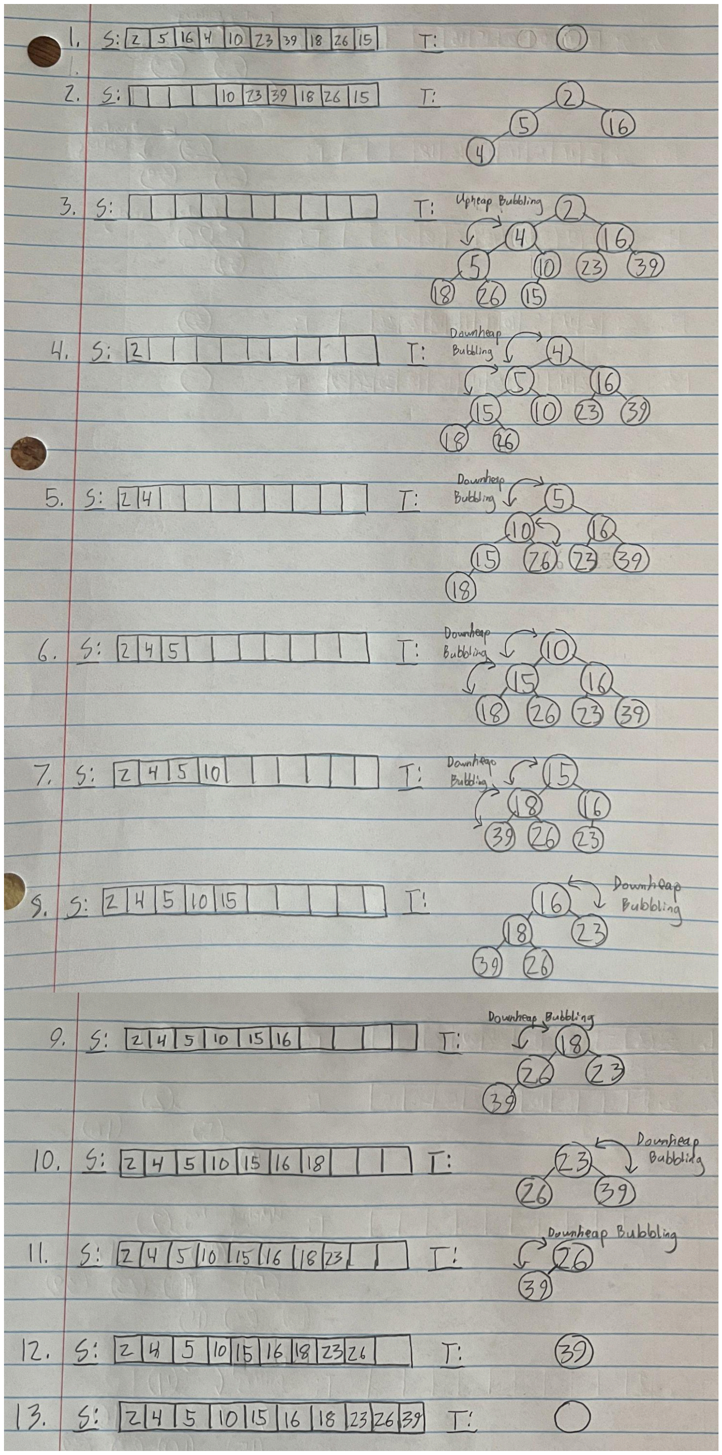
Problem #1 **Name:** Alexander Michelbrink

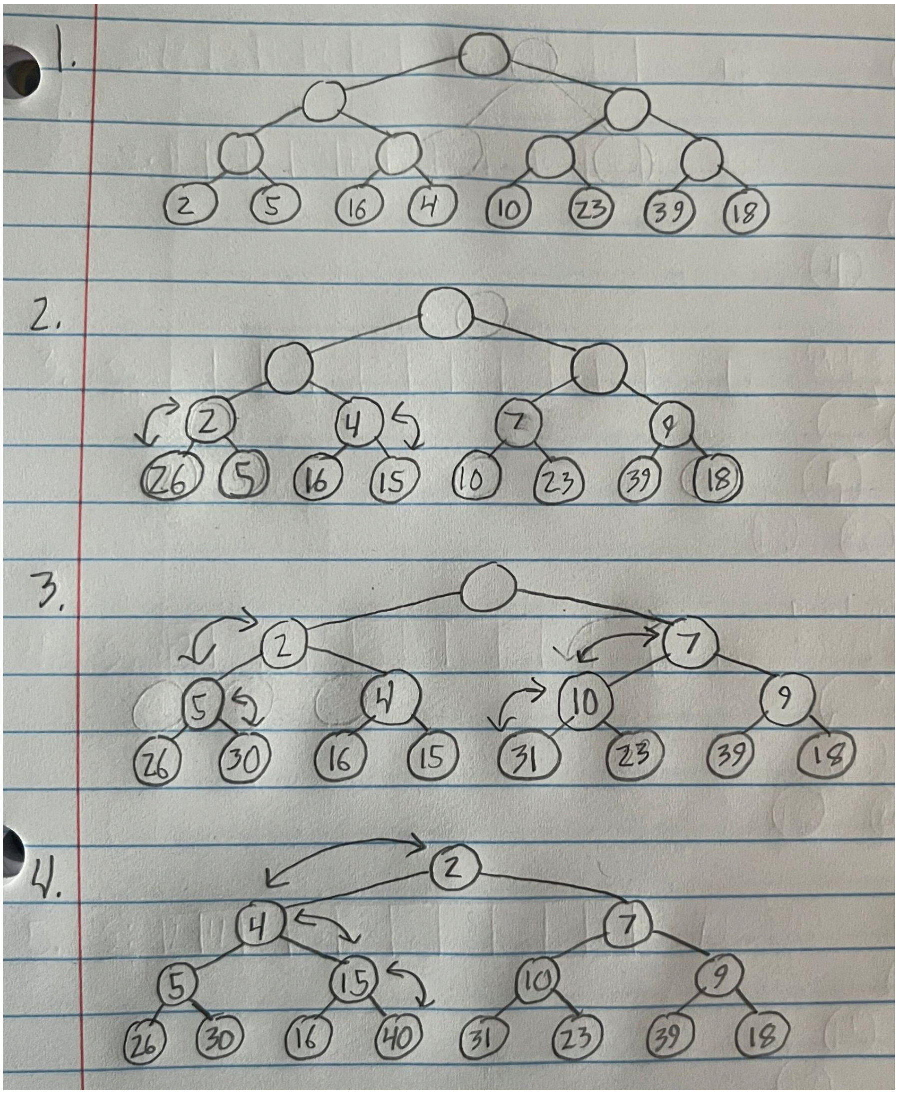
A.) ___

In the following steps of heap-sort shown, S represents the sequence and T represents the heap. The leaves are not shown in the heap diagrams for simplicity, but their existence is implied for each external node of T.



B.) ____

In the following steps of the bottom-up construction of a minimum heap, the leaves of each external node are once again not shown in the diagram for simplicity. Although, their existence is still implied for each external node.



Problem #2

This algorithm assumes…

- We have a key(***v***) method that returns the key of node ***v*** and that operates in O(1) time.
- We begin the algorithm by assigning ***v*** in the initial call of printSmallerAndEqual(***v,x***) to the root of ***T*** (root(***T***)).

Pseudocode:

```
Algorithm printSmallerAndEqual(v,x)
     if ( key(v) ≤ x )
          print( key(v) )
     if ( leftChild(v) AND key(leftChild(v)) ≤ x )
          printSmallerAndEqual(leftChild(v),x)
     if ( rightChild(v) AND key(rightChild(v)) ≤ x )
          printSmallerAndEqual(rightChild(v),x)
```

This algorithm runs in O(*k*) time.

Since each method and the comparisons used in the algorithm operate in O(1) time, we can focus on how many times the algorithm recursively calls itself for any heap ***T*** storing ***n*** keys. Per the second and third if-statements, the algorithm only makes a recursive call if the key of the left and/or right child of the current node ***v*** is less-than-or-equal to the value specified by ***x***. Therefore, when starting from the root, there will only be as many recursive calls as there are keys to print (***k***). This means the algorithm runs in O(*k*) time.

Problem #3

A.)

E_1	h(k) = 4	4
A	h(k) = 0	0
S_1	h(k) = 5	5
Y	h(k) = 11	11
Q	h(k) = 3	3
U	h(k) = 7	7
E_2	h(k) = 4	4 → 5 → 6
S_2	h(k) = 5	5 → 6 → 7 → 8
T	h(k) = 6	6 → 7 → 8 → 9
I	h(k) = 8	8 → 9 → 10
O	h(k) = 1	1
N	h(k) = 0	0 → 1 → 2

Resulting Hash Table

Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Key	A	O	N	Q	E_1	S_1	E_2	U	S_2	T	I	Y	

B.)

Double-Hashing Equation: $Index = (\, h(k) + j * h'(k) \,) \bmod N$

$$Index = (\,(k \bmod 13) + j * (\,1 + (k \bmod 11)\,)\,) \bmod 13$$

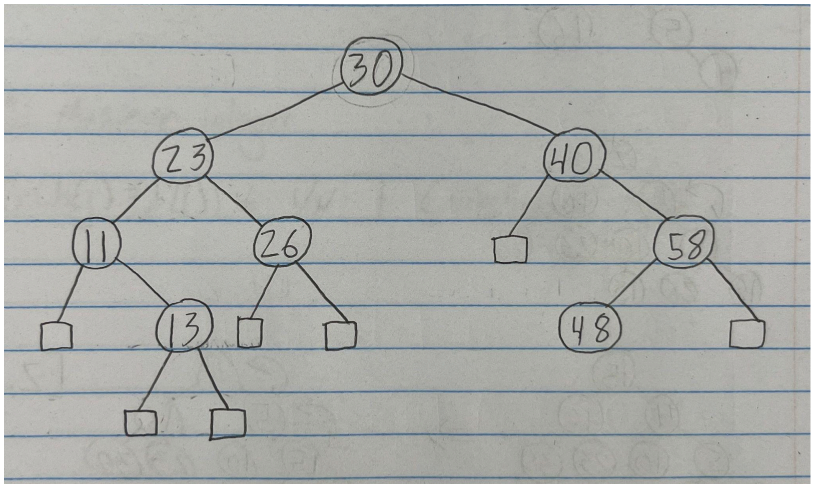
E_1	h(k) = 4, h'(k) = 5	4
A	h(k) = 0, h'(k) = 1	0
S_1	h(k) = 5, h'(k) = 8	5
Y	h(k) = 11, h'(k) = 3	11
Q	h(k) = 3, h'(k) = 6	3
U	h(k) = 7, h'(k) = 10	7
E_2	h(k) = 4, h'(k) = 5	4 → 9
S_2	h(k) = 5, h'(k) = 8	5 → 0 → 8
T	h(k) = 6, h'(k) = 9	6
I	h(k) = 8, h'(k) = 9	8 → 4 → 0 → 9 → 5 → 1
O	h(k) = 1, h'(k) = 4	1 → 5 → 9 → 0 → 4 → 8 → 12
N	h(k) = 0, h'(k) = 3	0 → 3 → 6 → 9 → 12 → 2

Resulting Hash Table

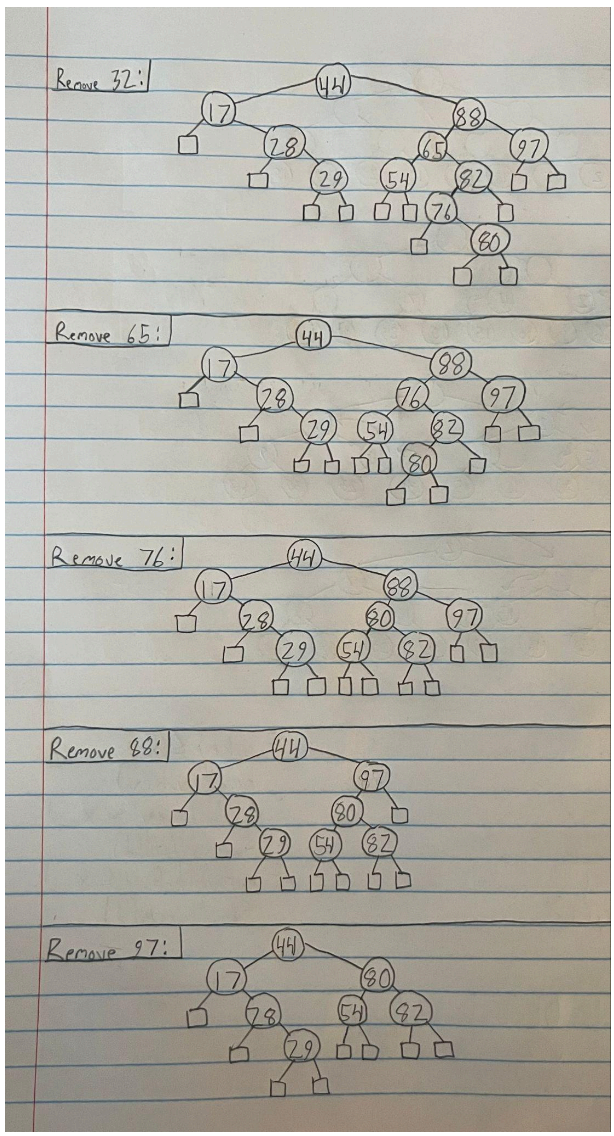
Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Key	A	I	N	Q	E_1	S_1	T	U	S_2	E_2		Y	O

Problem #4

A.)


30
23
40
11
26
58
13
48

B.)

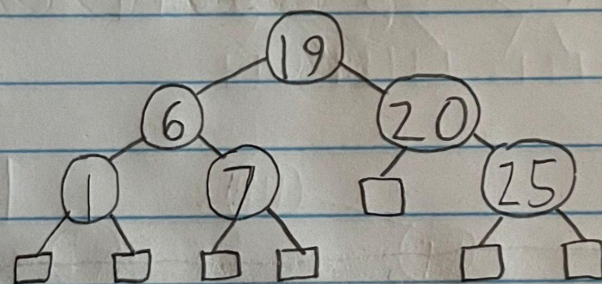

Remove 32:
Remove 65:
Remove 76:
Remove 88:
Remove 97:

C.)

Sequence: 1, 19, 6, 7, 20, 25

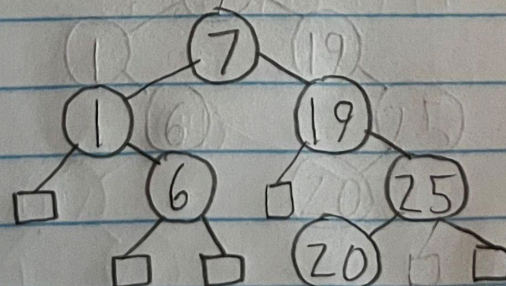
Order #1: 19, 6, 1, 7, 20, 25

Tree #1:



Order #2: 7, 1, 19, 6, 25, 20

Tree #2:



Problem #5

Pseudocode:

```
Algorithm smallestLargerThan(T,x)

    v ← T.root()
    y ← x

    foundMin ← false
    while ( !(foundMin) )

        if ( key(v) > x )
            y ← key(v)

        if ( key(v) > x AND leftChild(v) )
            v ← leftChild(v)
        else if ( key(v) ≤ x AND rightChild(v) )
            v ← rightChild(v)
        else
            foundMin = true

    if (y > x)
        return y
    else
        throw noValuesLargerThanXException
```

The algorithm operates in O(*h*) time where *h* is the height of the binary search tree.

Looking at the pseudocode, the assignments, comparisons, and methods used operate in O(1) time. Therefore, we can look at how many times the while-loop executes in order to find the worst-case running time. The loop will iterate until there are no viable children of *v* that *v* can then be assigned to. The worst case would be if an external node on the bottom layer of the tree was the last viable child of v. This means that the maximum number of iterations is *h* since there is no upward movement in the tree, only downward.