Christopher Tam
250803892

Question 1)

(a) The array A and the auxiliary array C after line 5 of the COUNTING-SORT algorithm.

A: (indexes range from 0 to 11, left to right)

6	0	2	0	1	3	4	6	1	3	2

C: (indexes range from 0 to 6, left to right)

2	2	2	2	1	0	2

(b) The array C after line 8 of the COUNTING-SORT algorithm.

C: (indexes range from 0 to 6, left to right)

2	4	6	8	9	9	11

(c) The array B and the auxiliary array C after *one* iteration of the loop in lines 10-12 of the COUNTING-SORT algorithm.

B: (indexes range from 0 to 11, left to right)

						2				

C: (indexes range from 0 to 6, left to right)

2	4	5	8	9	9	11

(d) The array B and the auxiliary array C after *two* iterations of the loop in lines 10-12 of the COUNTING-SORT algorithm.

B: (indexes range from 0 to 11, left to right)

					2		3			

C: (indexes range from 0 to 6, left to right)

2	4	5	7	9	9	11

(e) The array B and the auxiliary array C after *three* iterations of the loop in lines 10-12 of the COUNTING-SORT algorithm.

B: (indexes range from 0 to 11, left to right)

			1		2		3			

Christopher Tam
250803892

C: (indexes range from 0 to 6, left to right)

2	3	5	7	9	9	11

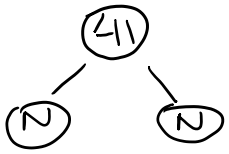
(f) The final sorted output array B

A: (indexes range from 0 to 11, left to right)

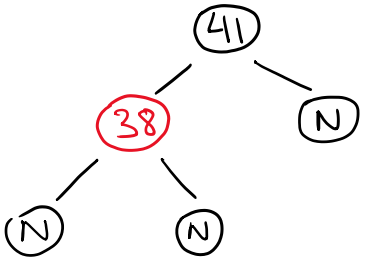
0	0	1	1	2	2	3	3	4	6	6

Question 2)

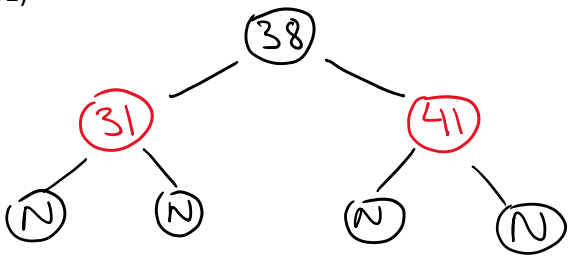
(Insert 41)



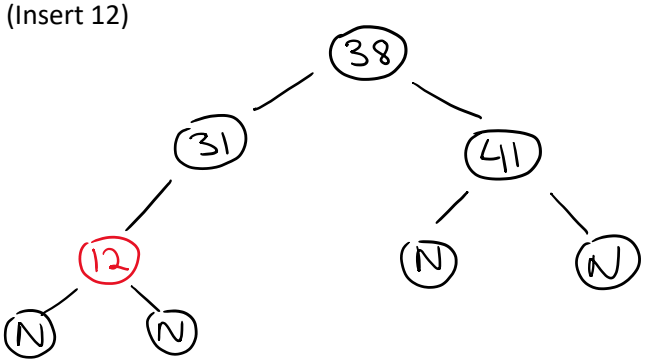
(Insert 38)



(Insert 31)

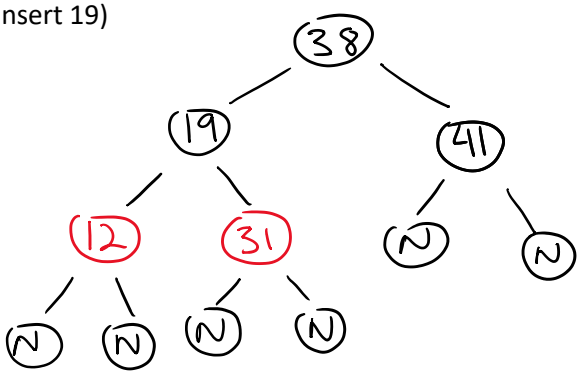


(Insert 12)

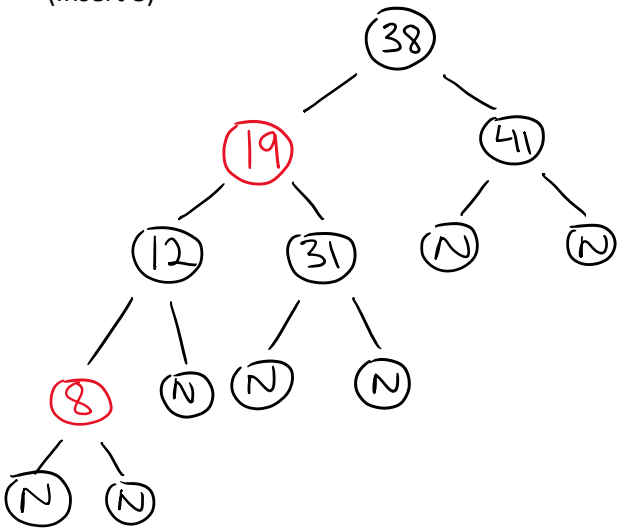


Christopher Tam
250803892

(Insert 19)

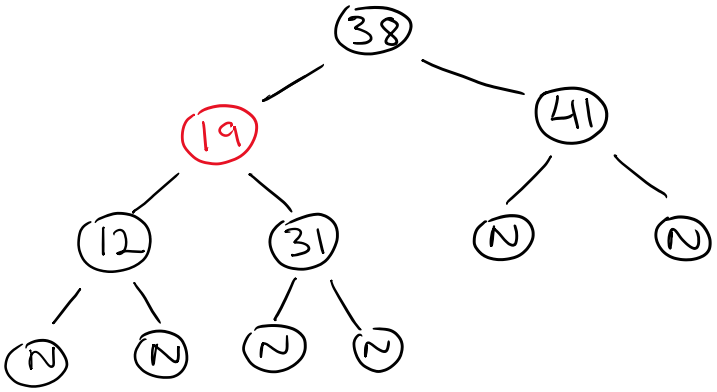


(Insert 8)



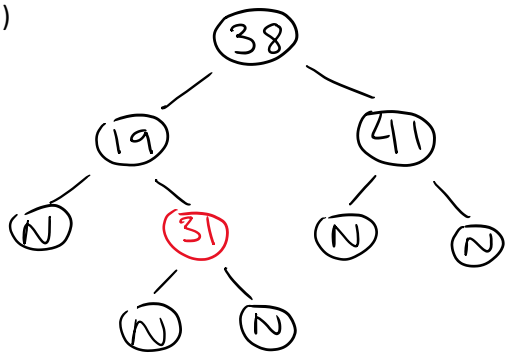
Question 3)

(Delete 8)

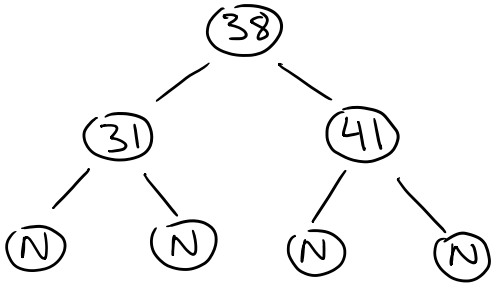


Christopher Tam
250803892

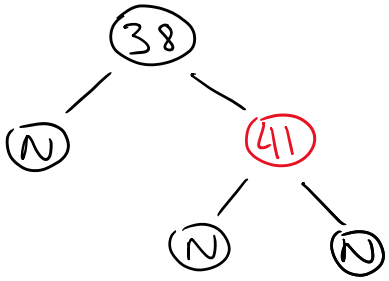
(Delete 12)



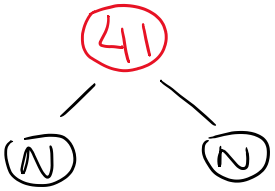
(Delete 19)



(Delete 31)



(Delete 38)



(Delete 41)

Christopher Tam
250803892

Question 4)

The algorithm to sort kn elements into one sorted sequence goes as follows:

1. Create an output array of size $k*n$
2. Create a min heap of size k and insert the first element of every sequence into the heap
3. Repeat the following steps $k*n$ times:
    a. Get the minimum element from the heap (since we have a min heap the minimum will always be at the root) and store it in the output array
    b. Replace the heap root with the next element from the sequence from which the element was extracted. If the array does not have any more elements, the replace the root with infinite.
    c. After replacing the root, heapify the tree.

This algorithm works because it essentially merges previously sorted but distinct arrays. When the min heap is created, one by one it gets the minimum element from the heap and stores it in the output.

The algorithm above has a time complexity of O(nk log k). This is because in the third step of the algorithm, it will run $n*k$ times. In every iteration of the loop, we call heapify which takes O(log$k$) time. Putting this together we get O(nk log$k$).

Question 5)

We know that a full binary tree has exactly $2n$-1 nodes. The following pseudo-code describes how to represent any optical prefix code using only $2n$-1+n⌈lg n⌉ bits.

a.) Encode the full binary tree structure by performing a pre-order traversal of T
b.) FOR EACH node recorded in the traversal {
        IF node is an internal node THEN write 0
        ELSE (node is a leaf) write 1

    }

Since we know the binary tree is full its structure is now uniquely determined. As a property of this tree, we can encode any character of C in ⌈lg n⌉ bits. Since there are n characters, we may encode them in pre-order traversal order using n ⌈lg n⌉ bits.

Question 6)

We will use a proof by strong induction to prove that every node has rank at most ⌊lg n⌋. In the base case where n = 1, the nodes rank is 0 = ⌊lg 1⌋. Let's suppose that the claim holds for 1 to n nodes. Given n+1 nodes, we may perform a UNION operation on two different sets with x and y nodes respectively where $x, y \leq n$. With this we know that the root of the set containing x nodes has a rank of at most ⌊lg x⌋, and that the root of the set containing y nodes has a rank of at most ⌊lg y⌋. If the ranks of sets x and y are unequal, then the UNION operation will result in a set that has the same rank as the highest of the

Christopher Tam
250803892

two individual sets. If the ranks are indeed equal then the rank of the UNION operation increases the rank by 1, and the resulting set has a rank of $\lfloor \lg x \rfloor + 1 \leq \lfloor \lg (n+1)/2 \rfloor + 1 = \lfloor \lg (n+1) \rfloor$.

Question 7)

We may determine the number of bits necessary to store *x.rank* for each node *x* given that their value will be at most $\lfloor \lg n \rfloor$. We may represent *x.rank* using Θ(lg(lg(n))) bits and may need to use Θ(lg(lg(n))) many bits to represent a number that can take that many values.