Yusuf Fawzy Elnady

Heap Data Structure

Summary:

In computer science, a **heap** is a specialized tree-based data structure which is essentially an almost complete tree that satisfies the **heap property**: in a max heap, for any given node C, if P is a parent node of C, then the key (the value) of P is greater than or equal to the key of C. In a min heap, the key of P is less than or equal to the key of C. The node at the "top" of the heap (with no parents) is called the root node.

The project is built using **C++, Code::Blocks**.

Heap Sort:

Heap sort is a comparison-based sorting technique based on **Binary Heap data structure**. To sort in an increasing order, The algorithm follow this approach:
- **Build a min heap from the input data**.
- **The smallest** item will be stored at the root of the heap, Then swap this element with the last element and decrease the heap size, repeating this step again and again till all of the heap is sorted in a descending order.
- **Printing the array elements** from the last element towards the first, will print them sorted in an increasing way.

Building the Heap:

The first task in heap sort is to convert the input array into a heap array, in other words, to heapify the array. The heapify procedure can be done in top-down, or bottom-up manner.
The implemented one is bottom-up so we just have a loop of n/2 using the Floyd's Algorithm

Insert Key:

Getting the new value as a parameter of the function, put initially the value at the end of the heap, then start to move the key up till it finds a place that suits it, the condition to **stop swapping** is reaching the root of the heap or the parent of current node becomes smaller than the key.

minHeapify:

This is a top-down approach starting from a given index (i) we start to move the element down and replace the element at index (i) with the smallest element from its children, we continue in this approach till we found that no more swaps are needed to be done. This is an iterative approach, for the recursive approach, follow these three steps:

```
heapify(node)
  heapify(leftchild)
  heapify(rightchild)
  moveDown(root)
```

Yusuf Fawzy Elnady

Pop:

1- remove the root of the heap by swapping the last element with the root.
2- do a minHeapify for the element at index 0.
3- decrease the heapSize variable.
4- return the element that has been deleted, if the heap was empty it returns INT_MIN.

PrintSorted – PrintOriginal:

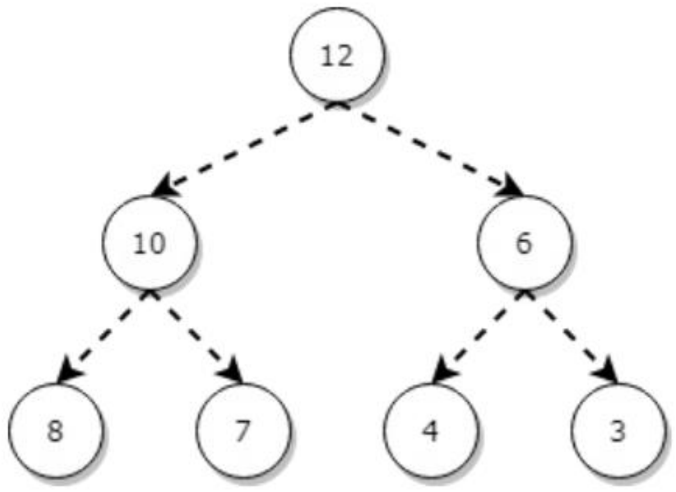
Print Sorted function prints the elements by pop upping the elements one after the other, but the Print Original just prints the heap array as it is stored internally as a tree.

Example:- Tracing a max heap insertion and deletion operations:

Start with an empty max-heap and insert the following values in order: 12, 10, 6, 8, 7, 4, 3. Call the resulting heap H.
Draw the array that corresponds to the heap H.
Draw the array that corresponds to the heap H after deleting 12.
Draw the array that corresponds to the heap H after inserting 11.

Value to insert	Heap state after
12	[12]
10	[12, 10]
6	[12, 10, 6]
8	[12, 10, 6, 8]
7	[12, 10, 6, 8, 7]
4	[12, 10, 6, 8, 7, 4]
3	[12, 10, 6, 8, 7, 4, 3]

Final heap structure: [12, 10, 6, 8, 7, 4, 3]



Yusuf Fawzy Elnady

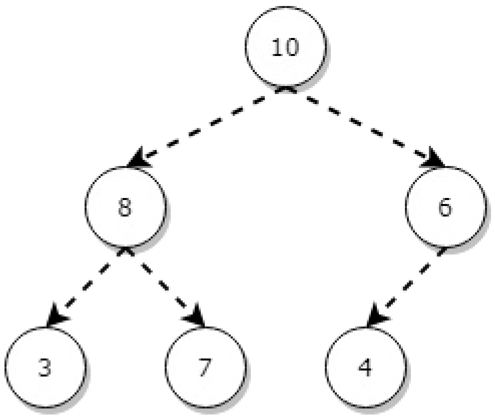
Deleting 12

We replace the root with the last node: [3, 10, 6, 8, 7, 4]

The new root is less than both its children. shift the new root '3' with the max if its children: [10, 3, 6, 8,7, 4]

Shift the '3' again with the max if its children: [10, 8, 6, 3, 7, 4]

Final heap structure: [10, 8, 6, 3, 7, 4]



Inserting 11

We insert the new value at the end of the array: [10, 8, 6, 3, 7, 4, 11]

The new node is greater than its parent, shift '11' up: [10, 8, 11, 3, 7, 4, 6]

Shift the '11' again with its parent: [11, 8, 10, 3, 7, 4, 6]

Final heap structure: [11, 8, 10, 3, 7, 4, 6]