Heap Sort

Heap sort is a comparison based sorting technique based on Binary Heap data structure. It is similar to selection sort where we first find the maximum element and place the maximum element at the end. We repeat the same process for remaining elements.

What is Binary Heap?

Let us first define a Complete Binary Tree. A complete binary tree is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible (Source Wikipedia).

A Binary Heap is a Complete Binary Tree where items are stored in a special order such that value in a parent node is greater(or smaller) than the values in its two children nodes. The former is called as max heap and the latter is called min heap. The heap can be represented by binary tree or array.

Array based representation for Binary Heap: Since a Binary Heap is a Complete Binary Tree, it can be easily represented as array and array based representation is space efficient. If the parent node is stored at index I, the left child can be calculated by 2 * I + 1 and right child by 2 * I + 2 (assuming the indexing starts at 0).

Heap Sort Algorithm for sorting an array in increasing order:

1. Build a max heap from the input data.
2. At this point, the largest item is stored at the root of the heap. Replace it with the last item of the heap followed by reducing the size of heap by 1. Finally, heapify the root of tree.
3. Repeat above steps while size of heap is greater than 1.

How to build the heap?

Heapify procedure can be applied to a node only if its children nodes are heapified. So the heapification must be performed in the bottom up order.

How to build the heap?

Heapify procedure can be applied to a node only if its children nodes are heapified. So the heapification must be performed in the bottom up order.

Lets understand with the help of an example:

```
Input data: [4, 10, 3, 5, 1]
         4(0)
        /   \
     10(1)   3(2)
    /   \
 5(3)    1(4)

The numbers in bracket represent the indices in the array
representation of data.

Applying heapify procedure to index 1:
         4(0)
        /   \
    10(1)    3(2)
    /   \
5(3)    1(4)

Applying heapify procedure to index 0:
        10(0)
        /  \
     5(1)  3(2)
    /   \
 4(3)    1(4)

The heapify procedure calls itself recursively to build heap
in top down manner.
```

Implementation:

```
1
2  // To heapify a subtree rooted with node i which is  an index in arr[]. n is size of heap
3  void heapify(int arr[], int n, int i)
4  {
5      int largest = i; // Initialize largest as root
6      int l = 2*i + 1; // left = 2*i + 1
7      int r = 2*i + 2; // right = 2*i + 2
8      if (l < n && arr[l] > arr[largest])    // If left child is larger than root
9          largest = l;
10     if (r < n && arr[r] > arr[largest]) // If right child is larger than largest so far
11         largest = r;
12     // If largest is not root
13     if (largest != i)
14     {
15         swap(arr[i], arr[largest]);
16         heapify(arr, n, largest);// Recursively heapify the affected sub-tree
17     }
18 }
19 // Main function for heap sort
20 void heapSort(int arr[], int n)
21 {
22     for (int i = n / 2 - 1; i >= 0; i--)  // Build heap (rearrange array)
23         heapify(arr, n, i);
24     for (int i=n-1; i>=0; i--)// One by one extract an element from heap
25     {
26         swap(arr[0], arr[i]);    // Move current root to end
27         heapify(arr, i, 0);// call max heapify on the reduced heap
28     }
29 }
```

Important Notes:

- Heap sort is an in-place algorithm.
- Its typical implementation is not stable, but can be made stable (See this).

Time Complexity: Time complexity of heapify is O(N*LogN). Time complexity of createAndBuildHeap() is O(N) and overall time complexity of Heap Sort is **O(N*LogN)** where N is the number of elements in the list or array.

Heap sort algorithm has limited use because Quicksort and Mergesort are better in practice. Nevertheless, the Heap data structure itself is enormously used.