Assignment 2

CS3340b – Analysis of Algorithms - Matteo Tanzi – 251011979

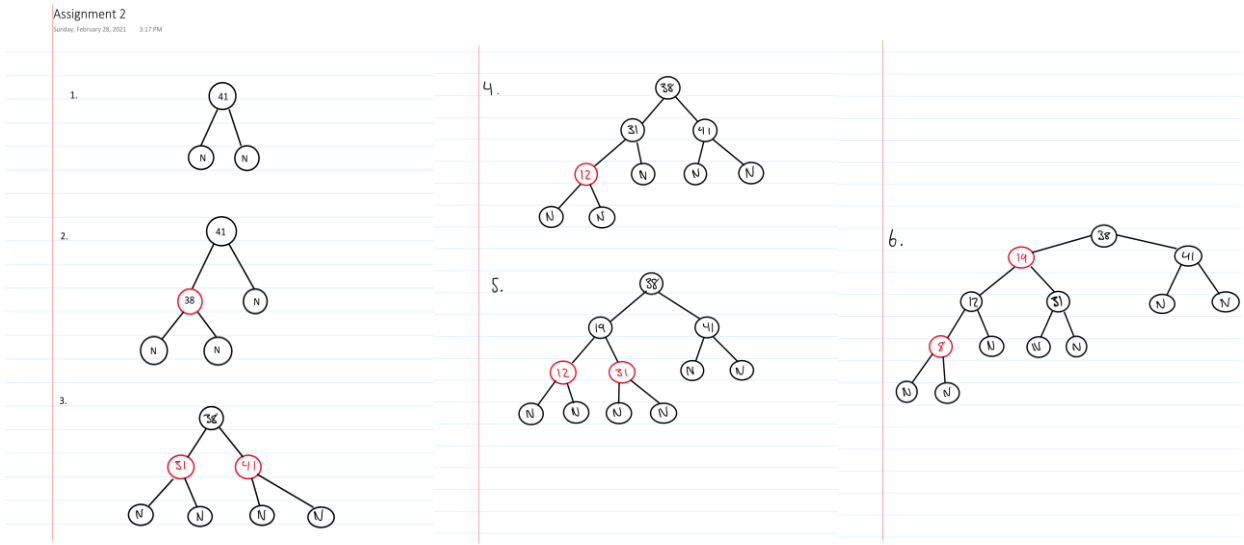
Question 1:

Using Figure 8.2 as a model, illustrate the operation of COUNTING-SORT on the array A = (6, 0, 2, 0, 1, 3, 4, 6, 1, 3, 2)

Step #:	Array Values	Notes:
1	C = (0,0,0,0,0,0,0) A = (6,0,2,0,1,3,4,6,1,3,2) B = (n,n,n,n,n,n,n,n,n,n,n)	After line 1 (text code) n = null
2	C = (1,2,2,2,1,0,6) A = (6,0,2,0,1,3,4,6,1,3,2) B = (n,n,n,n,n,n,n,n,n,n,n)	After line 5 (text code)
3	C = (1,4,6,8,9,9,11) A = (6,0,2,0,1,3,4,6,1,3,2) B = (n,n,n,n,n,n,n,n,n,n,n)	After line 8 (text code)
4	C = (1,4,5,8,9,9,11) A = (6,0,2,0,1,3,4,6,1,3,2) B = (n,n,n,n,n,2,n,n,n,n,n)	After 1 repetition of line 12 loop
5	C = (1,4,5,7,9,9,11) A = (6,0,2,0,1,3,4,6,1,3,2) B = (n,n,n,n,n,2,n,3,n,n,n)	After 2 repetitions of line 12 loop
6-14	C = (-1,2,4,6,8,9,9) A = (6,0,2,0,1,3,4,6,1,3,2) B = (0,0,1,1,2,2,3,3,4,6,6)	After 8 more iterations of final loop

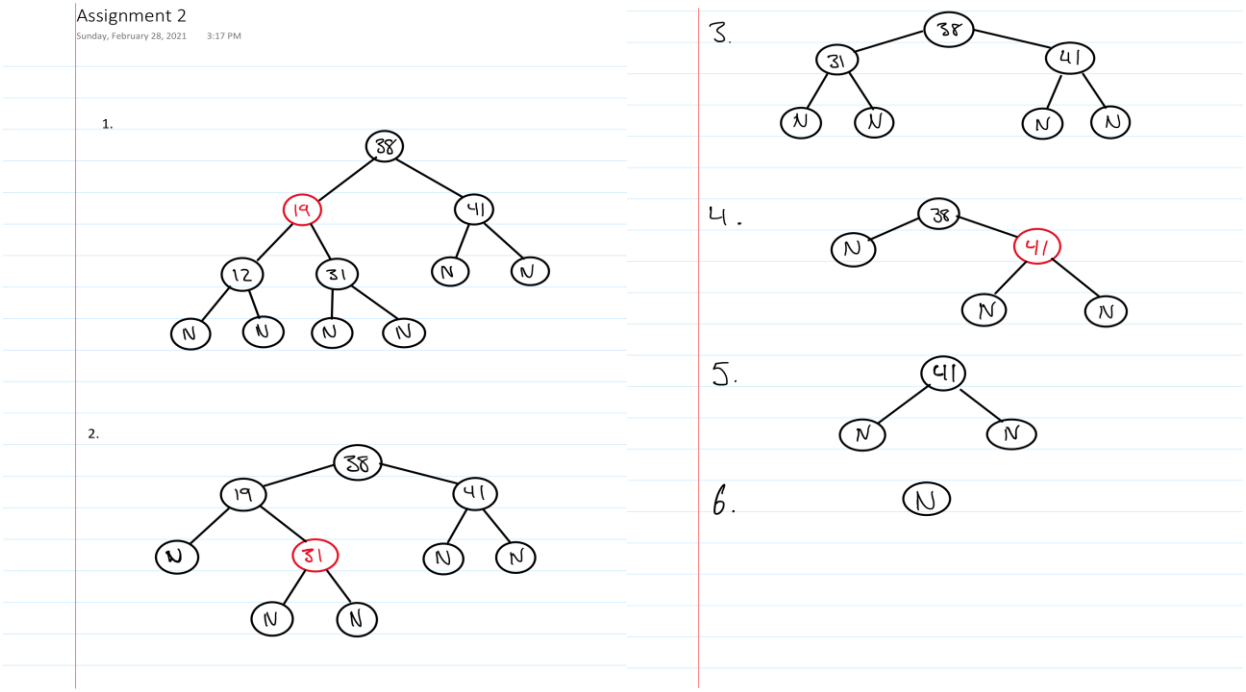
Question 2:

Show the red-black trees that result after successively inserting the keys 41, 38, 31, 12, 19, 8 into an initially empty red-black tree.



Question 3:

In Exercise 13.3-2, you found the red-black tree that results from successively inserting the keys 41, 38, 31, 12, 19, 8 into an initially empty tree. Now show the red-black trees that result from the successive deletion of the keys in the order 8, 12, 19, 31, 38, 41.



Question 4:

Given n elements and an integer k. Design an algorithm to output a sorted sequence of smallest k elements with time complexity O(n) when *k*log(n) ≤ n.

1. Describe your algorithm in English (not with pseudo code)

Build a min heap, this can be accomplished by creating a binary tree where an internal node is smaller to or equal to the values of that nodes children. This data structure uses the operations getMin(), extractMin() and insert() and heapify(). getMin() returns the root, extractMin() returns the min, removes it and calls heapify. Insert() takes a new key and inserts it into the tree. Heapify() ensures the structure of the binary tree follows the initial definition. After constructing the min heap with n elements, call extractMin() k times, to extract a sorted sequence of the smallest k elements.

2. Show why the algorithm is correct

The algorithm is correct because it denotes that the data structure defines the placement of values inside the tree. Since each update via insert() will use heapify to control the placement of values, we can ensure that k operations of extract min will result in the smallest k elements in a sorted sequence

3. Analyse the complexity of the algorithm

The time complexity of this algorithm can be defined by the time complexity of the methods.

getMin() – O(1)

extractMin() – O(1)

Insert() – O(n)

Heapify() - (klog(n))

Therefore, the time complexity is O(n + klog(n)) since klog(n) <= n, it runs in O(n) by reduction

Question 5:

Design an efficient data structure using (modified) red-black trees that supports the following operations:

Insert(x): insert the key x into the data structure if it is not already there.

1. Describe your algorithm in English (not with pseudo code)

Perform a binary tree insertion and colour the new node red if the node is not the root.

Given scenario's we will outline how to maintain the red black tree properties by case:

Case 1: if the uncle of the inserted node is red

Colour p(z) and y to black, and p(p(z)) to red. If p(p(z)) is the root node, change to black.

Case 2 (y is black):

2A (y is the right child of p(p(z)) & z is the right child of p(z):

- Perform a left rotation at p(z)

2B (y is the left child of p(p(z)) & z is the left child of p(z)):

- Perform a right rotation at p(Z)
- Transform case a,b to c,d

2C (y is the right child of p(p(z)) & z is the left child of p(z):

- Colour p(z) to black and p(p(z)) to red with a right rotation at p(p(z))

2D (y is the left child of p(p(z)) & z is the right child of p(z)):

- Colour p(z) to black and p(p(z)) to red with a left rotation at p(p(z))

2. Show why the algorithm is correct

The algorithm is correct because for each case of insertion, the properties of a red-black tree are upheld. Since the properties of the red-black tree are upheld and the node is inserted, the algorithm is correct.

Delete(x): delete the key x from the data structure if it is there.

1. Describe your algorithm in English (not with pseudo code)

Use binary search tree deletion to determine what nodes need to be deleted.

Let some node y that is to be deleted have either left(y) or right(y) or both as leaves. If y is red, then simply delete it

If y is black and either the left(y) or right(y) is an internal node, then colour y's internal child to black and delete y.

If y is black and both its children are external nodes, then let y = left(P(y)), delete y and make the new node double black. Let w = right (p(x)) where x is the new node

We will define some cases:

Case 1 (W is red): exchange the colours for w and p(w), left rotate p(w), this case now becomes case 2,3 and/or 4

Case 2 (W is black, right(w) is black and left (w) is black): colour w to red and move the extra black to p(x)

Case 3 (W is black, right(w) is black and left(w) is red): exchange the colours of w and left(w), right rotation at w, transform to case 4

Case 4 (W is black, right(w) is red): exchange the colours for w and p(w), colours right(w) to black and a left rotation at p(w)

2. Show why the algorithm is correct

The algorithm is correct because for each value deleted from the red-black tree, the properties of the red-black tree are preserved, and the value is deleted from the tree. Since these are the requirements for the delete(x) algorithm, it is correct

Find_Smallest(k): find the kth smallest key in the data structure.

1. **Describe your algorithm in English (not with pseudo code)**

To find the kth smallest in the data structure, an inorder traversal of the tree will be conducted by recursing down the left tree, saving the value in an array from [0-n], then recursing down the right tree. To print the kth smallest value, print the value of the array at position k-1.

2. **Show why the algorithm is correct**

Since red black tree's use binary search insertion and deletion, they uphold the property that from left to right, the nodes are inorder, therefore, saving each value of an inorder traversal and printing the k-1 position will yield the kth smallest value in the tree.

What are the time complexities of these operations?

The time complexity of insert and delete is O(log n)

The time complexity for find_smallest(K) is O(n)

Question 6:

Prove that every node in a disjoint set has rank at most [lg n].

Base Case:

n = 1, therefore the rank = 0 = log 1 (upholds)

Induction Hypothesis:

For nodes (0,1,2… n) [natural numbers], a node in a disjoint set has at most rank [log n]

Inductive Step:

To prove, we must analyse the ranks of two disjoint sets, x and y, after a union operation, where both x < n and y < n and the rank of x = rank of y

Due to inductive hypothesis, we know that the rank of x is at most [log x] and the rank of y is at most [log y]. Since the ranks are equal, the resulting set has a rank of [log x] + 1, since at most the rank of x is (n+1)/2 we can state that [log x] + 1 <= [log(n+1)/2] +1. Therefore, a node in the disjoint set has at most rank log(n+1)

Question 7:

A) In light of Exercise 21.4-2, how many bits are necessary to store x.*rank* for each node x?

The max number stored in k bits = 2^(k-1). Therefore, there number of bits required to store a number N is log(n). since the max value of N = log n. The maximum number of bits required is log (log (n))

B) is it enough to use one byte to store rank for q10? Explain your answer.

The number of bits required to store a character n is log n. Therefore, if the value of log n for the character library rank of q10 is less than 8, this will be possible. If not, it will not be possible.

Therefore, when the value of rank is less than 256, one byte can store the rank.

Question 8:

Prove that if we order the characters in an alphabet so that their frequencies are monotonically decreasing, then there exists an optimal code whose codeword lengths are monotonically increasing.

To prove this, we will use contradiction.

Based on the definition of an optimal codeword in some alphabet, we know that the frequency of C(n) >= C(n-1) and the depth of C(n) <= C(n-1)

If there are some two words, x and y, such that x has both a longer codeword (monotonically increasing) and also has a higher frequency (monotonically increasing).

Lets state that x is the most frequent and, has the greatest depth in the alphabet (codeword is monotonically increasing), and lets state that y is the second most frequent. If we were to state that y had the second greatest depth (monotonically increasing), we would imply that the dT(n-2) <= dT(n-1). Which would be false by contradiction.

Question 9:

Suppose we have an optimal prefix code on a set C ={0,1,…,n-1} of characters and we wish to transmit this code using as few bits as possible. Show how to represent any optimal prefix code on C using only 2n-1+n [lg n] bits. (*Hint:* Use 2n-1 bits to specify the structure of the tree, as discovered by a walk of the tree.)

Knowing that the optimal prefix code requirement is a full binary tree, we can specify that the number of nodes in a full binary tree is 2n-1.

To encode the structure of the binary tree, define the value of a leaf node to 1 and the value of an internal node to 0.

Perform a pre-order traversal of the tree, using log n as the amount of bits requires to encode a character library of size n, we can state that it will require n [log n] bits to encode the character set C, this can be demonstrated by encoding them in the order of the pre-order traversal.

Q10)

Final_set():

1. **Describe your algorithm in English (not with pseudo code)**
   - Given some Disjoint-Set, which has two defined arrays, parents[] and rank[], finalSets will operate as follows:
   - For all values of n for which exist in the array parent[size n], ensure that each value of n is included in a set.
   - If the parent value at n is n, then set the value of the rank to 1 and increment the total counter of current sets, this process resets the representatives such that the integers from 1 to final_sets() will be used as representatives
   - If the rank of I is 0 and the parent at I is > 0 then the parent of I is p(p(i))
   - Define some value of completion of final set to 1 and return the current total number of sets
   - Assume that methods, make set and union sets cannot operate when finals_sets is set to 1
2. **Show why the algorithm is correct**
   - The algorithm is correct because it uses complete method findSet() to reset the representatives of the sets, counters for each set that is reset and then uses a true/false value to finalize the methods makeSet() and union_sets(). Since all the requirements for the algorithm are complete, the algorithm is correct.
3. **Analyse the complexity of the algorithm.**
   - Since the algorithm searches through each value of an n sized array, and compares them, the time complexity will be O(n)

10b)

1. Define a 2d matrix array to the size and length of the input file. (71x71) Read in each line in the input file, for each + add a 1 to some 2d array that defines the image. Print all the values in the array.
2. Create the disjoint-set where the size is equal to the amount of values held in the original matrix. Make a set for each component, and if there is an existing component either behind or above it, union the sets. Finalize the sets, and define an array which has the size of the final amount of sets. Print the values of each component using ascii values and their associated set, if the set doesn't exist print a space.
3. Define some array char and some array setSorted, where char[ascii value] stores the ascii value of the associated char and set[ascii value] stores the amount of times the ascii value was used. Print the values in sorted order.
4. Use the values in sets from p3 to ensure that if the ascii is used less than 2 times or the ascii is a space, then a space is printed.