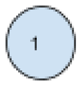
HOMEWORK-9
Total Points: 62

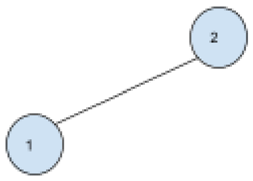
1. [12 Points] Insert 1, 2, 3, 4, 5, 6, 7, 9, 8, 10, 11, 13, 12 one by one in a max-heap. Show the max-heap after every insertion.

   Give one point for each correct tree after the insertion. Maximum points 12.

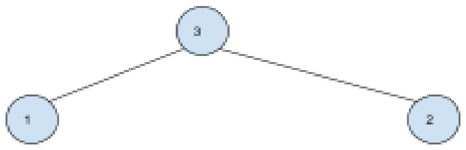
   After first insertion

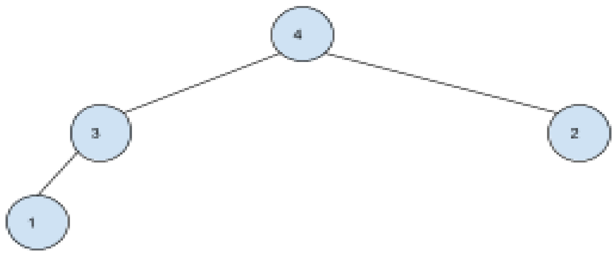
   After second insertion

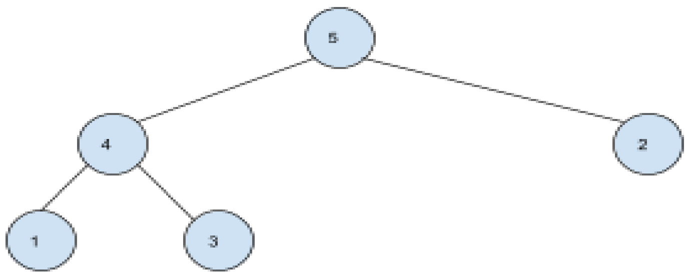
   After third insertion

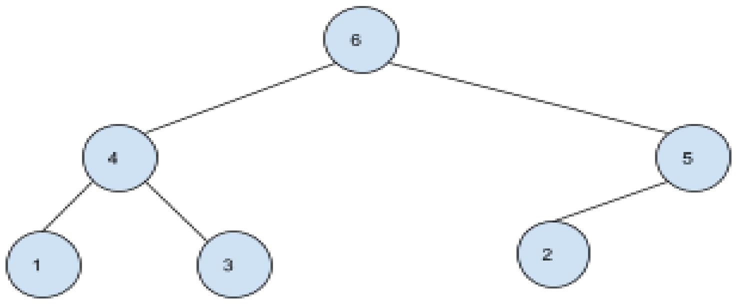
   After fourth insertion

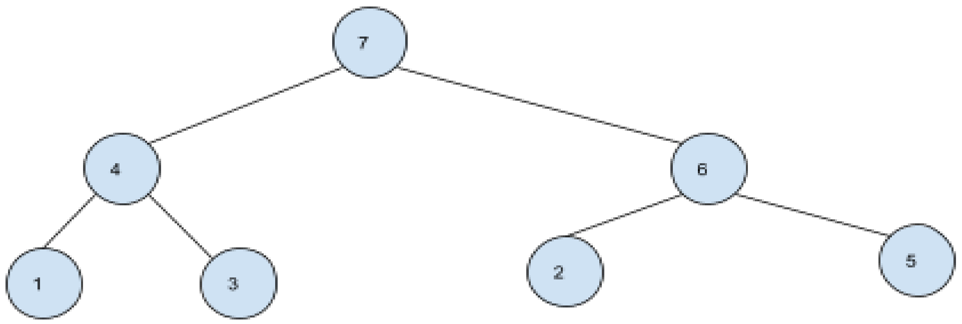
After fifth insertion



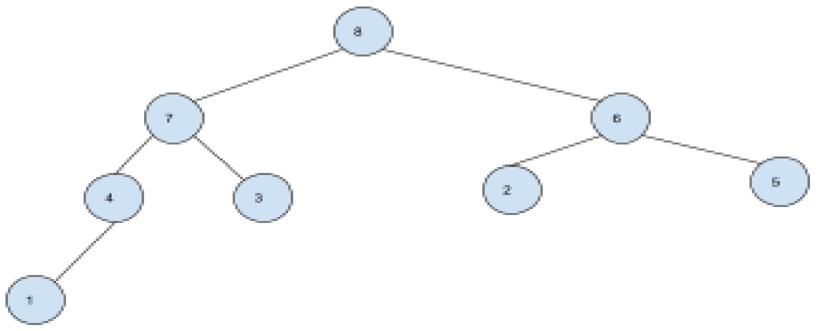
After sixth insertion



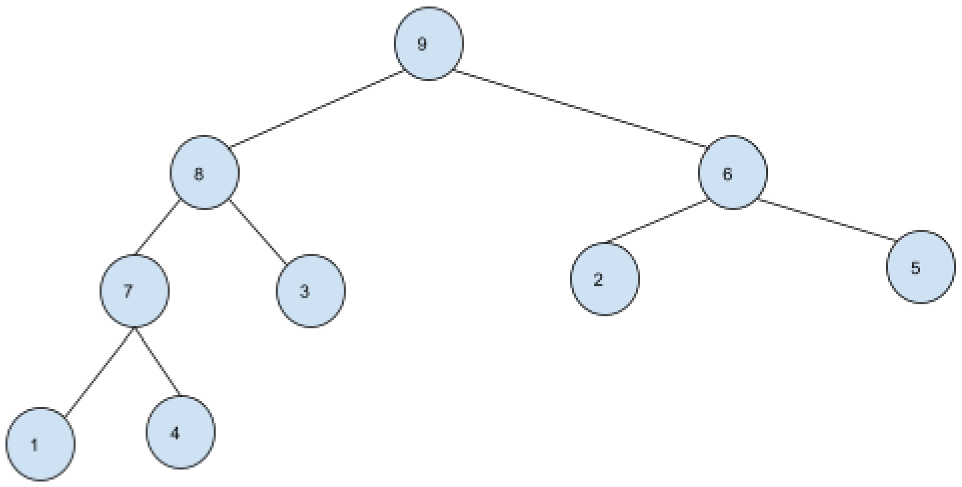
After seventh insertion



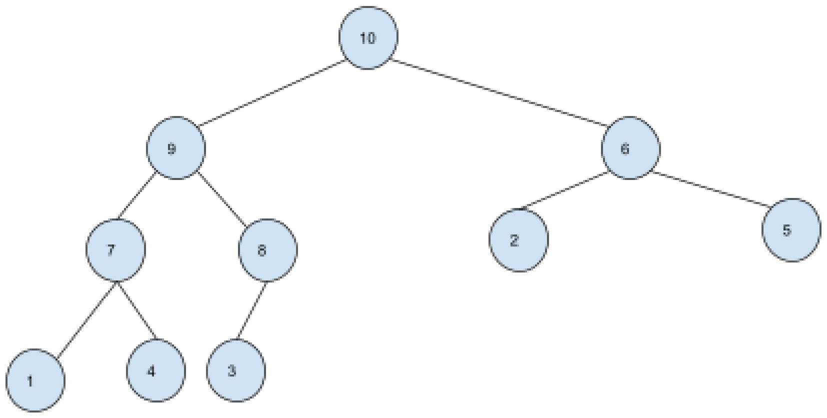
After eighth insertion



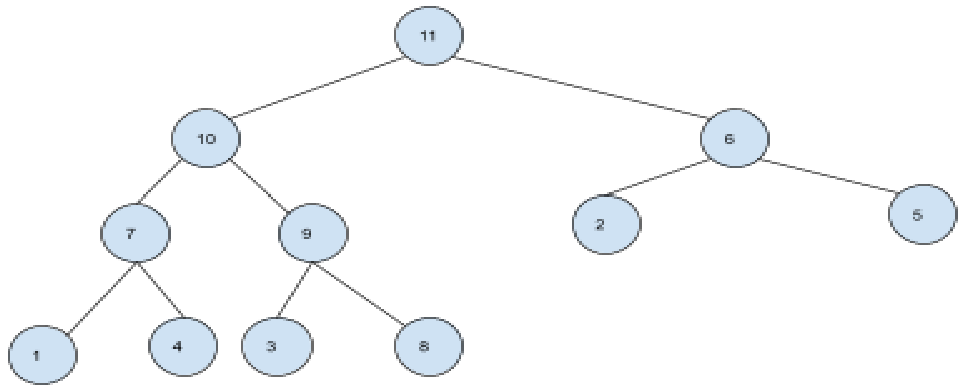
After ninth insertion



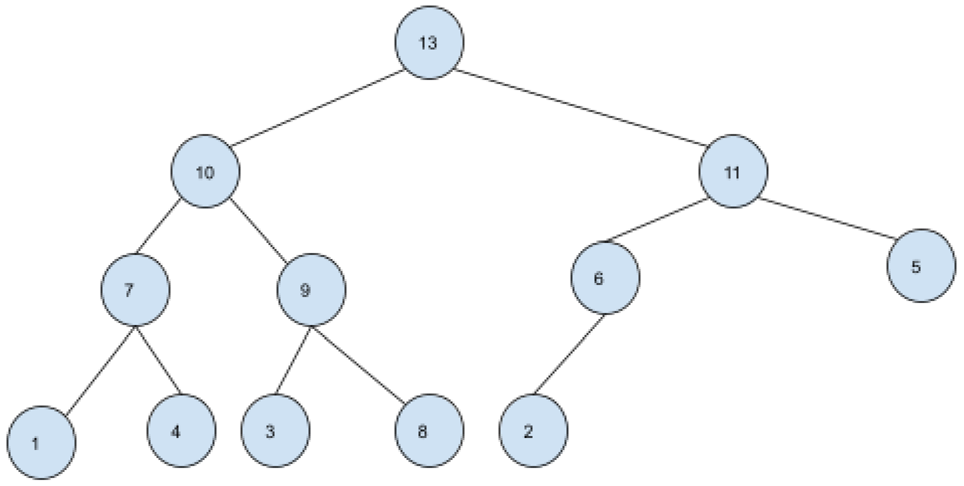
After tenth insertion



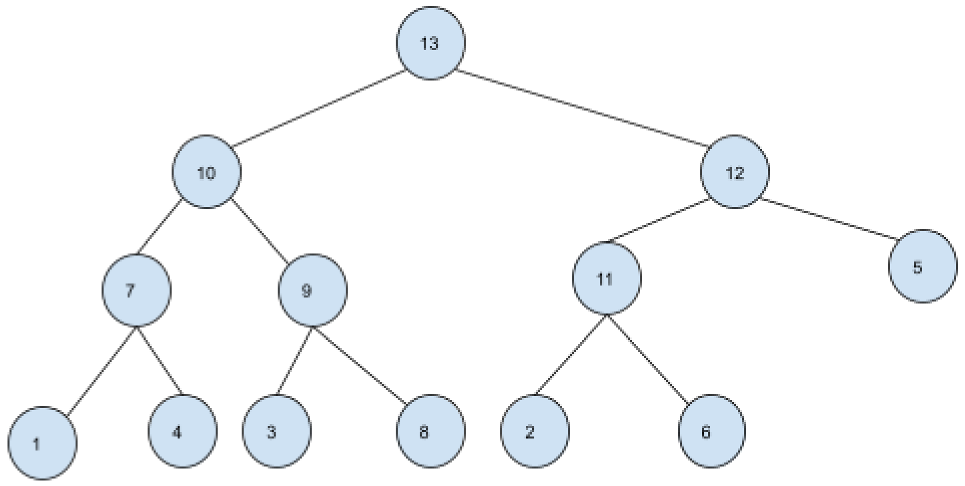
After eleventh insertion



After twelfth insertion

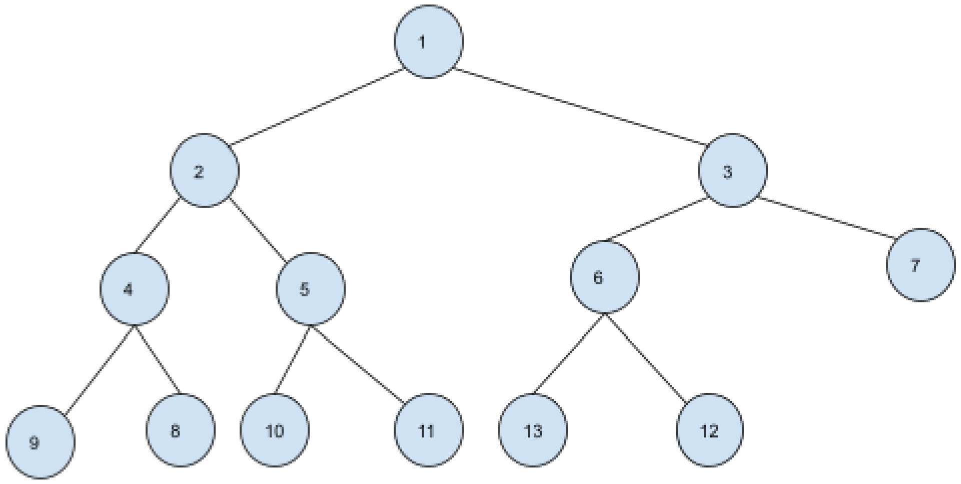


After thirteenth insertion

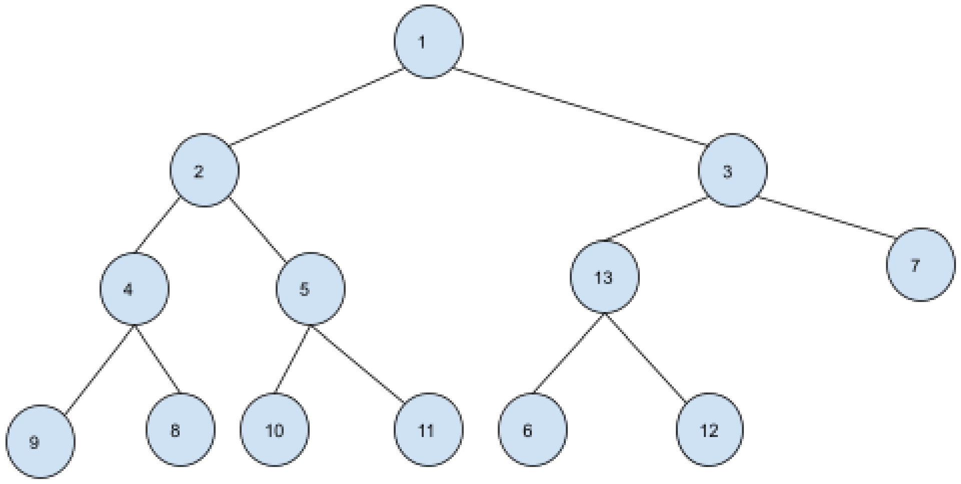


2. [12 Points] Build a max-heap from the given numbers 1, 2, 3, 4, 5, 6, 7, 9, 8, 10, 11, 13, 12 using an O(n) algorithm. All the numbers are already given to you in an array starting at index 1. Show all intermediate steps.

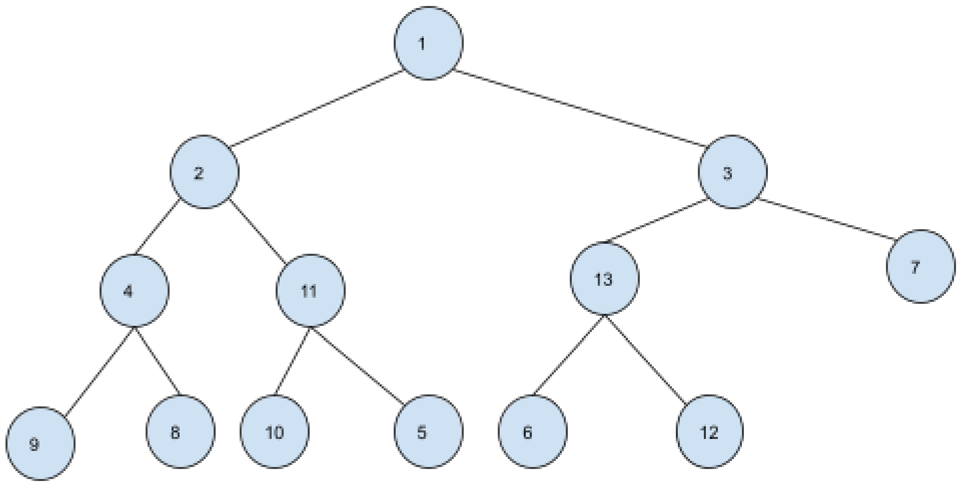
After first heapify [1 point]



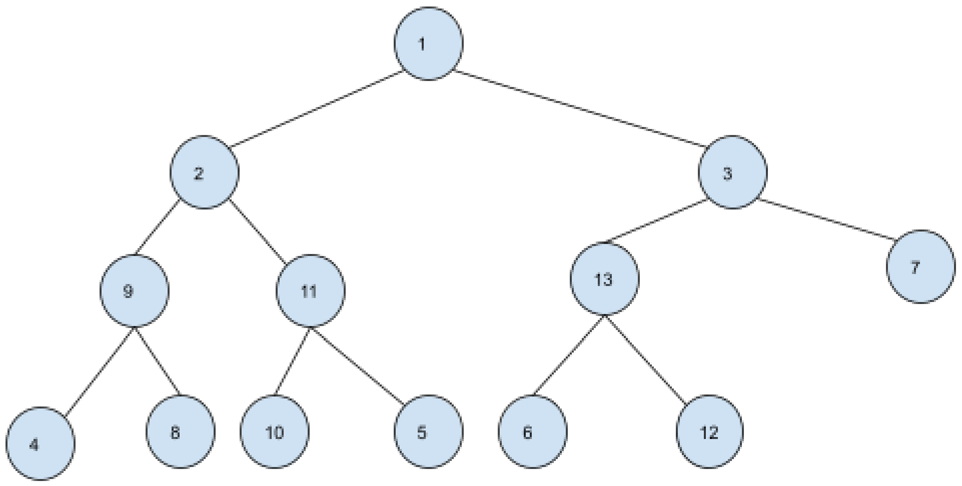
After second heapify [1 point]



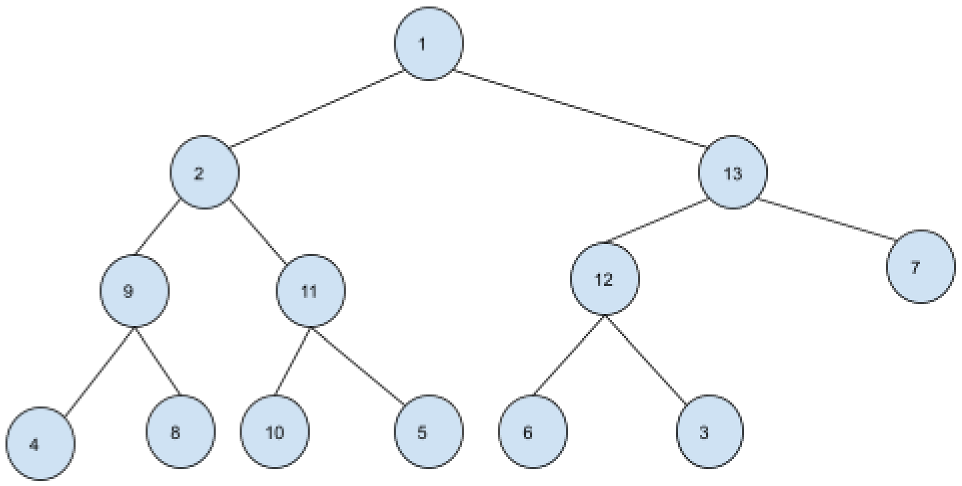
After third heapify [1 point]



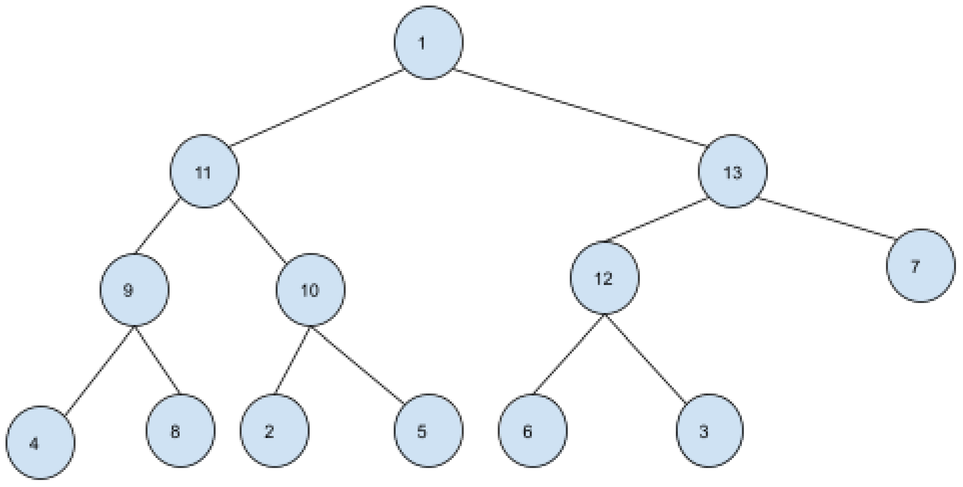
After fourth heapify [2 points]



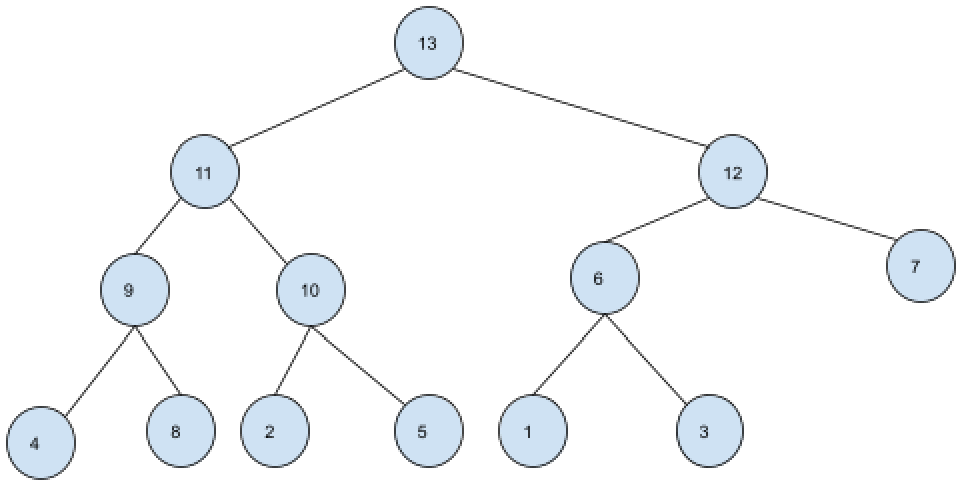
After fifth heapify [2 points]



After sixth heapify [2 points]



After seventh heapify [3 points]



3. [20 Points] We want to store (key, value) pairs of type “struct pair” in a data structure called priority-queue. The priority-queue supports three operations: insert, find, and delete. The insert operation inserts a (key, value) pair in O(log n) operations. The find operation returns a (key, value) pair corresponding to the largest key. If multiple pairs contain the largest key, find returns the one that was inserted first. The time-complexity of the find operation is O(1). The delete operation deletes a pair that contains the largest key. If multiple pairs contain the largest key, the one that was inserted first is deleted. The time complexity of the delete operation is O(log n). Give a pseudocode for the insert, find, and delete operations. What is the type of node in the priority-queue? The type “struct pair” is as follows.

```
struct pair {
  int key, value;
};
```

We can insert the elements in a heap. The type of node in the priority queue is struct entry, as shown below. A global arrival time is maintained, which gets updated every time an entry is added to the priority queue. If multiple entries contain the maximum key, the one with the least arrival time has the highest priority. The corresponding pseudocode is shown below.

Give up to 7 points if the solution is partially correct.

```
struct entry {
  struct pair p;
  int arrival_time;
};

struct Heap {
  struct entry *arr;
  int capacity;  // maximum size of heap
  int heap_size;  // number of elements in the heap
};

PARENT(i)
 floor(i/2)

cmp_entries(e1, e2)
// compare two elements e1 and e2 of type struct entry
// return true if e1 has more priority than e2; otherwise, return false
if e1.p.key == e2.p.key
   return e1.arrival_time < e2.arrival_time
return e1.p.key > e2.p.key

global_arrival_time = 0 // global variable

insert(H, x)
// H is a reference to the heap of type struct Heap
// x is a struct pair input that needs to be inserted
// Output: insert x in the heap

if H->heap_size == H->capacity
   perror("heap overflow")
H->heap_size = H->heap_size + 1
H->arr[H->heap_size].p = x
H->arr[H->heap_size].arrival_time = global_arrival_time
global_arrival_time = global_arrival_time + 1
```

```
i = H->heap_size
while i > 1 and  cmp_entry(H->arr[i], H->arr[PARENT(i)]) == True
   exchange H->arr[i] with H->arr[PARENT(i)]
   i = PARENT(i)
```

find(H)

```
// H is a reference to the heap of type struct Heap
// Output: return the value with maximum key, if multiple
// elements contain the maximum key, return the one that came first
    if H->heap_size == 0
       perror("Heap underflow")
    return H->arr[1].p
```

```
LEFT(i)
  2 * i
RIGHT(i)
  (2 * i) + 1
```

heapify(H, i)

```
// H is a reference to the heap of type struct Heap
// i is the index at which the heapify needs to be performed
// Output: node at index i satisfies the max heap property

l = LEFT(i)
r = RIGHT(i)
if (l <= H->heap_size and  cmp_entry(H->arr[l], H->arr[i])
   largest = l
else largest = i
if r <= H->heap_size and cmp_entry(H->arr[r], H->arr[largest])
   largest = r
if largest != i
   exchange H->arr[i] with H->arr[largest]
   heapify(H, largest)
```

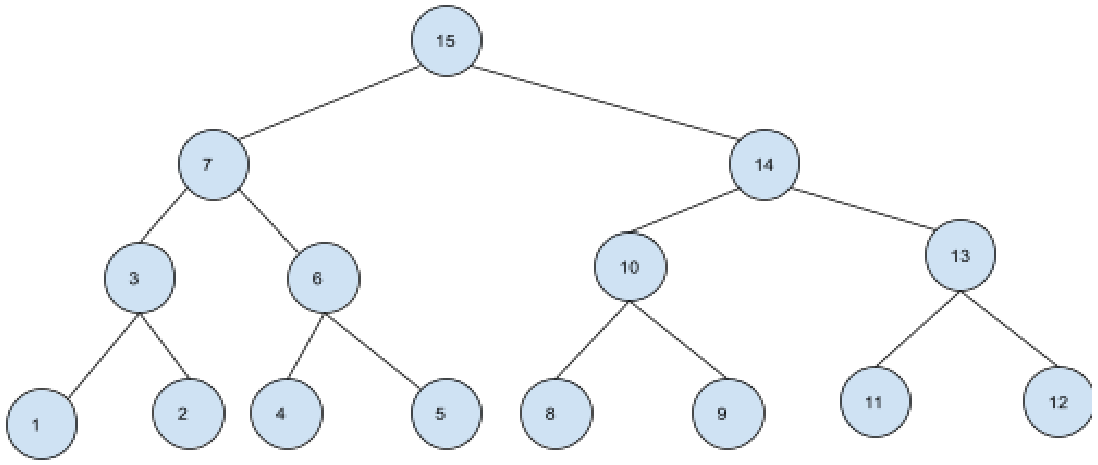
delete(H)

```
// H is a reference to the heap of type struct Heap
// Output: delete the entry with maximum key from the heap, if multiple
// elements contain the maximum key, delete the one that came first
// return the value of the deleted node
max = find(H)
H->arr[1] = H->arr[H->heap_size]
H->heap_size = H->heap_size – 1
heapify(H, 1)
```

```
return max
```

4. [6 Points] Create a max-heap containing 15 numbers 1, 2, 3, …, 15 in such a way that the post-order traversal of the max-heap gives us a sorted sequence in increasing order.

   No partial grading.



5. [4 Points] Give an algorithm to find the minimum element from a max-heap of integers. What is the time complexity of your algorithm?

   The minimum element can be anywhere in the leaves. The pseudocode is shown below. No partial marking.

```
findMin(H)
// H is the input heap
// H->n is the number of elements in the heap
// H->arr is the underlying array in the heap
start = floor(n/2) + 1
min = H->arr[start]
// minimum nodes must be in the leaves
for i in start+1  to n
   if min > H->arr[i]
      min = H->arr[i]
return min
```

6. [4 Points] What are the minimum and maximum number of connected components in a graph with n vertices? Justify your answer.
   Minimum number of connected components = 1, when all vertices are connected **[2 Points]**

Maximum number of connected components = n, when none of the vertices are connected **[2 Points]**

7. [4 Points] What are the minimum and maximum number of edges in a simple undirected graph with n vertices? Justify your answer.
   minimum number of edges = 0 // Graph doesn't need to be connected **[2 Points]**
   maximum number of edges = total number of unordered pair of vertices from a set of n vertices = nC2 = n(n-1)/2 **[2 Points]**