Mamoutou Sangare

Student Number: V 00010536

CSC 225

Assignment 3

Question 1

Algorithm MatchBoltNut(nuts[],bolts[],n)

Input: n bolts (put in a array) of different widths and n corresponding nuts (put another array)

Output: match each bolt to its nut

```
 low ← 0
high ← n -1
if low < high  then
      // pick last element in the bolts array for nuts partition
    pivot ← partition( nuts, low, high, bolts[high])
     //  partition of bolts using the partition of nuts
    partition (bolts, low, high, nuts[pivot])
    //  recursion on [low … pivot-1] and [pivot+1 …high]
    MatchBoltNut (nuts, bolts, low, pivot-1)
     MatchBoltNut (nuts, bolts, pivot+1, high)
```

Question2

a) The new heap after deleting the maximum element

```
            38
           /   \
         19     27
        / \    /  \
      12  11  1    25
     / \
    8   10
```

Remove 42 and move 10 to the root. Then the heap meets the heap shape property, so we have to bubble down 10 until it becomes smallest than its parent so that it can meet the order property.

b)

Binary search Tree after removing key 17

```
          21
         / \
      12     27
     / \       \
   10   15      33
  / \   /
 8   11 13
```

Remove 17 and move 21 to the root and check to see if the tree meets the binary search propriety. It does meet the propriety since 21>12 and 21<27 so we do not need to bubble down anything in the case.

Question 3

3-1

A treap is defined as a binary tree in which every node has both a search key and a priority where:

a- The inorder-sequence of search keys is sorted. That is, if node v is a left child of u, then key[v] < key[u]; if v is a right child of u, then key[v] > key[u] (a property of binary search tree),

b- Each node's priority is smaller than the priorities of its children. That is, if v is a child of u, then priority(v) > priority(u) (a property of min-heap). Thus, b) makes a treap a more balance binary tree. Inserting a node v (k,e) in a treap where k is the key and e is the priority, the property of binary search tree requires the treap to have the inorder-sequence of search keys sorted. That is, if node v is a left child of u, then key[v] < key[u]; if v is a right child of u, then key[v] > key[u] which is exactly the definition of binary search tree. Therefore a treap is exactly a binary search tree.

3-2 Since a treap is both a binary search tree with respect to the node, and a heap with respect to node priorities, we first insert the node in a treap by doing a normal binary search tree insertion. Then bubble the node upward in the tree by rotating it with its parent until its value is smaller than its parent.

```
Algorithm Insert(T.root(),v(k,e))
Input: root of the tree and the node v(k,e)where k is the key and e is
    priority in the tree
Output: insert v(k,e)in the tree

   if (T.root() == null) then  // insertion position found
     t.root()← v(k,e)
              return
   if (k.v(k,e) <= T.root().key) then // proceed to the left branch
     T.root().left ← Insert(T.root().left, v(k,e));
     if (e.v(k,e)>e.parent.v(k,e)) then   // right rotation if needed
        rightRotation(T.root().left,T.root().left.parent()
   else // k > T.root().key, i.e. proceed to the right branch
     T.root().right ← Insert(T.root().right, v(k,e))
              if (e.v(k,e)<e.parent.v(k,e)) then
        leftRotation(T.root().right,T.root().right.parent()
          return
```

3-3 The height of the treap in the worst case is n the number of elements inserted. Suppose we insert in an empty treap the following four (50, 73), (60, 50), (70, 45), (80, 44). The element (a,b) where a is the key and b is the propriety (max heap) then the treap will be:

```
(50,73)
      \
     (60,50)
          \
         (70,45)
              \
             (80,44)
```

So we insert 4 elements and the height of the treap is also 4.

4- Pseudocode using the concept of hashing to describe an efficient algorithm to determine whether or not a number is a Halloween number.

```
Algorithm isHaNumber(n)
  Input: a positive integer n
  Output : true/false to show whether or not a number is a Halloween number
  HashT  ← new Hash Table
  Find ← true
For i← 0 to number of digits of the number (n the first time)
 // take each digit of the number and put in a set S
 S ← {k1,k2,k3 ...} where k1,k2,k3,... are the digits of the number (n the first time)
 //sum of the square of all the element in S and store it in S2
S2 ← $k1^2$+ $k2^2$+ $k3^2$ + ...
   If (S2 is in HashT)    // the number will repeat itself if it is not a Halloween number
      Find = false
     break
   Else
    Push S2 to HashT
   endFor
   return find
```