PROBLEM SET 3

Zhiqiang Xie 77892769

Problem 1

- Find the median of an array:
  - Divide n elements into groups of 5
  - Find the median of each group
  - Recuisively, find the median x of the $\lfloor n/5 \rfloor$ medians
  - Partition the n elements around x and drop most of the smallest and largest.
  - The time complexity of finding the median is:
    - $T(n) = T(\lceil n/5 \rceil) + T(7n/10 + 6) + O(n)$, which is guaranteed $T(n) \in O(n)$
- Since all we want is to use as less bags as possible, as well more chocolates. In other word, we want those heaviest chocolates. Therefore, we have this divide procedure:

  Python

```python
def buy(choco, target):
    if sum(choco) < target:
        return False # it's impossible to achieve the goal
    plan = []
    def divide(sub, target):
        median = get_median(sub) # This function is illustrated above
        heavy, light = [], []
        for w in sub:
            if w >= median:
                heavy.append(x)
            else:
                light.append(x)
        sum_heavy = sum(heavy)
        if 0 <= sum_heavy - target < median: # Reach the best deal
            return heavy
        elif sum_heavy - target < 0:
            target -= sum_heavy
            plan += heavy
            return divide(light, target)
        else:
            return divide(heavy, target)
    return plan += divide(choco,target)
```

  The time complexity is $T(n) = T(n/2) + O(n) \in O(n)$

Problem 2: Binary Search Tree Practice

1. Firstly, let's specify some functions:
   - find_min: go down to the leftmost node of the BST, this node has the smallest value in the BST or sub-BST.
   - find_next: to find the successor that is only larger than the node in the BST or sub-BST:
     - If the node has a right child, then the leftmost node of the right sub-BST is the successor.
     - If the node has no right child, then:
       - If the node is its parent's left child, then the succesor is its parent.
       - If the node is its parent's right child, then we keep following the pointers to parents until an ancestor has a larger value. Once we reached the root node, terminate the traversal.

   Thus, from this naive procedure above, we shall see that we'll visit any node in the BST at most 3 times during the in-order traversal. Therefore, the time complexity $T(n) = n * \Theta(1) = O(n)$

2.

   1. **Python**

   ```
   def isBST(self, root):
       """
       :type root: TreeNode
       :rtype: bool
       """
       self.inorder = []
       self.traverse(root)
       return all(self.inorder[i] <= self.inorder[i+1] for i in
   range(len(self.inorder)-1))

   def traverse(self, root):
       if not root:
           return
       else:
           self.traverse(root.left)
           self.inorder.append(root.val)
           self.traverse(root.right)
   ```

   The time complexity $T(n) = O(n) + O(n) \in O(n)$

2. **Python**

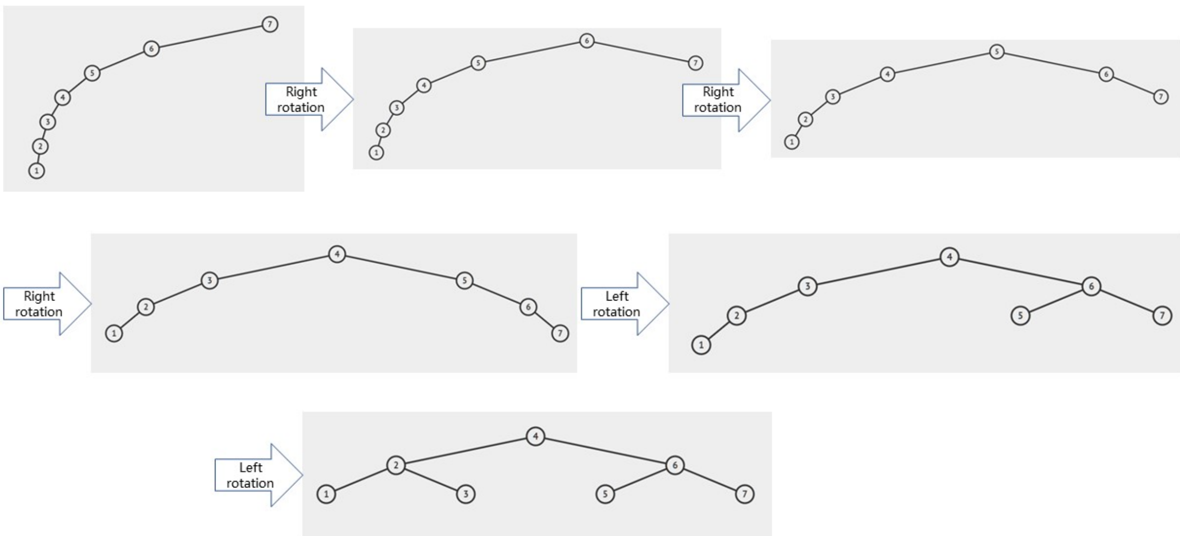
```python
def isBalanced(self, root):

    def check(root):
        if root is None:
            return 0
        left  = check(root.left)
        right = check(root.right)
        if left == -1 or right == -1 or abs(left - right) > 1:
            return -1
        return 1 + max(left, right)

    return check(root) != -1
```

The time complexity $T(n) = n * O(1) \in O(n)$

3.



Problem 3: Strongly 2-Universal Hashing

1. If H is strongly 2-universal, then for every pair of keys (x, y) where $x \neq y$ and for every $i \in \{0, 1, \ldots, m-1\}$, we have:
    - $\Pr[(h(x), h(y)) = (i, i)] = \frac{1}{m^2}$, for every $h \in H$

    Therefore, here are exactly m possible ways to make two keys x, y collide:

    - $h(x) = h(y) = i$, for all $i \in \{0, 1, \ldots, m-1\}$ and all $h \in H$
    - Thus, $\Pr[h(x) = h(y)] = \sum_{i=0}^{m-1} \Pr[(h(x), h(y)) = (i, i)] = \frac{m}{m^2} = \frac{1}{m}$, for every $h \in H$

    Thus, H is universal by definition.

2. A simple eaxmple with $m = |H| = |U| = 2$:

	x	y
h_1	0	0
h_2	1	0

It's easy to see the probability that two keys x, y collide is $1/2$ since only h_1 can cause the collision.

However, if H is a strongly 2-universal hash family, for a randomly choosen hash function h, all the possible pairs of $(h(x), h(y))$ must be equally likely.

- In this case, they're $(0,0),(0,1),(1,0),(1,1)$.

But $(h(x), h(y))$ never equals to $(0,1),(1,1)$ in our H.

Therefore, H is not strongly 2-universal.

3. Firstly, I assume H is a universal hash family. Thus we have $\Pr[h(x) = h(y)] = \frac{1}{m}$, for every $h \in H$
   - I denote the index of $h \in H$ by n, where $n >= 0$, therefore $H = \{h_0, h_1, \ldots, h_n\}$
   - Now we set a special x_s as a probe:
     - For every $h_i \in H$, we get $o = h_i(x_s)$. If $o \neq i$, we set $h_i(x_s) = i \bmod m$, and then we traverse the U to find a key y which satisfies that $h_i(y) = i$. Once we get it, we set $h_i(y) = o$, otherwise, just over.
   - Now we have a new hash family H, where we can index the target hash functions by x_s:
     - $h_i(x_s) = i \bmod m$, thus only $\lceil n/m \rceil$ hash functions are candidates. We can guarantee that the probability to make a collision to be greater than $\frac{1}{m}$ since we have the information of the target functions.
     - For all $h \in H$, $\Pr[h(x) = h(y)] = \frac{1}{m}$ for any pair of $(x, y), (x \in U, y \in U, x \neq y)$ is still satisfied. Since I don't change the number of pairs of collision in the output domian of any hash function $h_i \in H$.

   Therefore, the new H maintains its universality via the transformation and we guarantee the collide probability to be greater that $\frac{1}{m}$ as well.

4. Konwing $h(x)$ gives the adversary no information about $h(y)$ if h is randomly choosed from a strongly 2-universal hash family H. We can format this problem like this:
   - $X = h(x_s)$, where the X is our information towards the hash function h.
   - $\Pr[h(x) = h(y) | h(x) = X] = \frac{\Pr[h(x)=h(y), h(x)=X]}{Pr[h(x)=X]} = \frac{1/m^2}{1/m} = \frac{1}{m}$

   Therefore, no matter which x_s the adversary choose and which $X = h(x_s)$ the adversary knows, the probability of any particular y colliding with x is still $1/m$.