Sample Exam 2

---

(1) Create an algorithm such that given a string which consists of lowercase or uppercase letters, find the length of the longest palindromes that can be built with those letters. Do so in O(n) time.
This is case sensitive, for example "Aa" is not considered a palindrome here.

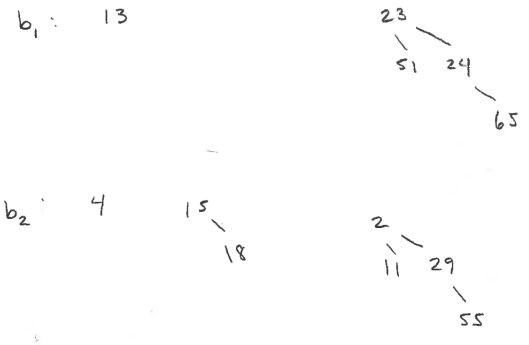
> Example:
> Input: "abcccdd"
> Output: 7
> Explanation: One longest palindrome that can be built is "dccaccd", whose length is 7.

(2) Create an algorithm such that given a non-empty list of words, return the k most frequent.

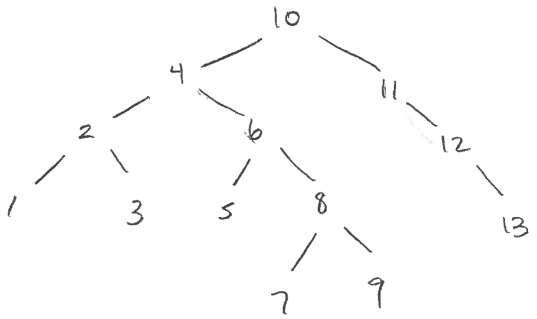
(3) Give the following red-black tree after inserting [20, 51, 1, 10, 23, 15, 17, 27].

(4) Using the hash f(x) = x mod 12 and a hash table of size 12. Show the resulting hash table of inserting [12, 2, 10, 48, 20, 36] using quadratic probing as a collision solution.

(5) Merge the following two binomial queues b1 and b2:



(6) Given the following splay tree, s1, show the resulting splay tree after accessing 3.



(7) Form a recurrence relation for the minimum number of full nodes F(h) in a AVL tree. *A node is full if it has exactly two children*

(8) Find the sqrt(x) using a binary search.

Solutions

(1) For each letter it occurs v times. We know that v // 2 * 2 of those characters can be paired on each side of the string. For example if we have "bbb" we know that "bb" can be paired.

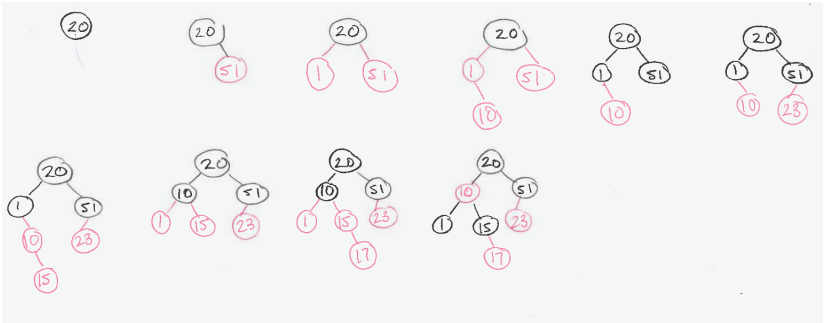
Now we need to check for characters with an odd length (v % 2 == 1) meaning that the palindrome must have a unique center. To see if we can accompany having a unique center we have to see if the palindrome is currently of even length and if so we can otherwise if the palindrome is already odd adding this unique center would no longer make it a palindrome.

```
def longest_palindrome(self, s):
    ans = 0
    # create a hash table to maintain the count of each key
    table = {}
    for c in s:
        # check if the key already exist
        if c not in table:
            table[c] = 0
        table[c] += 1

    # .values() returns all the values of the hash table
    for v in table.values():
        ans += v / 2 * 2
        if ans % 2 == 0 and v % 2 == 1:
            ans += 1
    return ans
```

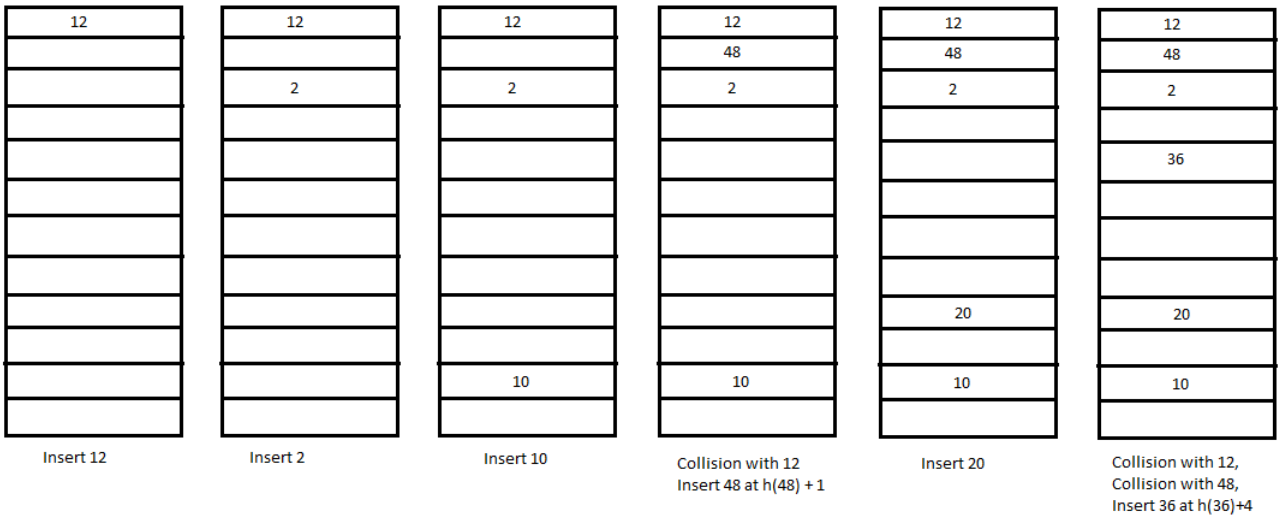
(2) We can easily achieve this by getting the counts for every word then creating a min heap with the numbers as negatives. Python has it's own heap structure called heapq which I will be using

```
import heapq
def top_k_words(words, k):
    # Create a hash table with each word having zero count
    word_count = {k: 0 for k in set(words)}

    # Remember this is a min heap so the biggest negative number is the smallest
    for word in words:
        word_count[word] -= 1

    # build the heap
    # first we convert the hash_table into a list of pairs since
    # this is because of how heapq takes input
    heap = [(-v,k) for k,v in word_count.items()]
    heapq.heapify(heap)

    # return the top k words
    top_words = []
    for i in range(k):
        top_words.append(heap[i][1])
```

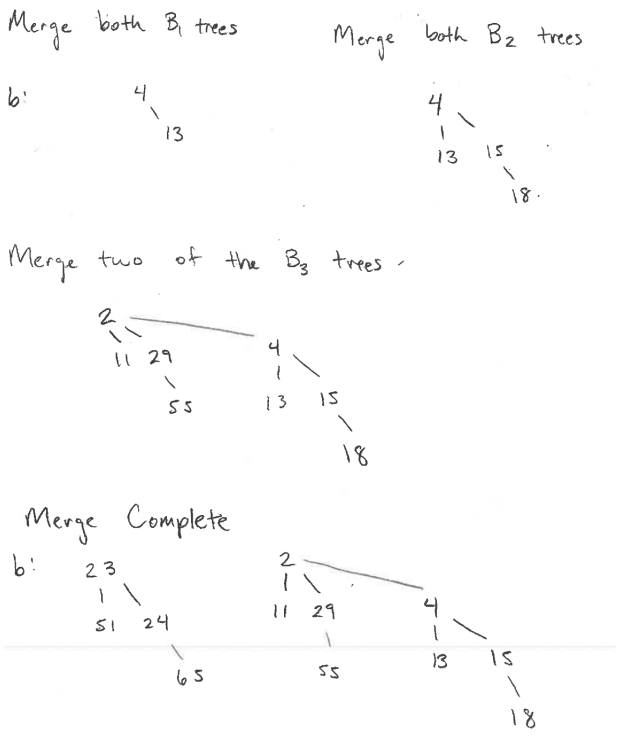
(3)



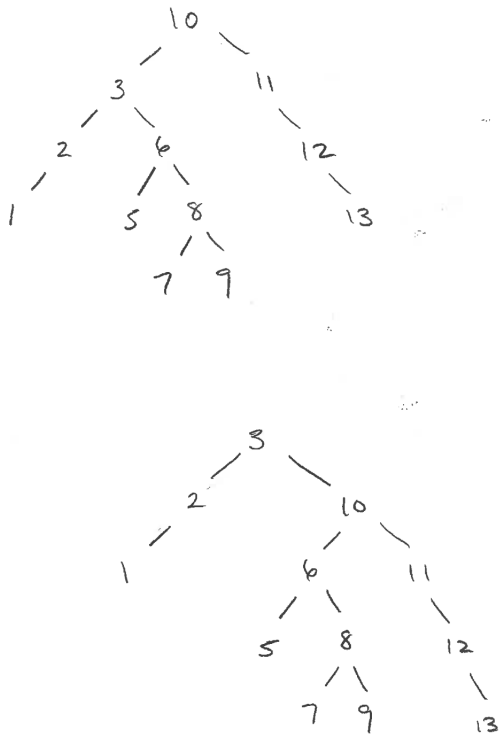
(4)


12
Insert 12
12
2
Insert 2
12
2
10
Insert 10
12
48
2
10
Collision with 12
Insert 48 at h(48) + 1
12
48
2
20
10
Insert 20
12
48
2
36
20
10
Collision with 12,
Collision with 48,
Insert 36 at h(36)+4

(5)


Merge both B_1 trees
Merge both B_2 trees
b_1:
4
13
4
13
15
18
Merge two of the B_3 trees.
2
11
29
55
4
13
15
18
Merge Complete
b_1:
23
51
24
65
2
11
29
55
4
13
15
18

(6)


10
3
11
2
6
12
1
5
8
13
7
9
3
2
10
1
6
11
5
8
12
7
9
13

(7)

$$F(0) = 0,\ F(1) = 0,\ F(2) = 1,\ F(3) = 2$$
$$F(h) = F(h\text{-}1) + F(h\text{-}2) + 1$$

(8) We can perform this by creating a list from 0 - x. Then perform a binary search on that list. This allows us to find the sqrt(x) in O(logN) time.

```
def sqrt(x):
    # create list from 0 - x
    bst = list(range(0,x+1))
    answer = 0
    while len(bst) > 0:
        mid = len(bst) // 2
        mid_v = bst[mid]

        # x is a perfect square
        if mid_v*mid_v == x:
            return mid

        # Since mid_v*mid_v is smaller it could be the ans so we update
        # Then we move x closer to sqrt(x) until it is larger
        elif mid_v*mid_v < x:
            answer = mid_v
            bst = bst[mid:]

        # else mid_v*mid_v is larger than x and we need to find a smaller number
        else:
            bst = bst[:mid]

    return answer
```