geeksforgeeks.org

Heap Sort - GeeksforGeeks

6-7 minutes

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 element.

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.

Why 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 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.

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.
```

- C++
- Java
- Python

C++

```
#include <iostream>
using namespace std;
void heapify(int arr[], int n, int i)
{
    int largest = i;
    int l = 2*i + 1;
    int r = 2*i + 2;
    if (l < n && arr[l] > arr[largest])
        largest = l;
    if (r < n && arr[r] > arr[largest])
        largest = r;
    if (largest != i)
    {
        swap(arr[i], arr[largest]);
        heapify(arr, n, largest);
    }
}
void heapSort(int arr[], int n)
```

```
    {
        for (int i = n / 2 - 1; i >= 0; i--)
            heapify(arr, n, i);
        for (int i=n-1; i>=0; i--)
        {
            swap(arr[0], arr[i]);
            heapify(arr, i, 0);
        }
    }
    void printArray(int arr[], int n)
    {
        for (int i=0; i<n; ++i)
            cout << arr[i] << " ";
        cout << "\n";
    }
    int main()
    {
        int arr[] = {12, 11, 13, 5, 6, 7};
        int n = sizeof(arr)/sizeof(arr[0]);
        heapSort(arr, n);
        cout << "Sorted array is \n";
        printArray(arr, n);
    }
```

Java

```
public class HeapSort
{
    public void sort(int arr[])
    {
        int n = arr.length;
        for (int i = n / 2 - 1; i >= 0; i--)
            heapify(arr, n, i);
        for (int i=n-1; i>=0; i--)
        {
            int temp = arr[0];
            arr[0] = arr[i];
            arr[i] = temp;
            heapify(arr, i, 0);
        }
    }
    void heapify(int arr[], int n, int i)
    {
        int largest = i;
        int l = 2*i + 1;
        int r = 2*i + 2;
        if (l < n && arr[l] > arr[largest])
            largest = l;
        if (r < n && arr[r] > arr[largest])
            largest = r;
        if (largest != i)
        {
            int swap = arr[i];
            arr[i] = arr[largest];
            arr[largest] = swap;
            heapify(arr, n, largest);
        }
    }
    static void printArray(int arr[])
    {
        int n = arr.length;
        for (int i=0; i<n; ++i)
            System.out.print(arr[i]+" ");
        System.out.println();
    }
    public static void main(String args[])
    {
        int arr[] = {12, 11, 13, 5, 6, 7};
        int n = arr.length;
        HeapSort ob = new HeapSort();
        ob.sort(arr);
        System.out.println("Sorted array is");
        printArray(arr);
    }
}
```

Python

```
def heapify(arr, n, i):
    largest = i
    l = 2 * i + 1
    r = 2 * i + 2
    if l < n and arr[i] < arr[l]:
        largest = l
    if r < n and arr[largest] < arr[r]:
        largest = r
    if largest != i:
        arr[i],arr[largest] = arr[largest],arr[i]
        heapify(arr, n, largest)
def heapSort(arr):
    n = len(arr)
    for i in range(n, -1, -1):
        heapify(arr, n, i)
    for i in range(n-1, 0, -1):
        arr[i], arr[0] = arr[0], arr[i]
        heapify(arr, i, 0)
arr = [ 12, 11, 13, 5, 6, 7]
```

```
heapSort(arr)
n = len(arr)
print ("Sorted array is")
for i in range(n):
    print ("%d" %arr[i]),
```

Output:

```
Sorted array is
5 6 7 11 12 13
```

Here is previous C code for reference.

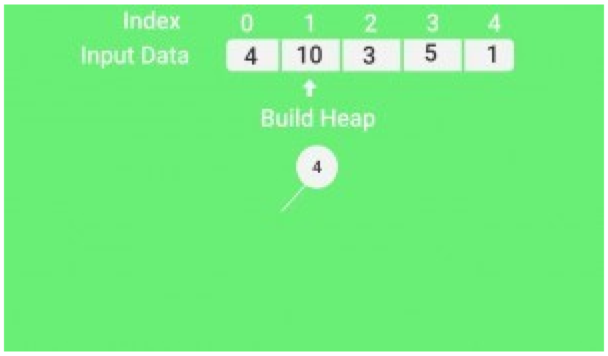
Notes:
Heap sort is an in-place algorithm.
Its typical implementation is not stable, but can be made stable (See this)

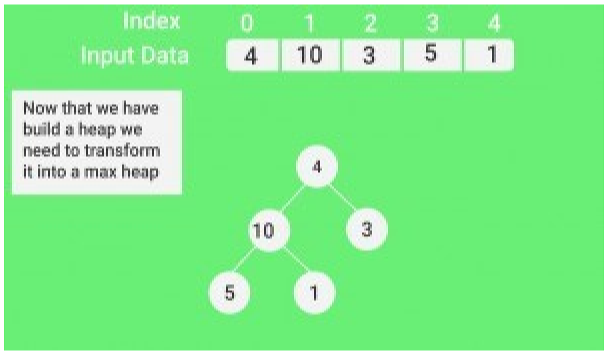
Time Complexity: Time complexity of heapify is O(Logn). Time complexity of createAndBuildHeap() is O(n) and overall time complexity of Heap Sort is O(nLogn).

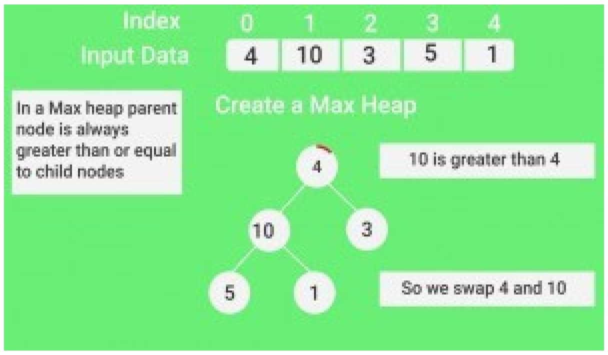
Applications of HeapSort
1. Sort a nearly sorted (or K sorted) array
2. k largest(or smallest) elements in an array

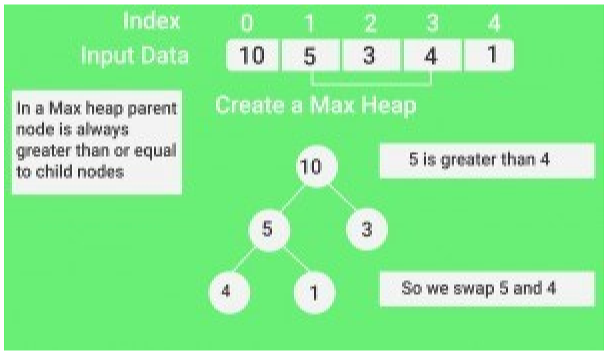
Heap sort algorithm has limited uses because Quicksort and Mergesort are better in practice. Nevertheless, the Heap data structure itself is enormously used. See Applications of Heap Data Structure

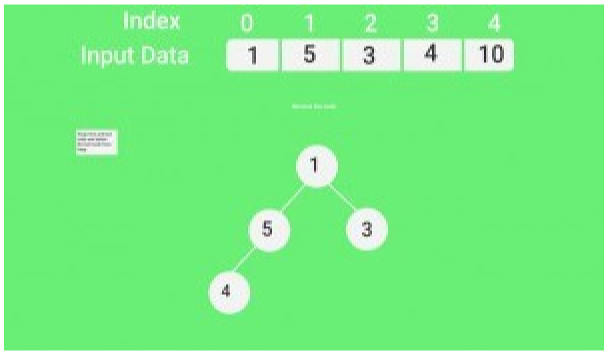
Snapshots:

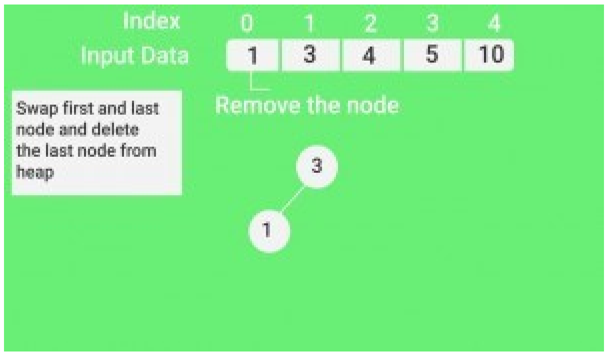

Index 0 1 2 3 4
Input Data 4 10 3 5 1
Build Heap
4


Index 0 1 2 3 4
Input Data 4 10 3 5 1
Now that we have build a heap we need to transform it into a max heap
4
10
3
5
1


Index 0 1 2 3 4
Input Data 4 10 3 5 1
In a Max heap parent node is always greater than or equal to child nodes
Create a Max Heap
10 is greater than 4
4
10
3
5
1
So we swap 4 and 10


Index 0 1 2 3 4
Input Data 10 5 3 4 1
In a Max heap parent node is always greater than or equal to child nodes
Create a Max Heap
5 is greater than 4
10
5
3
4
1
So we swap 5 and 4


Index 0 1 2 3 4
Input Data 1 5 3 4 10
1
5
3
4



Quiz on Heap Sort

Other Sorting Algorithms on GeeksforGeeks/GeeksQuiz:
QuickSort, Selection Sort, Bubble Sort, Insertion Sort, Merge Sort, Heap Sort, QuickSort, Radix Sort, Counting Sort, Bucket Sort, ShellSort, Comb Sort, Pigeonhole Sort

Coding practice for sorting.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.