CSC 225 Fall 2016
Allan Liu
V00806981

Assignment 3 – Theoretical part

Question 1.

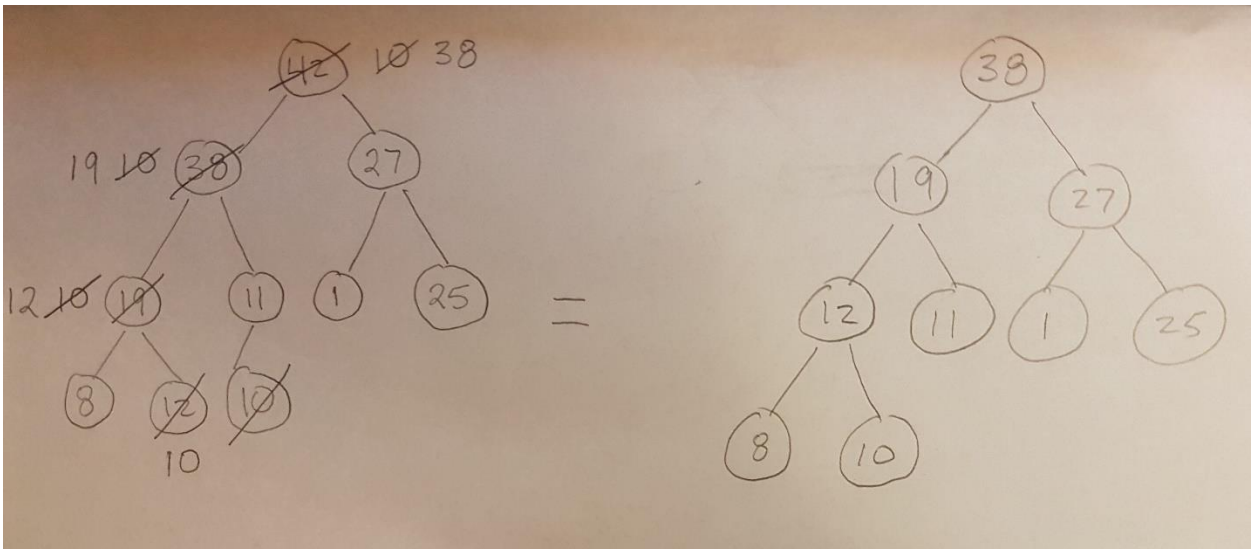
Input: Nuts and bolts in an array of characters
Output: Return an array of matched nuts and bolts
Algorithm NutsBolts (n, b, low, right)

```
    if (left < right)
        pivot ← partition (n, b[right], left, right);
        partition (b, left, right, nuts[pivot]);
        NutsBolts (n, b, left, pivot-1);
        NutsBolts (n, b, pivot+1, right);
    return
end
```

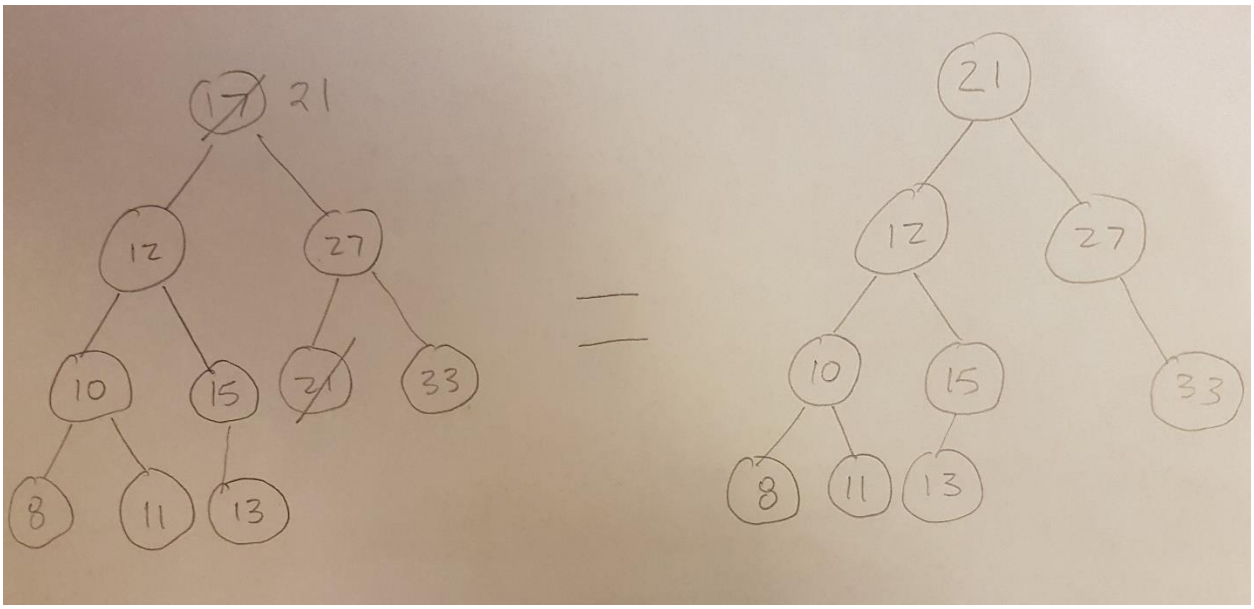
Divide and conquer method using quicksort with *expected* O(n log n) time.

Question 2.

a) Delete the root node (42) and replace with the last element (10). Re-establish order property by bubbling down the element (10).



b) Delete the element with the key (17) and replace it with the next element larger than it (21) by finding the smallest element in the right sub-tree.



Question 3.

1) Since a Treap is defined as a binary search tree, it must satisfy the property of having a search key as well as a priority for each node, similarly like a heap. When inserting nodes in an initially empty tree, we first follow the insertion algorithm for a binary search tree. For every key being inserted into the tree, it first compares the key with the root, if the key is less than the root then it will iterate to the next key to the left of the tree and compares itself with the following key[u]. Similarly, it will iterate to the next key to the right if it is larger than the key[u], this process will continue down the tree until it reaches a leaf. Next it needs to satisfy the priority property by rotating right if the new node is inserted in the left subtree and rotate left if it is inserted in the right subtree.

2)

- **Create new node with key equal to x if root is null**
- **Perform standard Binary Search Tree Insert**
- **Most recent inserted node will violate Min-Heap order property**
- **Use rotations to restore Min-Heap property**

Input: Inserting a desired key
Output: Treap where every node has a key and priority
Algorithm InsertTreap (k, e)

```
if (isEmpty())
        return newNode(k)
if (key <= T.root())
        left ← InsertTreap(k, left)
        if (left > T.root())
                root ← rotateRight(T.root())
else
        right ← InsertTreap(k, right)
        if (right > T.root())
                root ← rotateLeft(T.root());
return root
```

3)
The height of a Treap with n elements in the worst-case is O(n). In a normal BST insertion algorithm, insert always places the new node at a new leaf of the tree where the height of a regular BST depends on the root. With a similar concept like quicksort, if the root is closer to the median of the sequence of values the better the height of the Treap. In addition, the priority property also helps pick a good root for every subtree. Therefore, the worst case height of a Treap will only be obtained if for each subtree has a root node that is near the outer edge of the sequence of values and isn't affected by the priority.

Question 4.

Input: An integer number
Output: Return a boolean value true or false to determine whether it is a Halloween number or not
Algorithm isHalloween (number)

```
HashT ← new Hash Table
if (number <= 0)
        Return false;
while (true)
        while (number > 0)
                digit ← (number % 10);
                number ← (number/10);
                sum += digit * digit;
        if (sum = 1)
                return true;
        else if (sum is already in HashT)
                return false;
        else
                push sum into HashT;
```

Split the number into single digits and multiply each digit to itself, then add the result to a sum. If the number is not a Halloween number, then the sum will eventually be repeated in the hash table. When the result is 1 then it is a Halloween number.