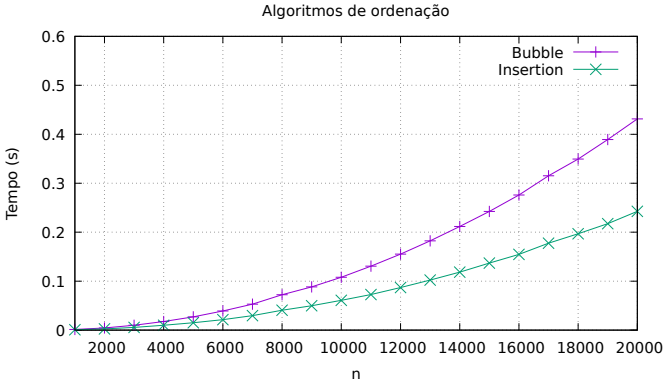
   Trabalhos com erros de compilação/interpretação receberão nota ZERO. Faça todos os testes necessários para garantir que seu programa está livre de erros de compilação/interpretação.

5. **Arquivo com o programa fonte**

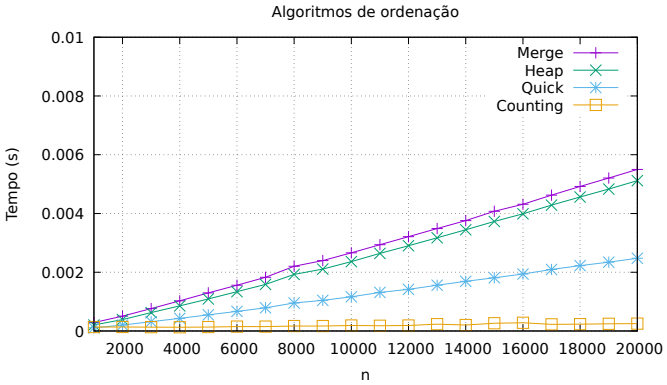
   Seu(s) arquivo(s) contendo o(s) fonte(s) do(s) programa(s) na linguagem escolhida deve(m) estar bem organizado(s). Um programa tem de ser muito bem compreendido por uma pessoa. Verifique se seu programa tem a indentação adequada, se não tem linhas muito longas,



(a) Execução de todos os algoritmos para os casos de teste do experimento "vetor aleatório".



(b) Algoritmos elementares.



(c) Algoritmos eficientes.

Figura 1: Execução dos algoritmos para o experimento "vetor aleatório". Observe que na Figura 1a, as linhas dos tempos de execução dos algoritmos eficientes e de tempo linear (CountingSort) ficam sobrepostas e não conseguimos distingui-las. Por isso, dois outros gráficos são mostrados: Figuras 1b e 1c, apresentando os desempenhos dos algoritmos neste experimento divididos em algoritmos elementares e algoritmos mais eficientes, respectivamente.

se tem variáveis com nomes significativos, entre outros. Não esqueça que um programa bem descrito e bem organizado é a chave de seu sucesso.

6. **Critérios de correção**

   Serão atribuídos até $5{,}0$ pontos para a implementação e até $5{,}0$ pontos para o relatório. Possíveis penalizações que podem diminuir a pontuação:

   - Implementação incorreta;
   - Códigos mal escritos ou desorganizados. Por exemplo, códigos com trechos repetitivos que executam a mesma tarefa;
   - Realizar os experimentos de maneira incorreta;
   - Gráficos pouco legíveis;
   - Incluir informações incorretas no relatório;
   - Não entregar algo que foi solicitado.

7. **Conduta Ética**

   O trabalho deve ser feito INDIVIDUALMENTE/COM SEU GRUPO. Cada estudante tem responsabilidade sobre cópias de seu trabalho, mesmo que parciais. Não compartilhe seu programa ou trechos de seu programa. Você pode consultar seus colegas para esclarecer dúvidas e discutir idéias sobre o trabalho, ao vivo ou no fórum de discussão da disciplina, mas NÃO copie o programa!

   Trabalhos considerados plagiados terão nota ZERO.