Name: Nathan Juman

Week #6 –In Class Activities

PART 1: (5 points)

Refer to 3.13 of your textbook to find the hot potato simulation

1. (1 pt) Which line in the code simulates removing the person with the hot potato?

   Line #13 → sim_queue.dequeue()

2. (1 pt) Who is the 1st and 4th persons removed in the simulation? Hint: use print statements

   1st → David , 4th → Bill

3. (3 pts) Analyze the running time of the hot_potato function as follows:
   a. Assume in our queue implementation enqueue is $O(1)$ and dequeue is $O(n)$
   b. Identify inputs to the function
      let n = size of the name list
      let m = num
      State all your answers in terms of n and m
   c. What is the running time of the for loop in lines 6-7. (there is an enqueue in the loop)

      O(n) → Iterating through the list names and enqueue them.

   d. What is the running time of the for loop in line 10-11? (there is a dequeue and an enqueue)

      O(mn)

   e. What is the running time of line 13?

      O(n)

   f. How many times will the while loop run?

      (n-1) times [length of list minus 1 (which is the last person)]

   g. Putting your answers of *d-f* together, what is the running time of the entire while loop?

      O(mn) → dominant term

   h. Line 15 takes $O(1)$ as by then the queue has size 1. Using this information and your answers to part g and c, what is the running time of the whole function?

      O(n+mn) → O(mn)

Part 2: (5 points)

1. Write the function frequencies that works as described by the header comments and works in $O(n)$ time where n is the size of the list L. Hint: Use a dictionary to help you. Your code should be similar to $Unique(L)$ which is also shown below for your reference

```
#purpose: Prints each number in the list along with the number of times it appears in the list
#input: List of numbers L
#Output: none?
def frequencies(L):
    hashmap = {}
    for i in L:
        if i in hashmap:
            hashmap[i] += 1
        else:
            hashmap[i] = 1
    return hashmap
```

```
#purpose: returns a count of the unique numbers in the list
#input: List of numbers L
#Output: count of the unique numbers in the list
def Unique(L):
    nondups = {}
    for i in L:
        if i not in nondups:
            nondups[i]=1
    return len(nondups)
```

2. Arguing line by line – state why the running time of your `frequencies` method is $O(n)$

Creating an empty hashmap (dict)

for loop iterating through items in a list → O(n)

Checking to see if item is already in hashmap → O(1) * in operator is O(1) for dictionaries*

if in hashmap, increment the key (number) by 1 (the amount of time it appears) → O(1)

else (if item not in hashmap)

assign that item to hashmap w/ value 1 (number has appeared once) → O(1)

O(n) (dominant term)

return hashmap

3. Can there be a more efficient algorithm for computing frequencies (in terms of Big-O)? Explain Why or why not?

Using a dictionary to store the names rather than a list? Accessing keys and their values can be done in O(1) time.

Part 3: 1 pt

The screen at the security desk needs to continuously display footage from each live camera one at a time in 10 secs intervals. For example if cam2, cam5, and cam3 are the cameras that are live now then it should display footage of cam2, then cam5, then cam3. Afterwards it should make a query to get the current set of live cameras and repeat the steps above. The process should go on indefinitely. Assume the queue raises a RuntimeError when a user tries to dequeue from an empty queue.

Explain how the code below implements this using exceptions

```
def security_monitor(q):
    try:
        while True:
            camera_name = q.dequeue()
            show_footage(camera_name) #displays footage from one camera
    except RuntimeError as e:
        get_live_cameras()

def get_live_cameras()
    camera_names = get_list_of_live_cameras()
    q = Queue()
    for c in camera_names:
        q.enqueue(c)
    security_monitor(q)
```

get_live_cameras() retrieves the list of cameras and iterates through each camera to enqueue them. After all the cameras in the list are enqueued, there is a function call to security_monitor(q).

security_monitor(q) uses exceptions to make sure the process is occuring indefinetly. The try: clause removes the camera in the front of the queue (dequeue), and shows the footage. This goes on until the queue is empty, where the exception is raised. Here, when the RuntimeError exception is raised, get_live_cameras is called. This restarts the process, ensuring an indefinite occurence.