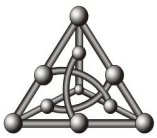


UNIVERSIDADE FEDERAL DE MATO GROSSO DO SUL
Faculdade de Computação

Implementação Algorítmica

Atividade 3 — Seleção do k-ésimo menor elemento

1 Descrição

Encontrar o elemento mínimo ou máximo em um conjunto de números é uma tarefa computacional básica que aprendemos resolver na primeira disciplina de programação de um curso superior da área da Computação. Nesta atividade, devemos solucionar uma generalização desse problema, conforme estudamos nas aulas:

> **Problema da seleção do k-ésimo menor elemento**: dados um conjunto A com n números inteiros e um número inteiro k, onde $1 \leq k \leq n$, encontrar o elemento x em A que é maior ou igual a exatamente $k - 1$ elementos de A.

Você deverá implementar quatro estratégias para solucionar tal problema, conforme descrição a seguir. Além disso, você deve medir os tempos de execução desses algoritmos e compará-los. Para realizar essa tarefa, será necessário gerar conjuntos de dados para realização de experimentos dessas estratégias.

Você deve implementar as seguintes estratégias para resolver o problema da seleção do k-ésimo menor elemento em um vetor A:

1. Ordene o vetor A e devolva o elemento na posição k. Este é o k-ésimo menor elemento do vetor A;
2. Construa uma lista de min-prioridades no vetor A. Remova os k primeiros elementos dessa lista de min-prioridades, imprimindo o valor do último elemento removido. Esse elemento é o k-ésimo menor elemento de A;
3. Faça a seleção aleatorizada. Isto é, execute o algoritmo RANDOMIZED-SELECT para encontrar o k-ésimo menor elemento de A;
4. Faça a seleção (de tempo de execução pior caso linear). Ou seja, execute o algoritmo SELECT para encontrar o k-ésimo menor elemento de A.

Observe que você pode escolher qualquer algoritmo de ordenação por comparação que estudamos para implementar a estratégia 1.

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. 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 $n = 100, 110, 120, \ldots, 990, 1000$. Suponha que o valor máximo possível é 20000.

   Em seguida, você deve sortear um número k no intervalo de 1 a n. Um conjunto A com n elementos e um número k, com $1 \leq k \leq n$, assim gerados são chamados de **caso de teste**. Para cada caso de teste, você deve executar os quatro algoritmos mencionados na seção 1 para selecionar o k-ésimo menor elemento desse 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 escolhidos (pseudo)aleatoriamente e arranjados em ordem decrescente. 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`** $=$ 10, **`fim`** $=$ 100 e **`stp`** $=$ 5, então os tamanhos dos vetores de entrada A que devem ser construídos são $n = 10, 15, 20, \ldots, 95, 100$. Suponha que o valor máximo possível é 20000.

   Em seguida, você deve sortear um número k no intervalo de 1 a n. Um conjunto A com n elementos e um número k, com $1 \leq k \leq n$, assim gerados são chamados de **caso de teste**. Para cada caso de teste, você deve executar os quatro algoritmos mencionados na seção 1 para selecionar o k-ésimo menor elemento desse 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 **`[[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 escolhidos (pseudo)aleatoriamente e arranjados em ordem crescente. 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`** $= 1000$, **`fim`** $= 10000$ e **`stp`** $= 1000$, então os tamanhos dos vetores de entrada A que devem ser construídos são $n = 1000, 2000, 3000, \ldots, 9000, 10000$. Suponha que o valor máximo possível é 20000.

   Em seguida, você deve sortear um número k no intervalo de 1 a n. Um conjunto A com n elementos e um número k, com $1 \leq k \leq n$, assim gerados são chamados de **caso de teste**. Para cada caso de teste, você deve executar os quatro algoritmos mencionados na seção 1 para selecionar o k-ésimo menor elemento desse 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 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 números inteiros não negativos 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). Por exemplo, se **`inc`** $= 200$, **`fim`** $= 20000$ e **`stp`** $= 200$, então os tamanhos dos vetores de entrada A que devem ser construídos são $n = 200, 400, 600, \ldots, 19800, 20000$. Suponha que o valor máximo possível é 20000.

   Em seguida, você deve sortear um número k no intervalo de 1 a n. Um conjunto A com n elementos e um número k, com $1 \leq k \leq n$, assim gerados são chamados de **caso de teste**. Para cada caso de teste, você deve executar os quatro algoritmos mencionados na seção 1 para selecionar o k-ésimo menor elemento desse 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 **`[[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 dos algoritmos são mostradas em segundos. O algoritmo de ordenação escolhido foi o QUICKSORT.

```
[[RANDOM]]
     n  QuickSort   PrQueue   RSelect    Select
-----------------------------------------------
  1000   0.000031  0.000033  0.000008  0.000096
  2000   0.000029  0.000049  0.000004  0.000034
  3000   0.000047  0.000042  0.000009  0.000052
  4000   0.000061  0.000033  0.000005  0.000067
  5000   0.000090  0.000061  0.000030  0.000102
  6000   0.000102  0.000144  0.000012  0.000099
  7000   0.000120  0.000029  0.000022  0.000114
  8000   0.000139  0.000189  0.000037  0.000165
  9000   0.000156  0.000165  0.000025  0.000142
 10000   0.000165  0.000201  0.000028  0.000170
 11000   0.000184  0.000178  0.000025  0.000167
 12000   0.000208  0.000202  0.000040  0.000195
 13000   0.000221  0.000140  0.000033  0.000196
 14000   0.000245  0.000448  0.000021  0.000212
 15000   0.000251  0.000336  0.000029  0.000236
 16000   0.000265  0.000307  0.000035  0.000256
 17000   0.000292  0.000421  0.000054  0.000266
 18000   0.000316  0.000278  0.000039  0.000301
 19000   0.000334  0.000078  0.000071  0.000294
 20000   0.000356  0.000385  0.000053  0.000305
```

```
[[REVERSE]]
     n  QuickSort   PrQueue   RSelect    Select
-----------------------------------------------
  1000   0.000010  0.000015  0.000001  0.000017
  2000   0.000021  0.000019  0.000006  0.000030
  3000   0.000034  0.000029  0.000007  0.000043
  4000   0.000044  0.000075  0.000007  0.000063
  5000   0.000065  0.000128  0.000006  0.000073
  6000   0.000072  0.000019  0.000002  0.000092
  7000   0.000084  0.000158  0.000009  0.000100
  8000   0.000104  0.000149  0.000011  0.000121
  9000   0.000128  0.000216  0.000019  0.000130
 10000   0.000136  0.000241  0.000015  0.000147
 11000   0.000135  0.000172  0.000019  0.000156
 12000   0.000156  0.000063  0.000014  0.000173
 13000   0.000174  0.000383  0.000014  0.000189
 14000   0.000194  0.000234  0.000036  0.000204
 15000   0.000194  0.000085  0.000025  0.000213
 16000   0.000214  0.000118  0.000028  0.000237
 17000   0.000227  0.000096  0.000011  0.000240
 18000   0.000238  0.000211  0.000040  0.000255
 19000   0.000262  0.000392  0.000040  0.000275
 20000   0.000273  0.000608  0.000031  0.000302
```

```
[[SORTED]]
     n  QuickSort   PrQueue   RSelect    Select
-------------------------------------------------
  1000   0.000009  0.000007  0.000001  0.000013
  2000   0.000019  0.000030  0.000002  0.000025
  3000   0.000030  0.000027  0.000005  0.000036
  4000   0.000042  0.000082  0.000005  0.000054
  5000   0.000056  0.000022  0.000008  0.000060
  6000   0.000064  0.000099  0.000012  0.000074
  7000   0.000089  0.000162  0.000010  0.000086
  8000   0.000093  0.000203  0.000009  0.000100
  9000   0.000112  0.000207  0.000011  0.000111
 10000   0.000121  0.000009  0.000010  0.000130
 11000   0.000130  0.000184  0.000013  0.000138
 12000   0.000143  0.000113  0.000026  0.000156
 13000   0.000160  0.000343  0.000016  0.000166
 14000   0.000171  0.000056  0.000026  0.000179
 15000   0.000189  0.000118  0.000016  0.000187
 16000   0.000203  0.000313  0.000029  0.000205
 17000   0.000206  0.000349  0.000028  0.000212
 18000   0.000224  0.000418  0.000025  0.000243
 19000   0.000245  0.000459  0.000030  0.000246
 20000   0.000283  0.000060  0.000022  0.000260
```

```
[[NEARLY SORTED]]
     n  QuickSort   PrQueue   RSelect    Select
-------------------------------------------------
  1000   0.000010  0.000006  0.000001  0.000015
  2000   0.000022  0.000013  0.000006  0.000028
  3000   0.000034  0.000015  0.000007  0.000044
  4000   0.000047  0.000010  0.000005  0.000058
  5000   0.000063  0.000092  0.000008  0.000064
  6000   0.000080  0.000092  0.000012  0.000080
  7000   0.000092  0.000057  0.000013  0.000097
  8000   0.000105  0.000063  0.000018  0.000113
  9000   0.000130  0.000103  0.000009  0.000120
 10000   0.000131  0.000174  0.000023  0.000125
 11000   0.000147  0.000089  0.000022  0.000144
 12000   0.000160  0.000258  0.000027  0.000154
 13000   0.000178  0.000333  0.000021  0.000167
 14000   0.000196  0.000079  0.000018  0.000203
 15000   0.000230  0.000428  0.000025  0.000192
 16000   0.000230  0.000249  0.000031  0.000221
 17000   0.000250  0.000270  0.000027  0.000233
 18000   0.000267  0.000266  0.000034  0.000281
 19000   0.000285  0.000568  0.000022  0.000244
 20000   0.000299  0.000226  0.000032  0.000274
```

3 Entrega

1. **O que entregar?**

   Um arquivo compactado a ser entregue deve conter o seguinte:

   - programa(s) desenvolvido(s) (e um arquivo **`Makefile`**, se for o caso),

- tabelas dos tempos de execução dos algoritmos (em texto), de acordo com a formatação apresentada na seção 2.1, e
- gráficos (no formato `.pdf`) gerados a partir das tabelas dos tempos de execução dos algoritmos. Exemplos relativos às tabelas da seção 2.1 usando o software **`gnuplot`** são mostrados na Figura 1.

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. Um fórum de discussão deste trabalho já se encontra aberto. Após abrir uma sessão digitando seu *login* e sua senha, vá até a sessão "Atividades" e escolha "Entrega da atividade 3". Você pode entregar a atividade quantas vezes quiser até às **23 horas e 59 minutos** do dia **6 de outubro de 2022**. A última versão entregue é aquela que será corrigida. Encerrado o prazo, não serão mais aceitos trabalhos.

3. **Linguagem de programação**

   Use a linguagem de programação de sua preferência para desenvolver esta atividade.

4. **Atrasos**

   Trabalhos atrasados não serão aceitos. Não deixe para entregar seu trabalho na última hora. Para prevenir imprevistos como queda de energia, problemas com o sistema, e/ou falha de conexão com a internet, sugerimos que a entrega do trabalho seja feita pelo menos um dia antes do prazo determinado.

5. **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.

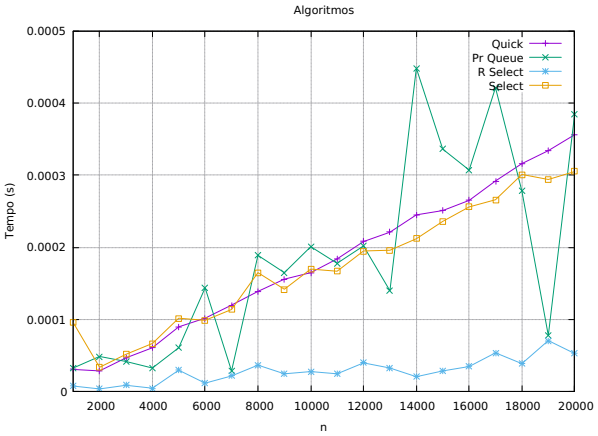
6. **Arquivo com o programa fonte**

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

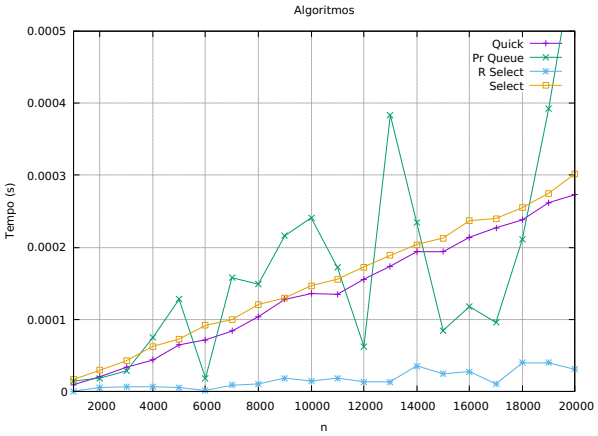
7. **Conduta Ética**

   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!

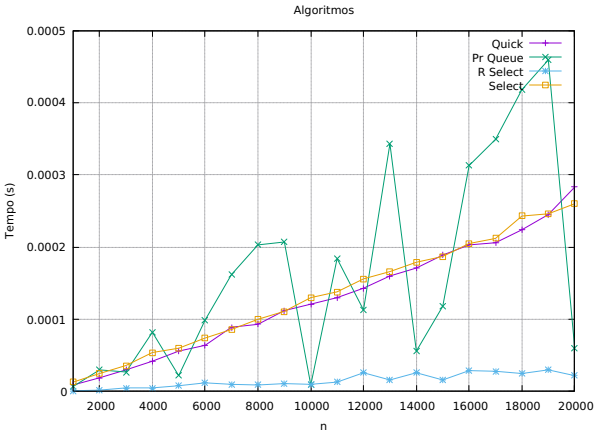
   Trabalhos considerados plagiados terão nota ZERO.



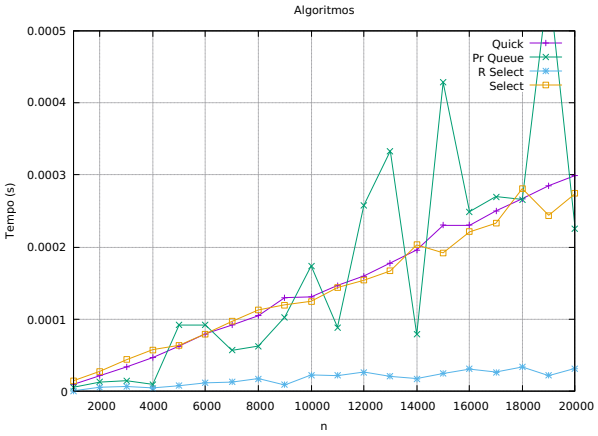
(a) Vetor aleatório.



(b) Vetor reverso.



(c) Vetor ordenado.



(d) Vetor quase ordenado.

Figura 1: Gráficos da execução dos algortimos de seleção para o caso de teste com **`inc`** $= 1000$, **`fim`** $= 20000$, **`stp`** $= 1000$ e **`rpt`** $= 10$.