# 

# Department of Computer Science

**CS 250: Data Structures and Algorithms**

**Class: BSCS-9AB**

**Lab 11: Binary Heap and Heap Sort**

**Date: January 5, 2021**

**Time: 10:00 am -1:00pm, 2:00pm – 5:00pm**

# Instructor: Dr. Yasir Faheem

# 

**Lab 11: Binary Heaps and Heap Sort**

**Introduction**

This lab is based on the Binary Heap (max-heap).

**Objectives**

The objective of this lab is to implement binary heap (max heap), and also the heap sort algorithm which is an application of binary heaps.

**Tools/Software Requirement**

Visual Studio C++

**Description**

A binary heap is a complete binary tree which satisfies the heap ordering property. The ordering can be one of two types:

 the *min-heap property*: the value of each node is greater than or equal to the value of its parent, with the minimum-value element at the root.

 the *max-heap property*: the value of each node is less than or equal to the value of its parent, with the maximum-value element at the root.

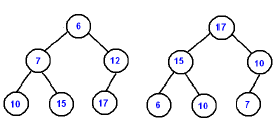


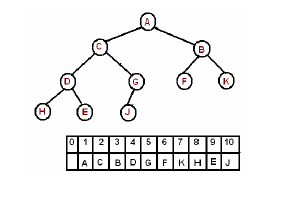
Figure : left) min-heap right) max-heap

In a heap the highest (or lowest) priority element is always stored at the root. A heap is not a sorted structure and can be regarded as partially ordered.

A heap is useful data structure when you need to remove the object with the highest (or lowest) priority. A common use of a heap is to implement a priority queue.

**Array Implementation**

A complete binary tree can be uniquely represented by storing its level order traversal in an array.



The root is the second item in the array. We skip the index zero cell of the array for the convenience of implementation. Consider k-th element of the array,

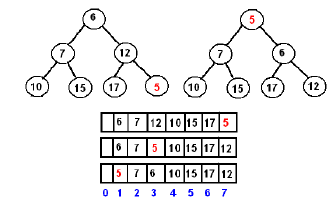
its left child is located at 2\*k index

its right child is located at 2\*k+1. index

its parent is located at k/2 index

**Insert**

The new element is initially appended to the end of the heap (left-most location available on the lowest level of the tree i.e. at index heap\_size+1.). The heap property is repaired by comparing the added element with its parent and moving the added element up a level (swapping positions with the parent).



**Delete Max**

The maximum element can be found at the root, which is the first element of the array. We remove the root and replace it with the last element of the heap and then restore the heap property

**Lab Tasks:**

**PART A:**

**CODE FOR ALL CLASSES**

Implement a binary **max-heap** using an array & implement the following functions.

#include<iostream>

using namespace std;

const int asize =5;

class Heap {

public:

int A[asize];

int heap\_size = 0;

int temp = 0;

Heap()

{

for (int i = 1; i <= asize; i++)

{

A[i] =0;

}

heap\_size = 0;

}

void Insert(int A[], int value)

{

heap\_size++;

A[heap\_size] = value;

if (isFull())

{

cout << "heap is full" << endl;

}

maxHeapifylowtoup(A, heap\_size);

}

bool isEmpty()

{

if (heap\_size == 0)

{

return true;

}

else { return false; }

}

bool isFull()

{

if (heap\_size == asize)

{

return true;

}

else { return false; }

}

int delMax(int A[])

{

int store = A[1];

for (int i = 1; i <heap\_size; i++)

{

A[i] = A[i+1];

}

maxHeapify(A, 1, heap\_size);

return store;

}

int size()

{

return heap\_size;

}

void maxHeapifylowtoup(int A[], int i)

{

int largest;

if (((i/2) >= 1) && (A[i] > A[i/2]))

{

largest = i/2;

int temp = A[i];

A[i] = A[largest];

A[largest] = temp;

maxHeapifylowtoup(A, largest);

}

}

void maxHeapify(int A[],int i, int n)

{ int largest;

if (i>=n)

{

return;

}

else

{

if (((2\*i) <= n) && (A[i]<A[2\*i]))

{

largest = 2\*i;

if ((((2\*i) + 1) <= n) && A[largest] < A[(2\*i + 1)])

{

largest = (2\*i+1);

}

int temp = A[i];

A[i] = A[largest];

A[largest] = temp;

maxHeapify(A, largest,n);

}

}

}

int Max(int A[])

{

return A[1];

}

void buildMaxHeap(int A[],int n)

{ for (int i =n/2;i>= 1;i--)

{

maxHeapify(A, i,n);

}

}

void displayHeap(int A[])

{

for (int i = 1; i <asize; i++)

{cout << A[i];}

}

void HeapSort(int A[])

{

if (heap\_size == 0)

{

return;

}

else

{

temp = A[1];

A[1] = A[heap\_size];

A[heap\_size] = temp;

heap\_size--;

cout << endl;

cout <<"heap\_size: " <<heap\_size<<endl;

maxHeapify(A, 1,heap\_size);

displayHeap(A);

cout << endl;

HeapSort(A);

}

}

};

int main()

{

Heap\* h=new Heap();

int Array[5];

h->Insert(Array,4);

h->Insert(Array, 2);

h->Insert(Array, 3);

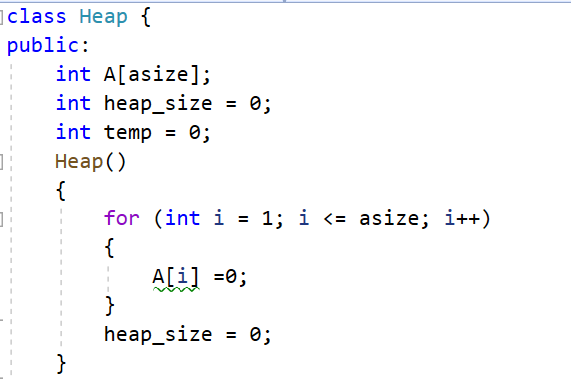
h->Insert(Array, 1);

h->HeapSort(Array);

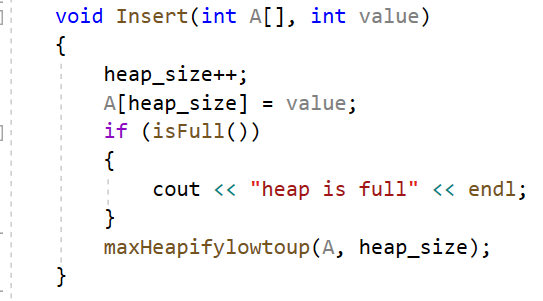
h->displayHeap(Array);

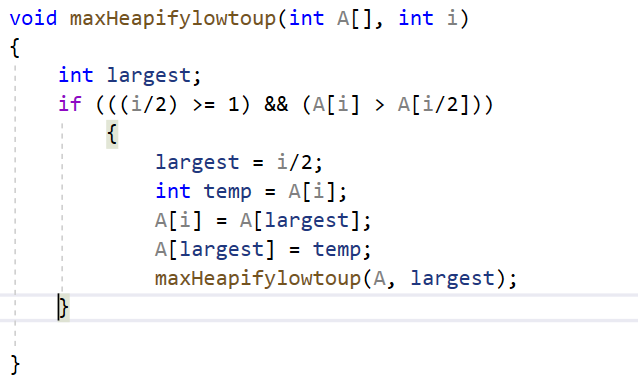
}

* **BinaryHeap()** creates a new, empty, binary heap.
  + Note that the size of a heap and the array in which its contents are stored are two different things.

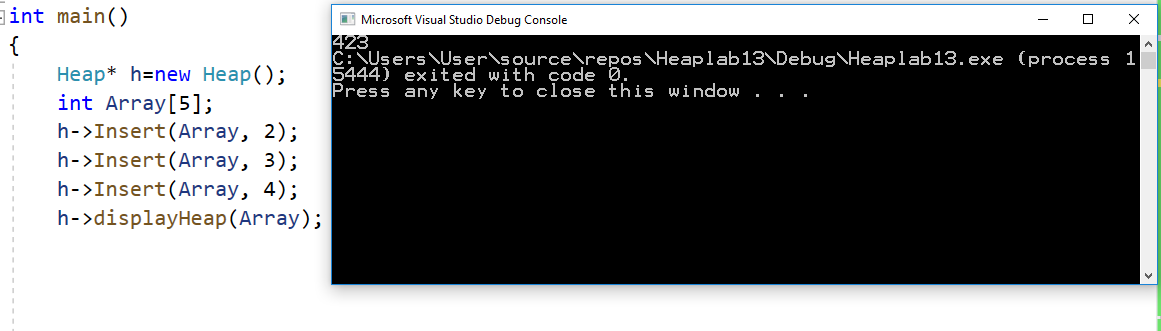


* **Insert(value)**   
  Adds a new item to the heap. A new item is always inserted at the lowest level at the leftmost available slot; in the array-based heap, the new item is inserted at the index heap\_size+1. Then, the MaxHeapify() function is called to restore the value property of max-heap.





**Output**



* **MaxHeapify(Array, i, n)**

Implement a function that shifts the value stored at index i to its logical position in a max-heap. You may use bottom-up or top-down approach to shift values. Moreover, you may use recursion or iteration to implement this function.

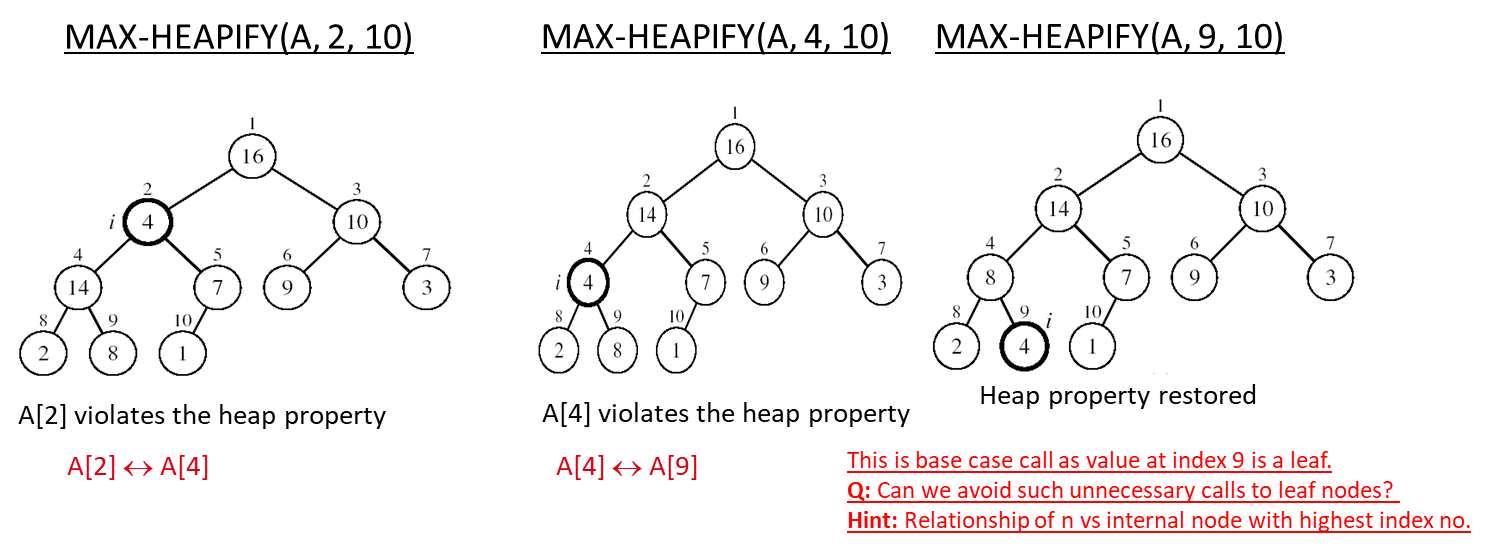
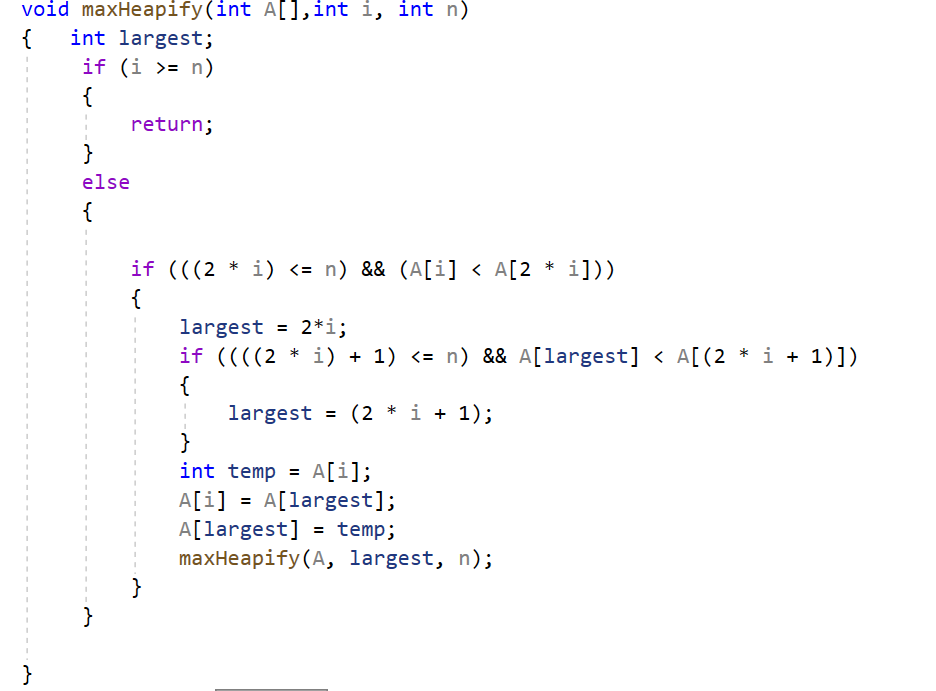
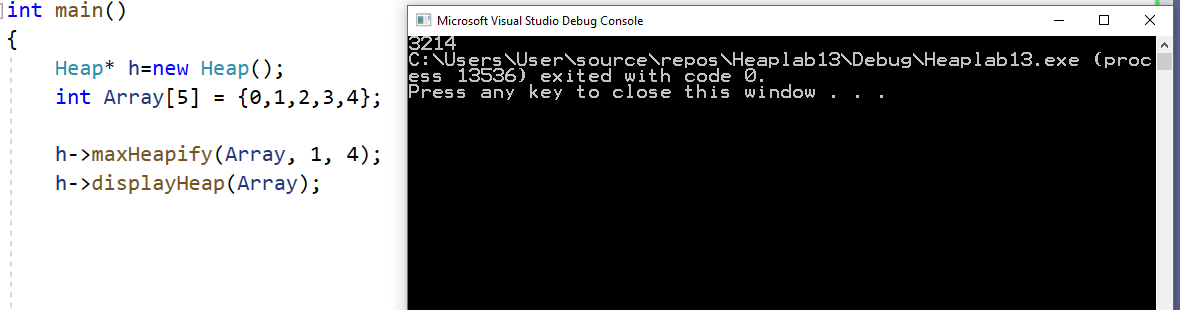
**

Figure : MaxHeapify at index i using Top Down Approach

**CODE**



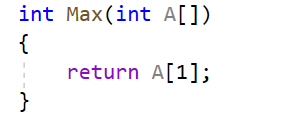
**Output**



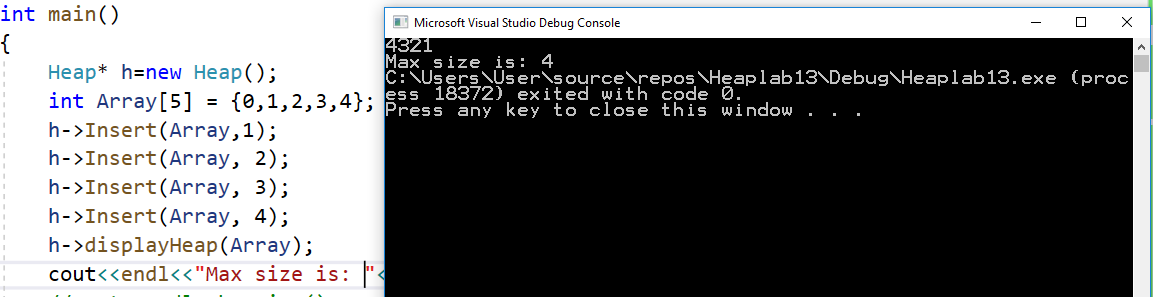
* **findMax()**

Returns the item with the maximum key value, leaving item in the heap.

**CODE**

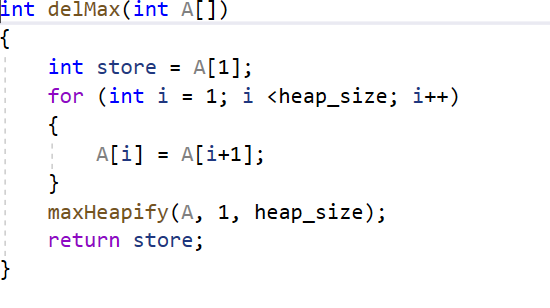


**OUTPUT**

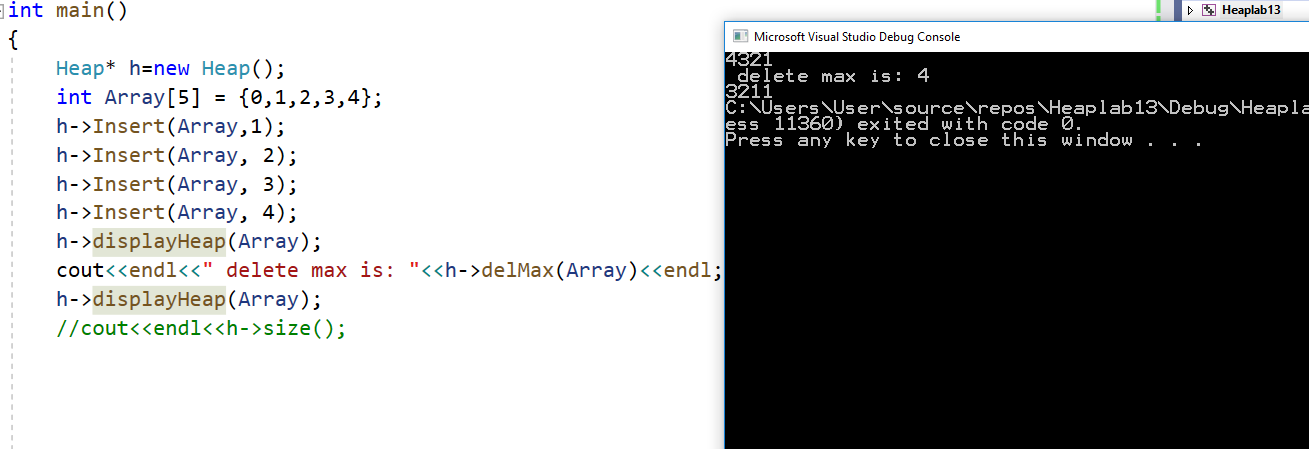


* **delMax()**

Returns the item with the maximum key value, removing the item from the heap. Make sure to call MaxHeapify() so that the value property of max-heap gets restored.

**CODE**

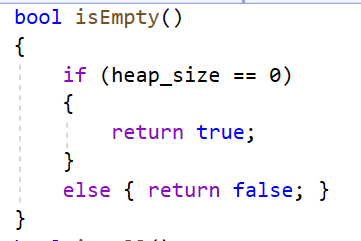
**OUTPUT**



* **isEmpty()**

Returns true if the heap is empty, false otherwise.

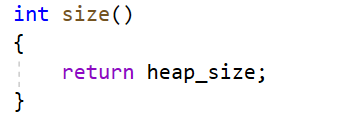
**CODE**



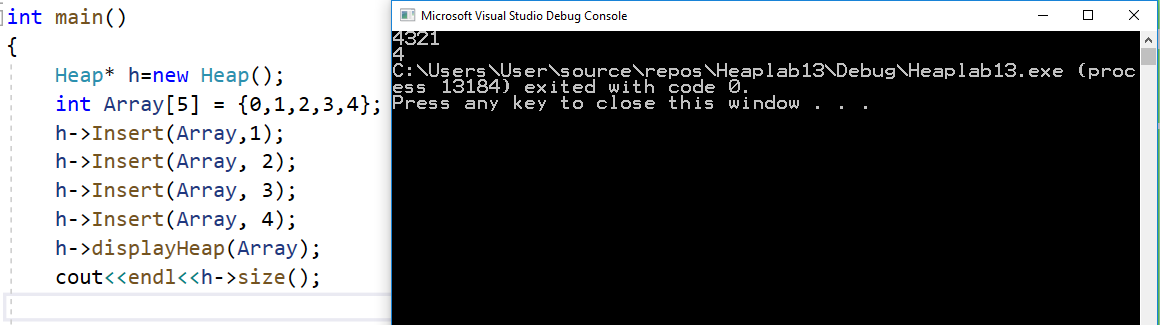
* **size()**

Returns the number of items in the heap.

**CODE**



**Output**



* **BuildHeap(array)** Implement a function that converts a given arbitrary arrayof size **n** into a max-heap.

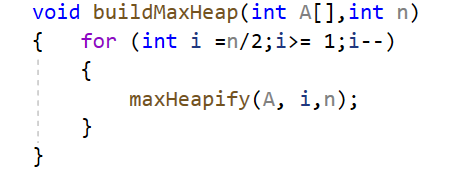
**Build-Max-Heap(A)**

1. *n = length*[A]-1; //remember that index 0 is not used to keep the parent-child relationship.

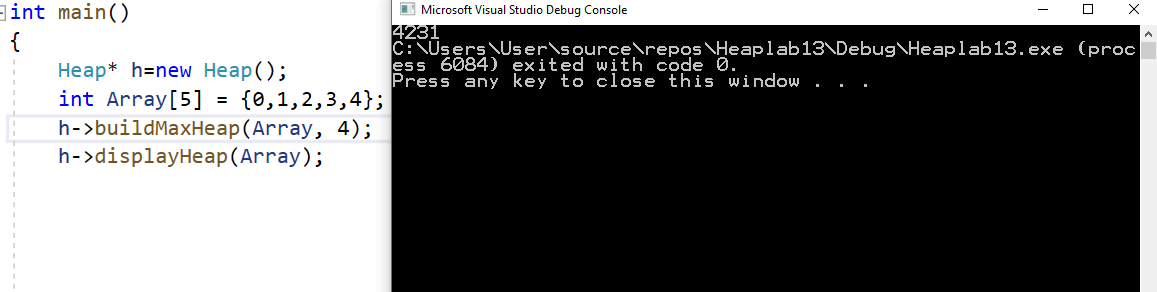
2. for *i* = *n/2* down to 1 // Do you remember why is initialized to from n/2 instead of n?

Write down the reason while submitting your solution.

3. Max-Heapify(A, *i, n*);



**Output**



**PART B: Heap Sort Algorithm**

It sorts an arbitrary list using heap representations. The steps of the heap sort algorithm can be summarized as follows:

* Build a **max-heap** from the given array.
* Swap the **root** (the maximum element) with the **last** element in the array
* **“Discard”** this last node by decreasing the heap size
* Call MAX-HEAPIFY on the new root
* Repeat this process until only one node remains

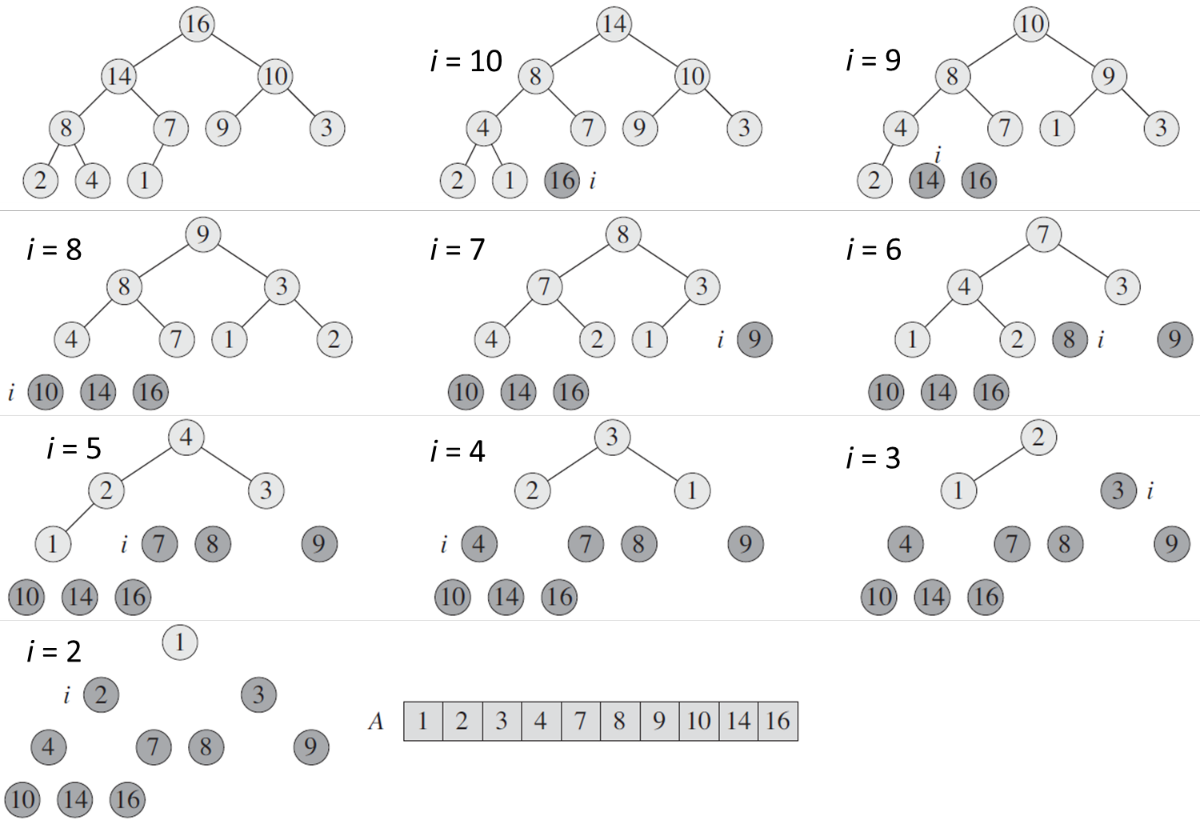


Figure : Example of Heap Sort Algorithm

**CODE**

void HeapSort(int A[])

{

if (heap\_size == 0)

{

return;

}

else

{

temp = A[1];

A[1] = A[heap\_size];

A[heap\_size] = temp;

heap\_size--;

cout << endl;

cout <<"heap\_size: " <<heap\_size<<endl;

maxHeapify(A, 1,heap\_size);

displayHeap(A);

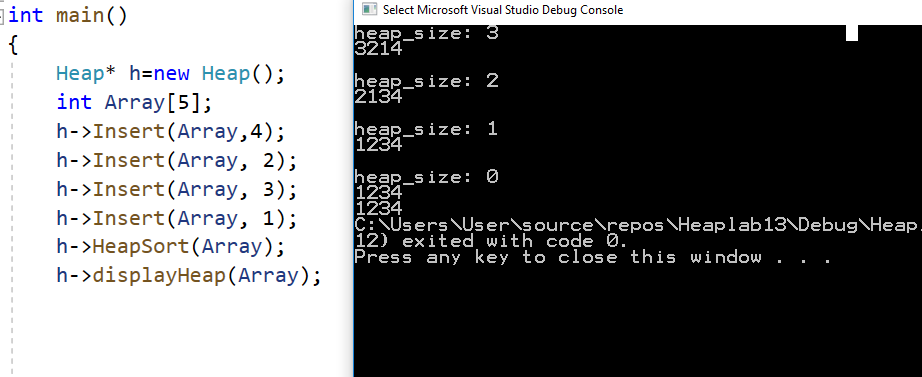
cout << endl;

HeapSort(A);

}

}

**OUTPUT**



**Deliverables**

Students are required to upload the lab on LMS before deadline.

**Note:** Use proper indentation and comments. Lack of comments and indentation will result in deduction of marks. You will submit your workingcodes in **word document** (do **NOT** take screenshot of code, just copy your code and paste it). The name of word document should follow this format. i.e. **YOUR\_NAME\_Lab#**