Homework 3 Solutions

Use a table size of 13 for the following hash questions 1-4 and
A given input of {3823, 8806, 8783, 2850, 3593, 8479, 1941, 4290, 8818, 7413}
with a hash function h(x) = x mod 13.

---

(1) **(10 pts)** Show the resulting separate chaining table

[4290]
[3823]
[2850,8479,7413]
[1941,8818]
[8806,3593]
[8783]

(2) **(10 pts)** Show the resulting table using linear probing

4290
3823
2850
8479
8806
3593
1941
8783
8818
7413

(3) **(10 pts)** Show the resulting table using quadratic probing

4290
3823
2850
8479
8806
3593
8818
8783
1941
7413

(4) **(10 pts)** Show the resulting table using double hashing with $h_2(x) = 11$ - (x mod 11).

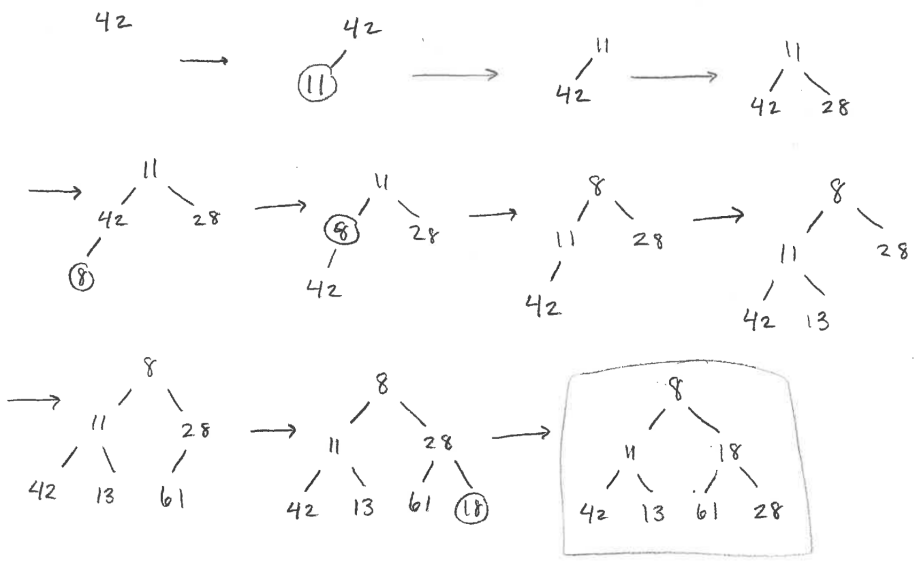
4290
3823
2850
1941
8806
7413
8479
8783
3593
8818

(5) **(20 pts)** Given an array of strings, group anagrams together in O(nlogn) time.
Example
Input: [”eat”, ”tea”, ”tan”, ”ate”, ”nat”, ”bat”],
Output:[[”ate”,”eat”,”tea”],[”nat”,”tan”],[”bat”]]

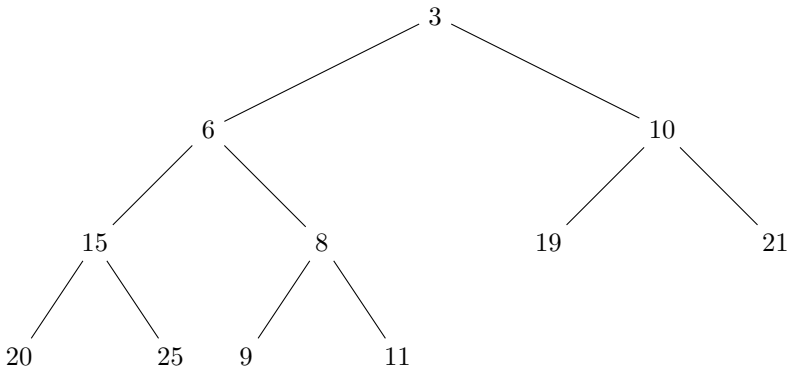
For this the intuition is to use the sum of the ASCII values as the key. However this will not work. We know that since there is no procedence in addition x+y+z=z+y+x however two different combinations of numbers could also equal each other: x+y+z=a+b which would group words that are not anagrams together. So there are two possible ways of doing this and I will present both. The first is to sort the string and use it as the key as every anagram will have the same sorted key. The second is to use an array of [1-26] which keeps the counts of which alphabets are used as the key.

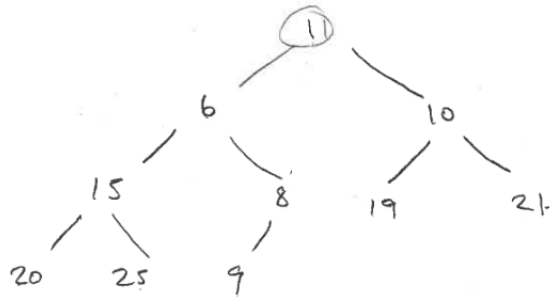
```python
# Method 1 using sorting
def anagrams_sorted(arr):
    hash_table = {}
    for w in arr:
        # we will use the sorting algorithm already implemented in python
        # however not just any sorting algorithm would work.
        # It needs to sort in linear time meaning using bucket sort / radix sort
        # sorted returns a list and I would like to keep my keys a string
        # so ''.join() concatenates the list into a string
        key = ''.join(sorted(w))

        # Check if the key exist and if not add it
        if key not in hash_table:
            hash_table[key] = []

        # add the word to the list of anagrams
        hash_table[key].append(w)

    # return the values of the hash_table
    return list(hash_table.values())

# Method 2 using counting
import string
def anagrams_count(arr):
    hash_table = {}
    for w in arr:
        count_arr = [0] * 26
        for c in w:
            # string.lowercase.index(c) returns which index the alphabet is
            # i.e. a = 0, b = 1, c = 2, ..., z = 25
            count_arr[string.lowercase.index(c)] += 1
        # convert the count array to a string
        key = ''.join(count_arr)

        # Check if the key exist and if not add it
        if key not in hash_table:
            hash_table[key] = []

        # add the word to the list of anagrams
        hash_table[key].append(w)

    # return the values of the hash_table
    return list(hash_table.values())
```

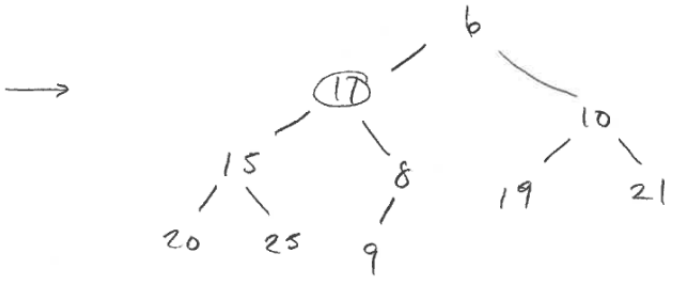
(6) **(10 pts)** Show the resulting binary heap after inserting the following values one at a time {42, 11, 28, 8, 13, 61, 18}

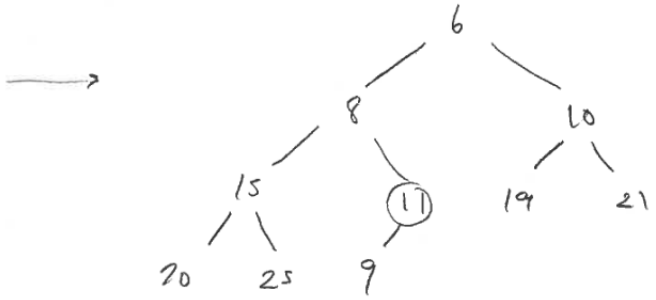


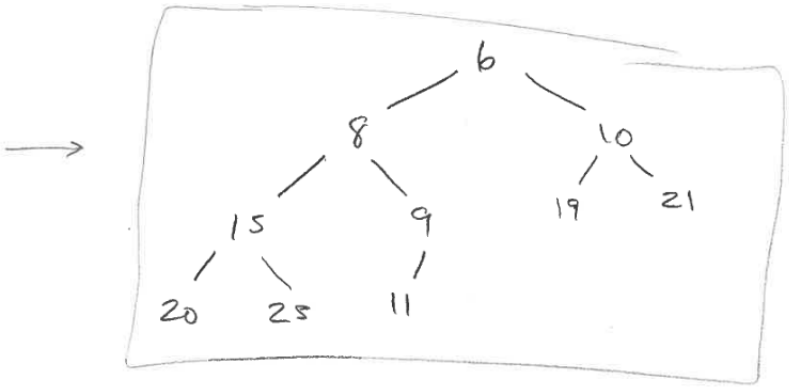
(7) **(10 pts)** Show the result of deleteMin on the following heap



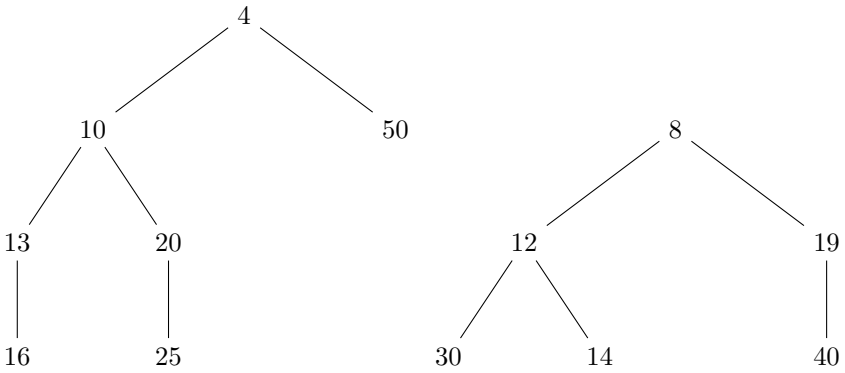

11
6
10
15
8
19
21
20
25
9

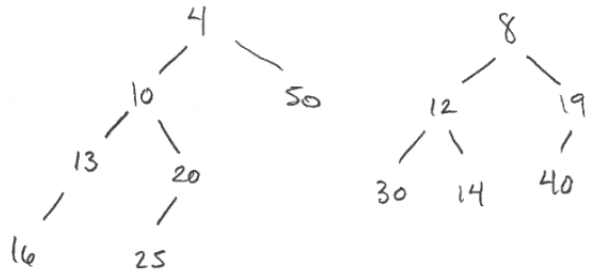

6
17
10
15
8
19
21
20
25
9

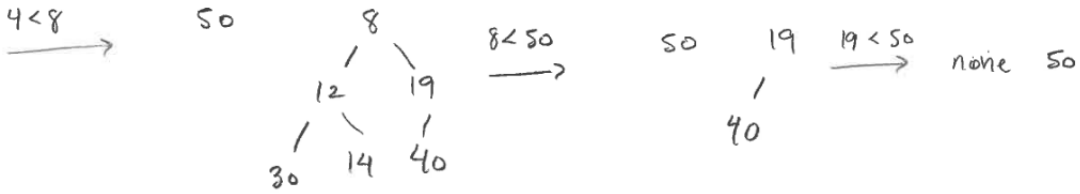

6
8
10
15
11
19
21
20
25
9

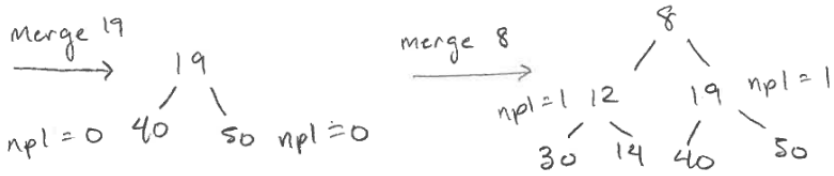

6
8
10
15
9
19
21
20
25
11

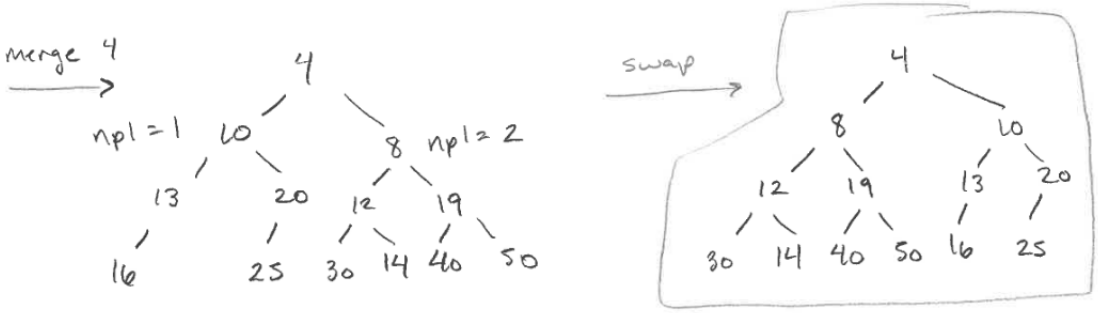
(8) **(10 pts)** Show the merge of the following two leftist heaps.



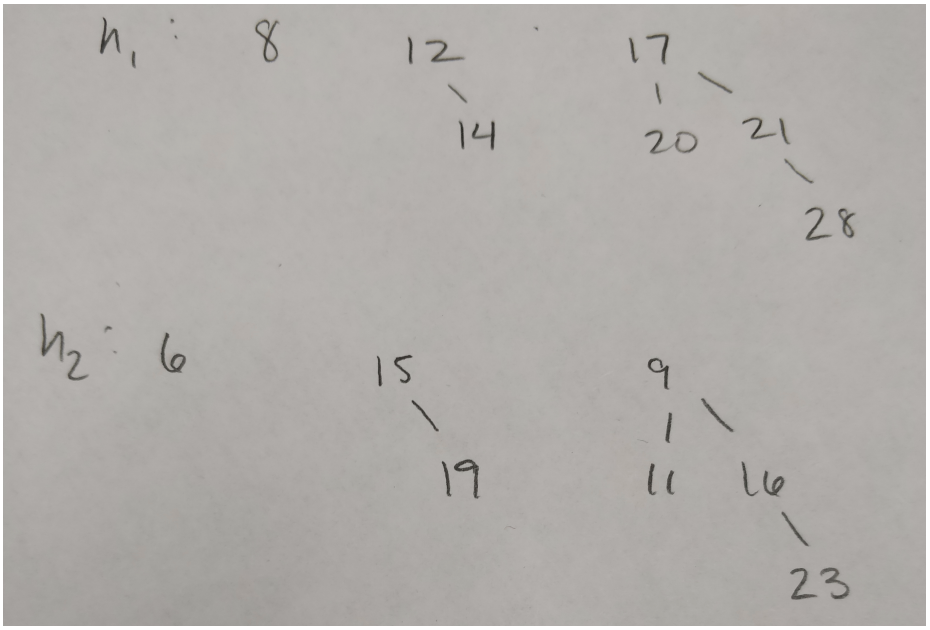








(9) **(10 pts)** Show the result of merging the two binomial queues

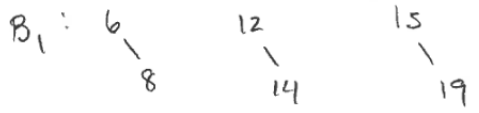


All possible trees

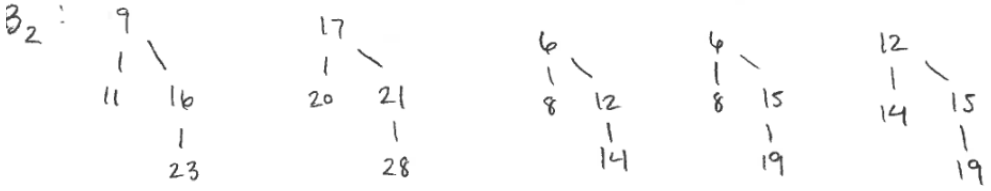
B_0: None

A valid solution would be any 3 B_1, B_2, B_3 tree combination such that no numbers repeat

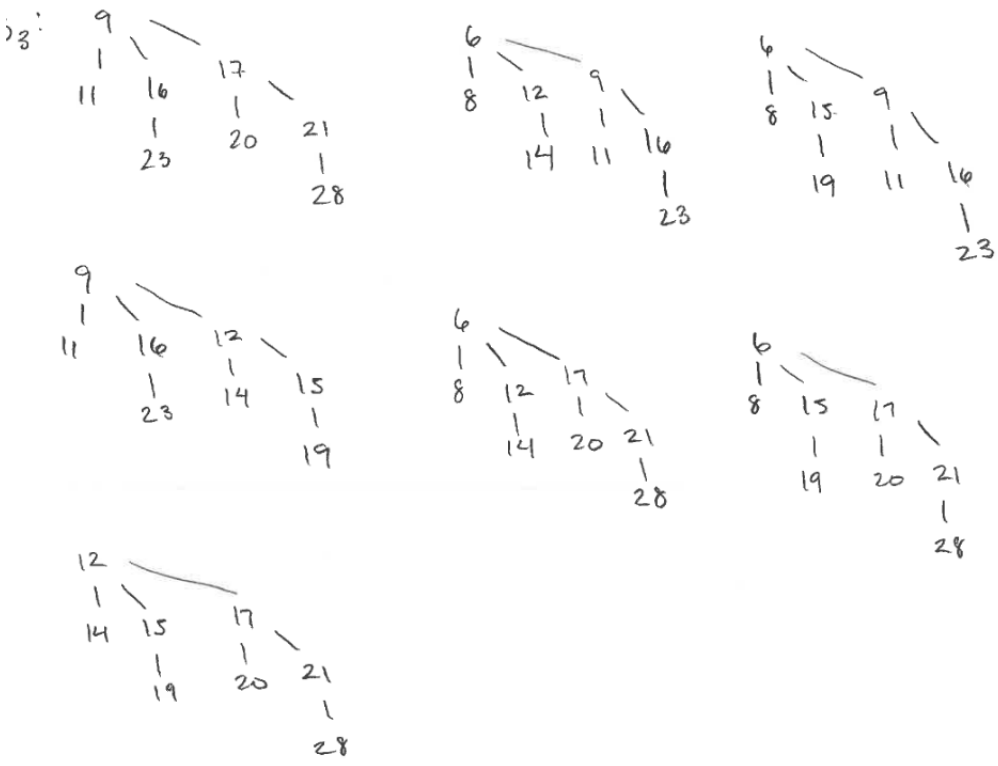
B_1:



B_2:



B_3:



(10) **(20 pts)** Given a non-empty array of integers, return the k most frequent elements in O(n) time.
To do this we can use a heap. Since we want the k most frequent we can simply use a maxheap. Since the built in heap library for python has a minheap we can take the negative of all the counts therefore the largest number is not the min. First we create a hash table to get the counts for each item

```python
import heapq
from collections import defaultdict
def k_largest(arr, k):
    # Get the counts of each element
    # Here we are going to use something called a defaultdict
    # What it does is if the key does not exist it creates a default value for it
    # In this case it will be an integer which defaults to 0

    # Create hash table
    hash_table = defaultdict(int)
    for x in arr:
        hash_table[x] += 1

    # convert to a list of -value,key pairs for the heap
    counts = [(-v,k) for k,v in hash_table.items()]

    # build the heap
    heapq.heapify(counts)

    k_largest = []
    # we then pop k times
    for i in range(k):
        # Here we only want the element not the value
        # Since the heap is in a (-v,k) pair we take the key using index 1
        k.largest.append(heapq.heappop()[1])

    return k_largest
```