CS2023 - Data Structures and Algorithms
In-class Lab Exercise

Week 8

Name: Wijetunga W.L.N.K Index Number: 200733D

You are required to answer the below questions and submit a PDF to the submission link provided under this week lab section before end of the session time (no extensions will be provided). You can either write / type your answers, but either way your answers should be readable.

GitHub Repository - https://github.com/namiwijeuom/CS2023-Data-Structures-and-Algorithms-In-class-Lab-Exercises/tree/main/Lab%208

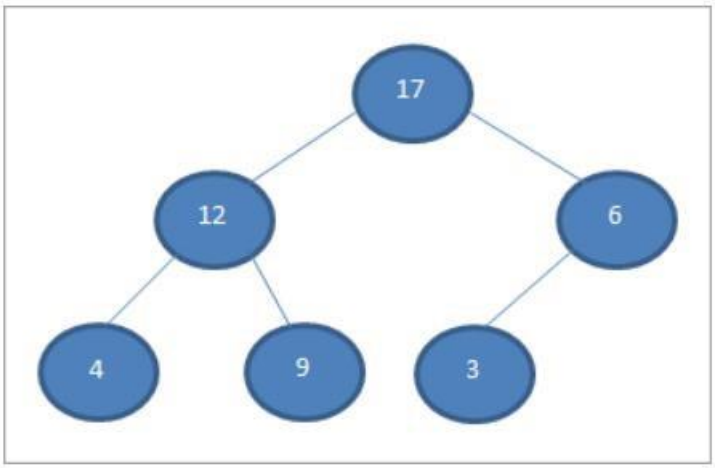
Exercise:

A binary heap is represented using a complete binary tree. A complete binary tree is a binary tree in which all the nodes at each level are completely filled except for the leaf nodes and the nodes are as far as left.

While representing a heap as an array, assuming the index starts at 0, the root element is stored at 0. In general, if a parent node is at the position i, then the left child node is at the position (2*i + 1) and the right node is at (2*i +2).

Sample Input array
4 17 3 12 9 6

Sample max-heap for above input



Implement Heapsort by creating max-heap in C++. Use the given "*heap.cpp*" file for your implementation.

- In the given program, the function heapify() is used to convert the elements into a heap using recursion.
- The function heapSort() sorts the array elements using heap sort. It starts from the nonleaf nodes and calls the heapify() on each of them. This converts the array into a binary max heap.
- The main() function only shows an example of a given set of numbers. Modify it to read numbers from the user or assign random values.

Sample Output:

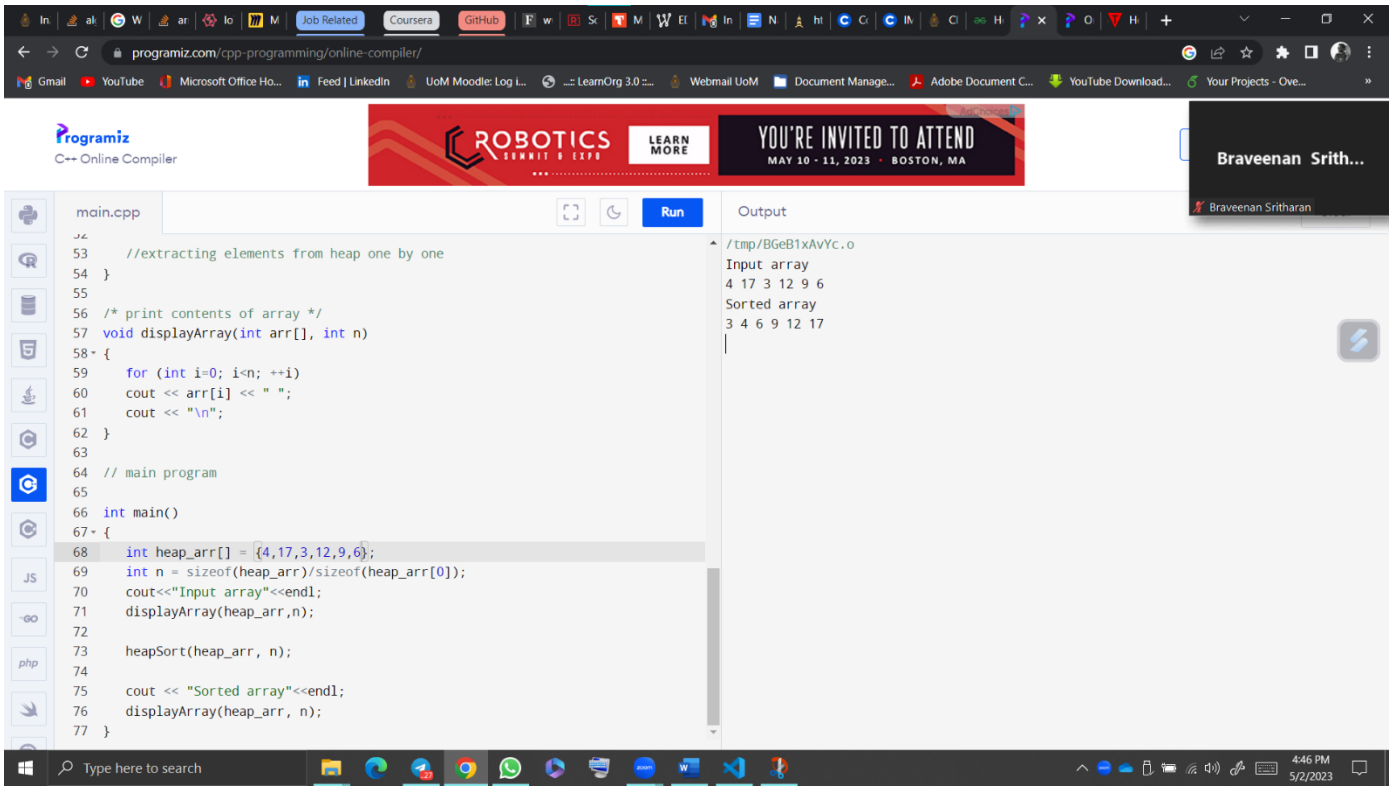
```
Input array: 4 17 3 12 9 6
Sorted array:
3 4 6 9 12 17
```

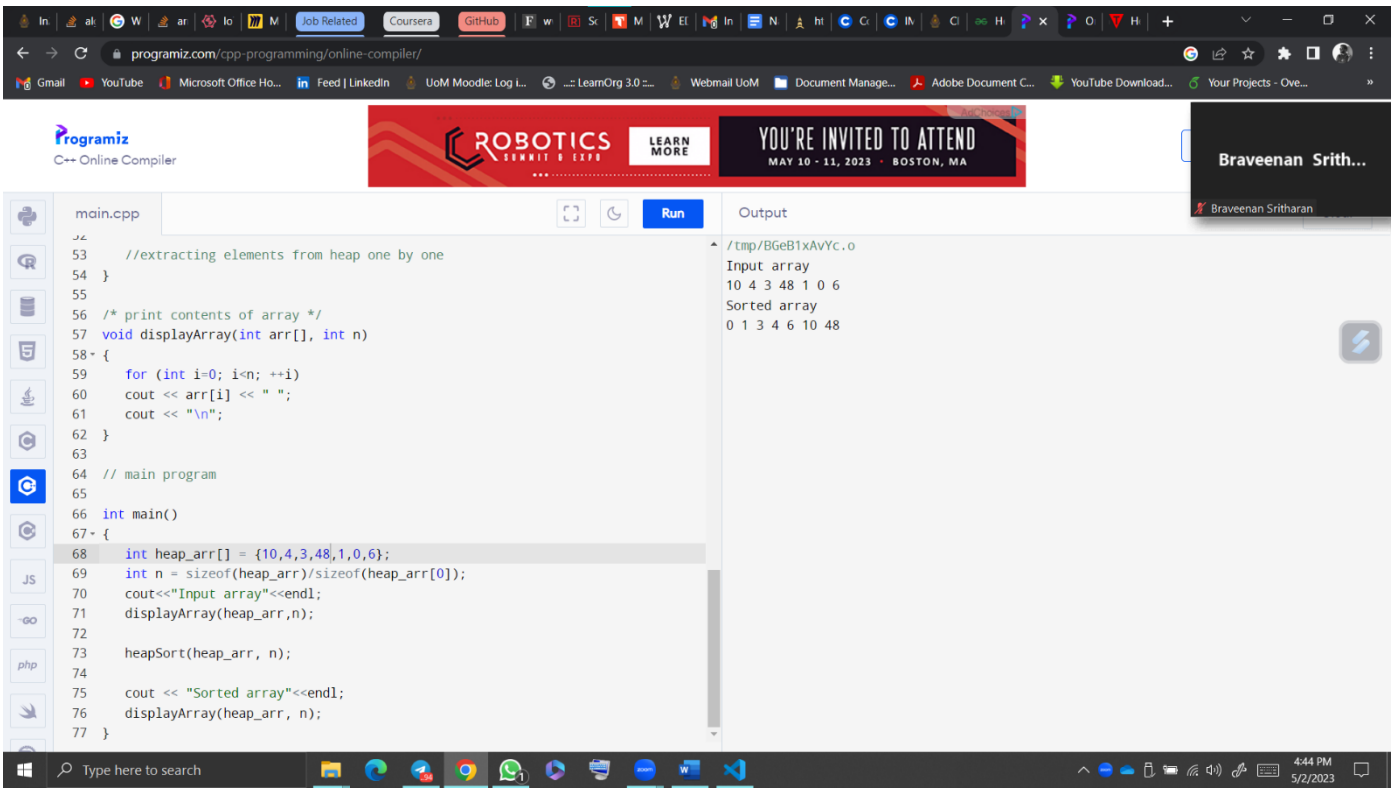
Submission:

- Include screen shots of terminal output, sample max-heap drawn for your input, and GitHub link for your implementation.
- Discuss on the time complexity of heap sort.
- Upload the answers as a PDF file named, "**In<no>_IndexNO_lab8.pdf**"

Answers:

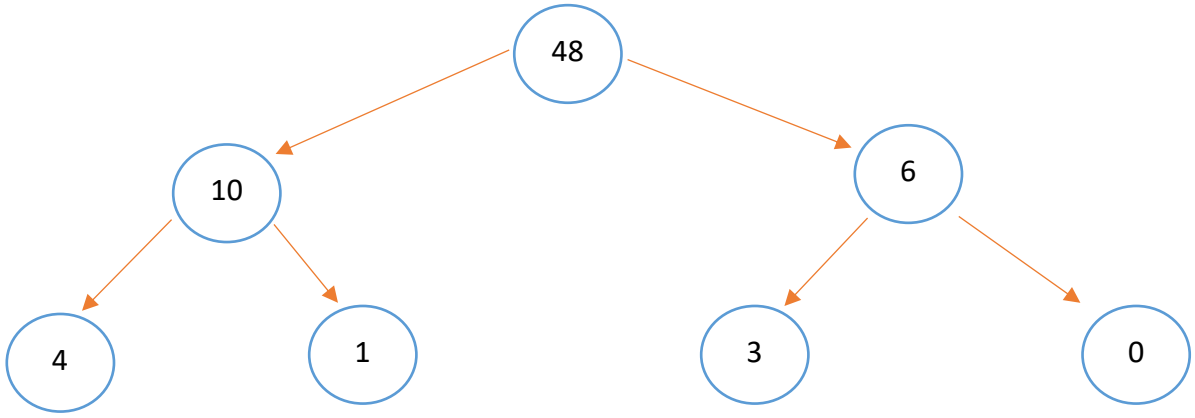
Q1)


programiz.com/cpp-programming/online-compiler/
Programiz
C++ Online Compiler
ROBOTICS
LEARN MORE
YOU'RE INVITED TO ATTEND
MAY 10 - 11, 2023 · BOSTON, MA
Braveenan Srith...
Braveenan Sritharan
main.cpp
Run
Output
//extracting elements from heap one by one
}
/* print contents of array */
void displayArray(int arr[], int n)
{
for (int i=0; i<n; ++i)
cout << arr[i] << " ";
cout << "\n";
}
// main program
int main()
{
int heap_arr[] = {4,17,3,12,9,6};
int n = sizeof(heap_arr)/sizeof(heap_arr[0]);
cout<<"Input array"<<endl;
displayArray(heap_arr,n);
heapSort(heap_arr, n);
cout << "Sorted array"<<endl;
displayArray(heap_arr, n);
}
/tmp/BGeB1xAvYc.o
Input array
4 17 3 12 9 6
Sorted array
3 4 6 9 12 17
Type here to search
4:46 PM
5/2/2023


programiz.com/cpp-programming/online-compiler/
Programiz
C++ Online Compiler
ROBOTICS
LEARN MORE
YOU'RE INVITED TO ATTEND
MAY 10 - 11, 2023 · BOSTON, MA
Braveenan Srith...
Braveenan Sritharan
main.cpp
Run
Output
//extracting elements from heap one by one
}
/* print contents of array */
void displayArray(int arr[], int n)
{
for (int i=0; i<n; ++i)
cout << arr[i] << " ";
cout << "\n";
}
// main program
int main()
{
int heap_arr[] = {10,4,3,48,1,0,6};
int n = sizeof(heap_arr)/sizeof(heap_arr[0]);
cout<<"Input array"<<endl;
displayArray(heap_arr,n);
heapSort(heap_arr, n);
cout << "Sorted array"<<endl;
displayArray(heap_arr, n);
}
/tmp/BGeB1xAvYc.o
Input array
10 4 3 48 1 0 6
Sorted array
0 1 3 4 6 10 48
Type here to search
4:44 PM
5/2/2023

Q2)

{10, 4, 3, 48, 1 , 0 , 6}



Q3)

- Best Case Complexity - Occurs when the array is already sorted. The best-case time complexity is **O(n log n).**
- Average Case Complexity - Occurs when the array elements are in jumbled order that is not properly ascending and not properly descending. The average case time complexity is **O(n log n).**
- Worst Case Complexity - Occurs when the array elements are required to be sorted in reverse order, meaning that if the array elements to be sorted in ascending order, but its elements are in descending order. The worst-case time complexity is **O(n log n).**