Projeto de Algoritmo com Implementação nº 1

Lucas Valente Viegas de Oliveira Paes - RA 220958

1 Complexidade de cada método

Método 1 Nesse método, itera-se k vezes pelo vetor, que tem n elementos. Em cada iteração, usamos variáveis auxiliares como min_{atual} e $max_{minimos}$ para determinar se o elemento atual é o k-ésimo menor. Como essas operações dentro dos laços tomam tempo constante, então esse algoritmo é $O(kn)$.

Método 2 Nesse método, primeiro ordena-se o vetor com QuickSort, cuja complexidade é $O(nlogn)$, para depois adicionar os k menores elementos ao vetor de saída, em $O(k)$. Logo, esse algoritmo é $O(nlogn)$.

Método 3 Nesse método, primeiro constrói-se um min-heap in-place a partir do vetor, levando $O(n)$. Então, remove-se k vezes o menor elemento do heap para adicionar ao vetor de saída, levando $O(klogn)$. Logo, esse algoritmo é $O(klogn)$

2 Resultados do experimento

Método 1 É melhor que os métodos 2 e 3 apenas para valores de k muito pequenos. Isso se deve ao overhead do Heapify, que envolve muitas trocas de posição no vetor, e do QuickSort, que envolve trocas e particionamento.

Método 2 Desempenhou melhor para valores de k grandes, uma vez que, como $k \to n$, o método 3 tende a se comportar como um HeapSort, que, no geral, tem desempenho pior que o do QuickSort.

Método 3 Esse método obteve melhor desempenho para valores medianos de k, uma vez que, nessa faixa, $k << n$ e, assim, $O(klogn) < O(nlogn)$.

Finalmente, os valores de transição de k encontrados foram: $k1 = \frac{15+15+15}{3} = 15$ e $k2 = \frac{437508+442390+443234}{3} = 441044$