UTFPR - Universidade Tecnológica Federal do Paraná

Aluno: *Jessé Pires Barbato Rocha*
RA: *2149389*
Disciplina: *Algoritmos e Estruturas de Dados 2*

5. Execute os três algoritmos sugeridos acima para buscar a k-ésima menor chave de um vetor aleatório gerado com a função int* random_vector_unique_elems(int n, int seed) com n = 1000, 10000, 100000, 500000 e seed = 42. Preencha as tabelas a seguir com o tempo de execução dos seguintes casos de testes (só precisa executar o *SelectionMinK* até p/ k = 10000):

	n = 1000	n = 10000	n = 100000	n = 500000	n = 1000000
SelectionMinK	$1x10^{-5}$	$4,6x10^{-5}$	$5,06x10^{-4}$	$2,2x10^{-3}$	$4,42x10^{-3}$
HeapMinK	$5x10^{-5}$	$5,5x10^{-4}$	$6,142x10^{-3}$	$23,287x10^{-3}$	$47,105x10^{-3}$
QuickMinK	$1,4x10^{-5}$	$4,1x10^{-4}$	$4,007x10^{-3}$	$12,407x10^{-3}$	$32,112x10^{-3}$

Figura 1: Tempos de Execução para k = 1

	n = 1000	n = 10000	n = 100000	n = 500000	n = 1000000
SelectionMinK	$1,267x10^{-3}$	$12,271x10^{-2}$	-	-	-
HeapMinK	$1,83x10^{-4}$	$2,321x10^{-3}$	$2,9548x10^{-2}$	$1,62058x10^{-1}$	$3,51923x10^{-1}$
QuickMinK	$4,6x10^{-5}$	$3,57x10^{-4}$	$5,356x10^{-3}$	$2,032x10^{-2}$	$5,0436x10^{-2}$

Figura 2: Tempos de Execução para k = $\frac{n}{3}$

	n = 1000	n = 10000	n = 100000	n = 500000	n = 1000000
SelectionMinK	$1,733x10^{-3}$	$16,6208x10^{-2}$	-	-	-
HeapMinK	$2,81x10^{-4}$	$3,229x10^{-3}$	$4,2696x10^{-2}$	$2,2981x10^{-1}$	$4,97322x10^{-1}$
QuickMinK	$4,7x10^{-5}$	$4,62x10^{-4}$	$4,064x10^{-3}$	$1,9581x10^{-2}$	$4,7346x10^{-2}$

Figura 3: Tempos de Execução para k = $\frac{n}{2}$

6. Você notou algum padrão nos resultados obtidos no item 5? Explique o que você descobriu.

R: nota-se, de acordo com o tempo que cada algoritmo leva para encontrar a *k*-ésima menor chave de um vetor que, em primeiro lugar, o *selectionMinK* é um tanto eficiente enquanto se tem um vetor pequeno. Conforme o tamanho do vetor aumenta, sua eficiência é perdida, uma vez que ele tem que varrer uma ou mais vez o vetor (dependendo de *K*), procurando o menores valores sucessivos. A exceção para tal padrão é quando *k = 1*. Neste caso, tal algoritmo ainda tem um desempenho bom por só precisar varrer encontrar a *k*-ésima menor chave uma vez. Outro padrão notado é que o tempo gasto pelo *HeapMinK* e pelo *QuickMinK* aumenta aproximadamente *10¹* segundos conforme *N = N * 10* e aumenta aproximadamente metade do tempo anterior quando *N = N * 10*. Isso ocorre pelo fato de que a complexidade desses algoritmos (para o melhor caso e caso médio no *QuickMinK* e para qualquer caso no *HeapMinK*) é $\theta(n \, lg \, n)$, o que faz com que o tempo siga esse padrão de aumento.