Tutorial 12 (14-19 NOV 2022)

Q.1 Construct a Max Binary Heap using following elements: 10, 22, 32, 42, 50, 45, 35, 25, 15, 5, 8, 18, 28, 38, and 48 and sort them in ascending order using heap sort.

Q2. Construct a Min Binary Heap using following elements: 10, 22, 30, 40, 50, 45, 35, 25, 15, 5, 8, 18, 28, 38, and 48

Q3. Analyse the performance of the Binary Heap, if it is created using (a) array and (b) binary tree (complete)

The tree uses more time and memory. The complexities are the same, but the constant factors are different.

The pointers of the tree use a lot of memory, compared to the array-based heap, where you barely need any additional space but the one taken by the values themselves. And manipulating these pointers takes time too. Allocating and de-allocating nodes might take some time and space also.

Q4. You have been given a Max Binary Heap and it is desired to convert it into a Min Binary Heap. Write a program to perform the desired task.

#include <bits/stdc++.h>

**using** **namespace** std;

// to heapify a subtree with root at given index

**void** MaxHeapify(**int** arr[], **int** i, **int** N)

{

**int** l = 2 \* i + 1;

**int** r = 2 \* i + 2;

**int** largest = i;

**if** (l < N && arr[l] > arr[i])

        largest = l;

**if** (r < N && arr[r] > arr[largest])

        largest = r;

**if** (largest != i) {

        swap(arr[i], arr[largest]);

        MaxHeapify(arr, largest, N);

    }

}

// This function basically builds max heap

**void** convertMaxHeap(**int** arr[], **int** N)

{

    // Start from bottommost and rightmost

    // internal mode and heapify all internal

    // modes in bottom up way

**for** (**int** i = (N - 2) / 2; i >= 0; --i)

        MaxHeapify(arr, i, N);

}

// A utility function to print a given array

// of given size

**void** printArray(**int**\* arr, **int** size)

{

**for** (**int** i = 0; i < size; ++i)

        cout << arr[i] << " ";

}

// Driver's code

**int** main()

{

    // array representing Min Heap

**int** arr[] = { 3, 5, 9, 6, 8, 20, 10, 12, 18, 9 };

**int** N = **sizeof**(arr) / **sizeof**(arr[0]);

**printf**("Min Heap array : ");

    printArray(arr, N);

    // Function call

    convertMaxHeap(arr, N);

**printf**("\nMax Heap array : ");

    printArray(arr, N);

**return** 0;

}

Q5. You have been given two Max Binary Heaps and it is desired to merge both heaps into a single heap. Write a program to perform the desired task.

|  |
| --- |
| #include <iostream>  **using** **namespace** std;   .  **void** maxHeapify(**int** arr[], **int** N, **int** idx)  {    **int** l = 2 \* idx + 1;  **int** r = 2 \* idx + 2;  **int** max = idx;  **if** (l < N && arr[l] > arr[idx])          max = l;  **if** (r < N && arr[r] > arr[max])          max = r;        // Put maximum value at root and      // recur for the child with the      // maximum value  **if** (max != idx) {          swap(arr[max], arr[idx]);          maxHeapify(arr, N, max);      }  }    // Builds a max heap of given arr[0..n-1]  **void** buildMaxHeap(**int** arr[], **int** N)  {      // building the heap from first non-leaf      // node by calling max heapify function  **for** (**int** i = N / 2 - 1; i >= 0; i--)          maxHeapify(arr, N, i);  }    // Merges max heaps a[] and b[] into merged[]  **void** mergeHeaps(**int** merged[], **int** a[], **int** b[], **int** N,  **int** M)  {      // Copy elements of a[] and b[] one by one      // to merged[]  **for** (**int** i = 0; i < N; i++)          merged[i] = a[i];  **for** (**int** i = 0; i < M; i++)          merged[N + i] = b[i];        // build heap for the modified array of      // size n+m      buildMaxHeap(merged, N + M);  }    // Driver's code  **int** main()  {  **int** a[] = { 10, 5, 6, 2 };  **int** b[] = { 12, 7, 9 };    **int** N = **sizeof**(a) / **sizeof**(a[0]);  **int** M = **sizeof**(b) / **sizeof**(b[0]);    **int** merged[N + M];        // Function call      mergeHeaps(merged, a, b, N, M);    **for** (**int** i = 0; i < N + M; i++)          cout << merged[i] << " ";    **return** 0;  } |

**Output**

12 10 9 2 5 7 6

**Time Complexity:**O(N + M)  
**Auxiliary Space:**O(N + M)