**1a. Using the simplest binary search tree insertion algorithm, show the tree that results after inserting into the above tree the nodes 80, 65, 75, 15, 35 and 25 in that order.**

10

25

35

15

75

65

80

70

30

40

60

20

50

**1b. After inserting the nodes mentioned in part a, what is the resulting BST after you delete the node 30, then the node 20?**

35

75

65

80

70

25

40

60

50

15

10

**1c. After inserting the nodes mentioned in part a, what would be printed out by in-order, pre-order, and post-order traversals of the tree (assume your traversal function prints out the number at each node as it is visited)?**

In-Order: 10, 15, 20, 25, 30, 35, 40, 50, 60, 65, 70, 75, 80

Pre-Order: 50, 20, 10, 15, 40, 30, 25, 35, 60, 70, 65, 80, 75

Post-Order: 15, 10, 25, 35, 30, 40, 20, 65, 75, 80, 70, 60, 50

**2a. Show the resulting heap.**

2

3

7

4

5

1

**2b. Show how your heap from part a would be represented in an array.**

4

2

1

5

3

7

(Front) (Back) int count = 6;

**2c. Remove the top item from the heap and show the resulting array after the removal operation.**

5

3

4

1

2

(Front) (Back) int count = 5;

**3a. Show a C++ structure/class definition for a binary tree node that has both child node pointers and a parent node pointer.**

struct Node

{

Node(int value, Node\* parent)

: data(value), left(NULL), right(NULL), m\_parent(parent) {}

Node\* m\_parent;

Node\* left; //child

Node\* right; //child

int data;

};

**3b. Write pseudocode to insert a new node into a binary search tree with parent pointers.**

insert(Node\* node, Node\* root)

if root is NULL, set node to the root and node’s parent pointer to NULL

else if node’s value is smaller than root’s value

if root has no left child

set root’s left pointer to node and node’s parent pointer to root

else pass in the root’s left child as root in another call to insert

else if node’s value is greater than root’s value

if root has no right child

set root’s right pointer to node and node’s parent pointer to root

else pass in the root’s right child as root in another call to insert

**4a.** O(C+S)

**4b.** O(log(C) + S)

**4c.** O(log(C) + log(S))

**4d.** O(log(S))

**4e.** O(1)

**4f.** O(log(C) + S)

**4g.** O(Slog(S))

**4h.** O(Clog(S))