U C L A

Computer Science Department

CS 180 **Algorithms & Complexity** **ID: _ _ _-_ _-_ _ _ _**

Each question has <u>20 points</u> **Total Time: <u>1.5 hours</u>** **Tuesday, November 8th**

Problem 1

a. Describe Prim's MST algorithm (in English).

Prim's algorithm is described as follows:

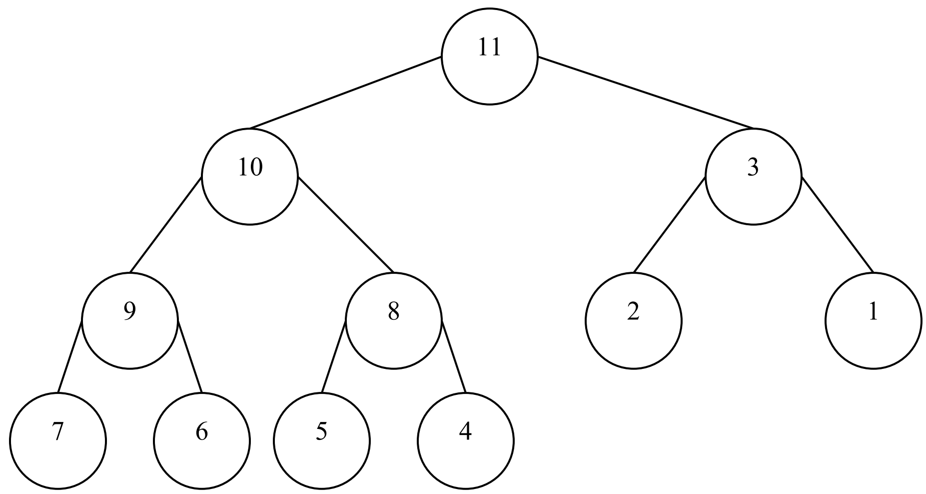
1. Create a tree containing a single vertex, chosen arbitrarily from the graph.
2. Create a set containing all of the edges in the graph
3. Repeat the following two steps until every edge in the set connects two vertices in the tree
    a. Remove from the set an ege with minimum weight that connects a vertex in the tree to a vertex not in the tree
    b. Add that edge to the tree

b. Analyze its time complexity (using a heap).

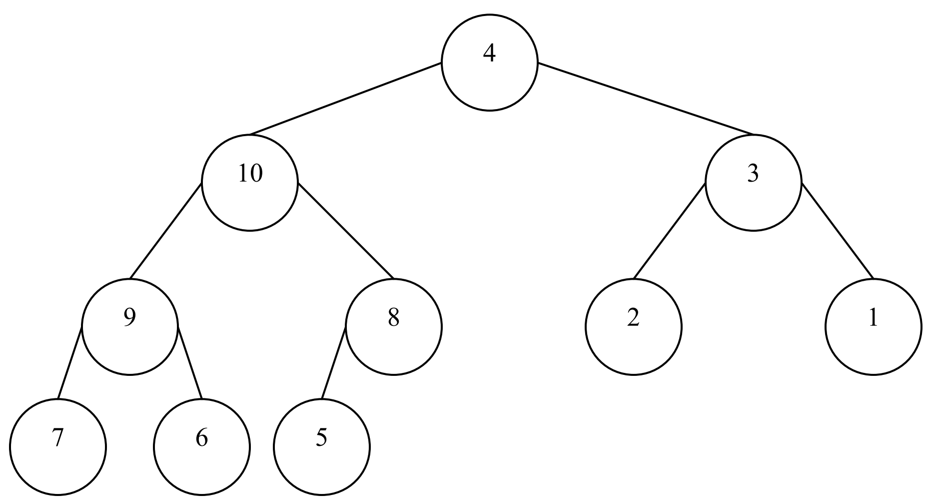
The time complexity of Prim's algorithm is O(mlogn), where m = |E| and n = |V|.

A proof of this statement can be found in the textbook.

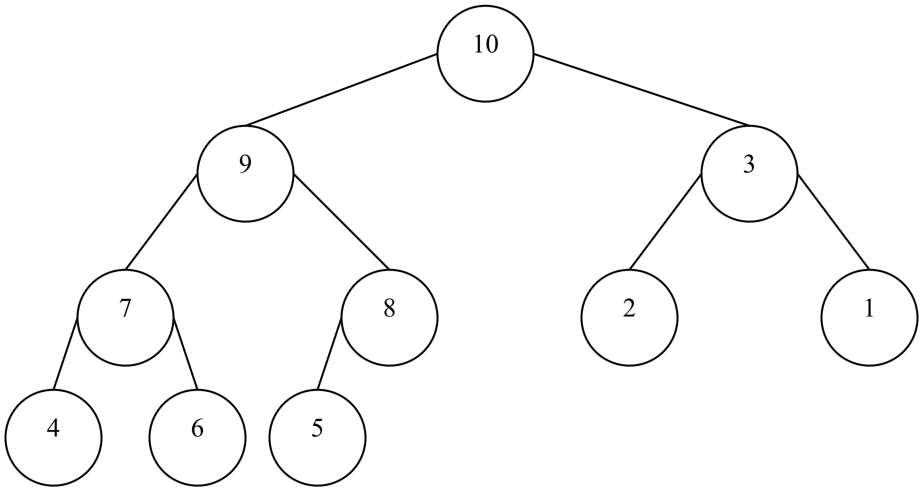
Problem 2. Show each step of delete-max and heapify as you delete 3 numbers (one-by-one) from the following heap.



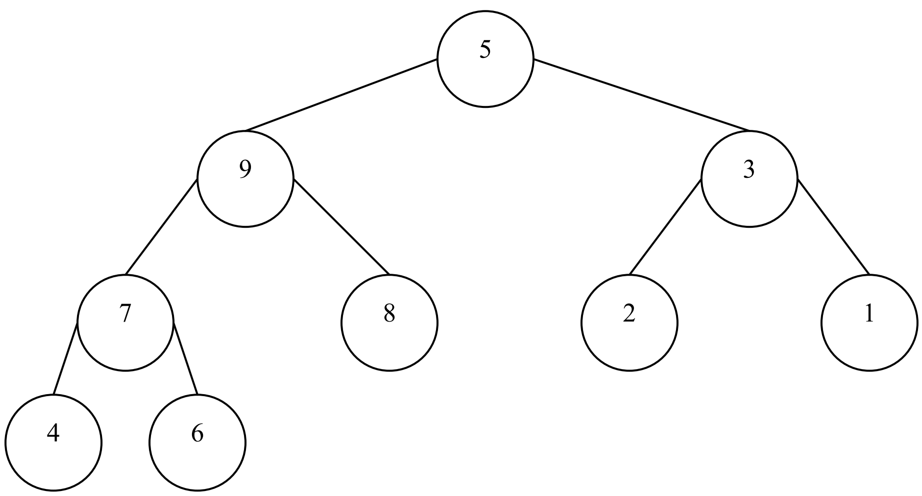
Delete-max:



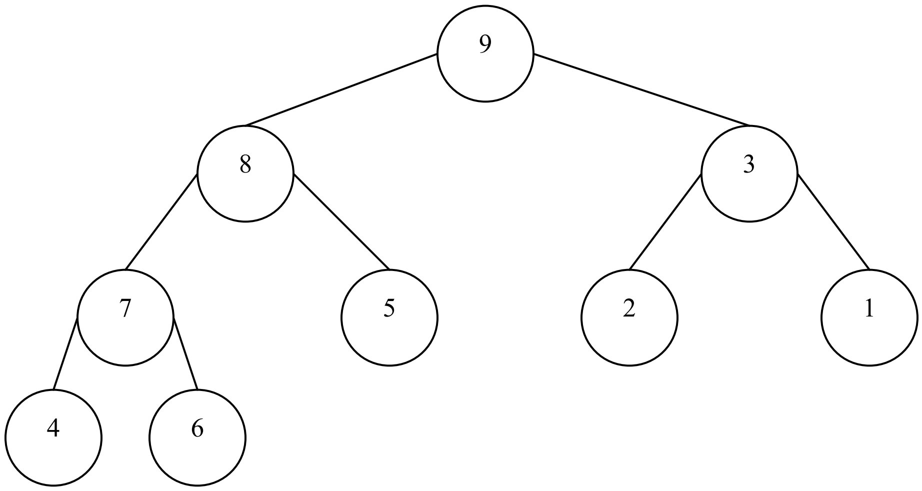
Heapify:



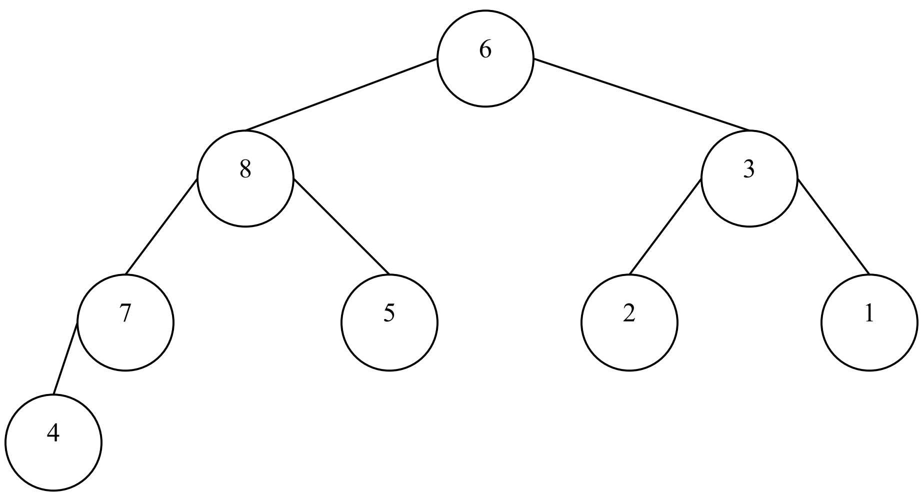
Delete-max:



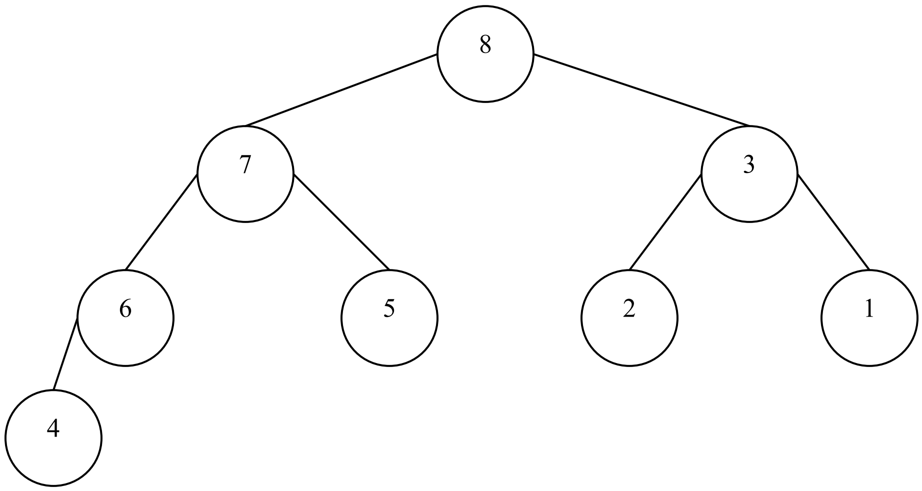
Heapify



Delete-max



Heapify



Problem 3. Consider a sorted sequence $a_1, \ldots, a_n$ of distinct integers.

a. Design an **efficient** algorithm that decides whether there exists an integer a_i such that $a_i = i$ (for example, if the sequence is -1, 3, 4, 5, 7, 9 then the answer is NO).

Let A[1..n] be an array containing the integers.

```
If A[n/2] = n/2
        Then return True
Else if A[n/2] > n/2
        Then recursively examine the subsequence A[1..(n/2)-1]
Else // A[n/2] < n/2
        Then recursively examine the subsequence A[(n/2)+1..n]
Return False
```

b. At most, how many such numbers could there be?

There can be at most n such integers, A[i] = i for i = 1, 2, …, n

c. Analyze the time complexity of your algorithm.

The time complexity is O(logn), analogous to binary search.

Problem 4.

a. Let G be a directed graph with n vertices. Design an efficient algorithm to label all vertices with distinct integers 1 to n such that the label of each vertex is at least one greater than the label of at least one of its predecessors, or to determine that no such labeling is possible.

A source is a vertex whose in-degree is 0
Q : a queue of vertices

```
For each vertex v in G
        If v is a source
                Enqueue v in Q
While Q is not empty
        Remove s, the source at the front of the queue
                Initiate a DFS from s
                (During the DFS, remove each vertex from G as it is popped from
                the stack; assign labels to the vertices in the order in which they
                are discovered)
If any vertices remain in G
        Return the assignment of labels to each vertex
Else
        Return False (i.e. no such labeling is possible)
```

b. Analyze the time complexity of your algorithm.

The time complexity of this algorithm is O(n + m), where n = |V| and m = |E|. The first for-loop takes O(n) time. During the while loop, each vertex in the graph is processed at most once by any DFS; no vertex can be processed more than once by a DFS, since it is removed from G as soon as it is popped from the stack. The cost of a DFS is O(n + m). Therefore, the total time complexity of this algorithm is O(n + m).

Problem 5. Consider n positive integers $d_1, d_2 \ldots d_n$ such that $d_1 + d_2 + \ldots + d_n = 2n-2$.

a. Design an efficient algorithm for constructing a tree with n vertices of degrees exactly $d_1, d_2 \ldots d_n$.

1. For i ← 1 to n
    a. Mark[i] ← FALSE
2. Sort the list of numbers in O(nlogn) time using Heapsort or Mergesort. (After sorting, assume that $d_n \geq d_{n-1} \geq \ldots \geq d_1$
3. Create a vertex v_n, corresponding to d_n (We will let v_n be the root of the tree. v_n will have exactly d_n children)
4. Let $k = d_n$. Let children[v_n] = $\{v_{n-1}, v_{n-2}, \ldots, v_{n-k}\}$. These children will correspond to values in the $d_{n-1}, d_{n-2}, \ldots, d_{n-k}$ sorted list.
5. Attach an edge between v_n and each of its children.
6. Set $d_n = 0$ and remove d_n from the list
7. Mark[n] ← TRUE
8. For i ← 1 to k
    a. $d_{n-i} \leftarrow d_{n-i} - 1$
    b. MARK[n-i] ← TRUE
9. Re-sort the remaining degree sequence.
10. For i ← 1 to k
    a. If $d_{n-i} = 0$
        i. Remove d_{n-i} from
    b. Else
        i. Recursively repeat Steps 3-9 for each child v_{n-i} of v_n with the following change: the selected children of v_{n-i} MUST be satisfy Mark[n-i] = FALSE at each step. (Mark[n-i] = TRUE indicates that the vertex already has a parent. Ensuring that each child vertex has exactly one parent prevents cycles)

b. Prove the correctness of your algorithm.

Before the formal proof, we acknowledge a few facts that are well-known from graph theory:

1. A tree with n vertices has EXACTLY m = n – 1 edges
2. A tree cannot contain cycles
3. $\sum_{v \in V} \deg[v] = 2m$ (= 2n – 2 for trees)

The algorithm describe above is greedy. To prove correctness we must establish that the problem exhibits *optimal substructure* and that the *greedy choice property* is satisfied.

1. Optimal Substructure

To see that the problem exhibits the greedy choice property, observe that each vertex in the tree is the root of its own sub-tree. The array Mark (coupled with the Else statement in step 10) ensures that each vertex is added to the tree exactly once, and that each vertex (other than the root of the tree) has exactly one parent when it is added to the tree.

2. Greedy Choice Property

   The first vertex, v_n (the root of the tree) has exactly d_n children. Each other vertex v_{n-i} has exactly 1 parent and $(d_{n-i} - 1)$ children. Correctness follows from the fact that the degree of each child vertex is decremented as it is added to the tree. Property 3 above ensures that the degrees of all of the vertices in the tree add up to twice the number of edges.