1)

a.

50

20 60

10 40 70

15 30 65 80

25 38 72

b.

Pre-order:

50

20

10

15

40

30

25

38

60

70

65

80

72

In-Order:

10

15

20

25

30

38

40

50

60

65

70

72

80

Post-order:

15

10

25

38

30

40

20

65

72

80

70

60

50

c.

After deleting node 30, we get

50

20 60

10 40 70

15 25 65 80

38 72

//Note: we could have replaced 30 with 38 as well, but I chose 25 instead

After deleting 20, we get

50

15 60

10 40 70

25 65 80

38 72

2.

a.

struct Node

{

Node(int n)

: l\_child(nullptr), r\_child(nullptr), m\_parent(nullptr), m\_value(n)

{}

~Node() {}

Node\* l\_child;

Node\* r\_child;

Node\* m\_parent;

int m\_value;

};

b.

void insertNode(int n) //since spec doesn’t forbid duplicates, I allowed it,

if (bstPtr is nullptr) //in case bst doesn’t have any nodes

set m\_root to newNodePtr with value n

return

let current equal m\_root

while we are not done //until we insert new node

if this value is less than current value

if current’s left child pointer is not nullptr

set current to left child ptr

else

set current’s parent pointer to this current node

set current’s l\_child ptr to a new Node

return

else //case when this value is greater than or equal to current

if current’s right child is not a nullptr

set current to its right child

else

set current’s parent pointer to this current node

let curren’ts r\_child ptr equal a new Node

return

3.

a.

7

3 6

0 2 4

b.

int arrHeap[6] = {7, 3, 6, 0, 2, 4};

c.

int arr\_heap[5] = {6, 3, 4, 0, 2};

4. Note: I put explanations for myself to learn, so please don’t take points off

a.

need to find course c in the vector, so O of C there, plus for a list, we may need to traverse through all S elements in list.

O(C + S)

b.

we can do binary search for course c in map, but then we may need to check all S elements in list.

O(logC + S)

c.

map is bst, and so is set.

O(logC + logS)

d.

unorder\_map is basically a hash table, and its speed is O(1), regardless of # of items, and set is bst

plus we’re told unordered\_map and unordered\_set are hash tables that never allow load factor exceed constant

O(1 + logS) = O(logS)

e.

O(1 + 1) = O(1)

f.

O(logC \* + S)

g. //double check

O(1 + S) = O(S)

h. //double check

O(C + logS)