1) Um Heap é uma estrutura de dados em forma de árvore binária que é frequentemente usada para implementar uma fila de prioridade. A propriedade principal de um Heap é que o elemento na raiz sempre tem a maior ou menor prioridade em relação aos outros elementos do Heap, dependendo se o Heap é um Max-Heap ou um Min-Heap.

A operação de retirada, também conhecida como "pop" ou "extract", é usada para remover o elemento raiz do Heap. Para fazer isso, primeiro precisamos salvar o valor do elemento na raiz do Heap. Em seguida, movemos o elemento mais à direita na última linha do Heap para a posição da raiz. Depois disso, o Heap pode estar fora de ordem, então precisamos reordenar os elementos para restaurar a propriedade do Heap. Para fazer isso, comparamos o valor do elemento na posição da raiz com o valor dos seus filhos e trocamos o valor com o menor (ou maior, dependendo se o Heap é um Min-Heap ou um Max-Heap) dos filhos, se necessário. Em seguida, repetimos esse processo para o novo elemento na posição da raiz e seus filhos, até que a propriedade do Heap seja restaurada. Por fim, retornamos o valor que salvamos na etapa inicial.

A operação de inserção, também conhecida como "push" ou "insert", é usada para adicionar um novo elemento ao Heap. Para fazer isso, adicionamos o novo elemento ao final do Heap, à direita do último elemento na última linha. Em seguida, o Heap pode estar fora de ordem, então precisamos reordenar os elementos para restaurar a propriedade do Heap. Para fazer isso, comparamos o valor do novo elemento com o valor do seu pai e trocamos os dois se necessário. Em seguida, repetimos esse processo com o novo elemento e seu pai, até que a propriedade do Heap seja restaurada.

Esses processos são repetidos até que a propriedade do Heap seja restaurada. A complexidade da operação de retirada e inserção é O(log n), onde n é o número de elementos no Heap. Isso torna o Heap uma estrutura de dados eficiente para implementar filas de prioridade em algoritmos que exigem um desempenho rápido.

2) O HeapSort não é recomendado para arquivos com poucos registros devido ao alto custo de memória e complexidade constante necessários para construir a estrutura de Heap a partir dos dados, o que pode ser excessivo e desnecessário para arquivos pequenos.

3)

8 5 6 4 2 1 0 (vetor inicial)
6 5 1 4 2 0 8 (após a construção do heap)
0 5 1 4 2 6 (após a troca da raiz com o último elemento não classificado)
1 4 2 5 6 (após a reorganização do heap)
2 4 5 6 (após a troca da raiz com o último elemento não classificado)
4 5 6 (após a reorganização do heap)
5 6 (após a troca da raiz com o último elemento não classificado)
6 (após a reorganização do heap)

Vetor ordenado: 0 1 2 4 5 6 8

4) 2 5 3 0 2 3 0 3 (vetor inicial)
0 0 2 2 3 3 3 5 (após a contagem das ocorrências)
0 0 2 4 7 10 13 14 (após a contagem acumulada das ocorrências)
2 5 3 0 2 3 0 3 (vetor inicial)
0 0 2 4 7 10 13 14 (vetor de contagem acumulada)
2 5 3 0 2 3 0 3 (vetor inicial)
0 0 1 1 3 6 9 13 (posições finais dos elementos no vetor ordenado)
0 0 2 2 3 3 3 5 (vetor ordenado)

Vetor ordenado: 0 0 2 2 3 3 3 5