Name: Nam Huynh

ID: nsh1507

1.

c. T(1) = 2

T(n) = $(n + 1) + \frac{1}{n}\sum_{q=1}^{n-1} T(q)$

=> nT(n) = $(n^2 + n) + \sum_{q=1}^{n-1} T(q)$

T(n+1) = $(n + 2) + \frac{1}{n+1}\sum_{q=1}^{n} T(q)$

=> (n+1)T(n+1) = $n^2 + 3n + 2 + \sum_{q=1}^{n} T(q)$

(n+1)T(n+1) – nT(n) = $n^2 + 3n + 2 + \sum_{q=1}^{n} T(q) -$

$n^2 - n - \sum_{q=1}^{n-1} T(q)$

(n+1)T(n+1) – nT(n) = $2n + 2 + \sum_{q=1}^{n} T(q) - \sum_{q=1}^{n-1} T(q)$

(n+1)T(n+1) – nT(n) = $2n + 2 + \sum_{q=1}^{n} T(q) - \sum_{q=1}^{n-1} T(q)$

(n+1)T(n+1) – nT(n) = $2n + 2 + T(n)$

$$(n+1)T(n+1) = 2n + 2 + T(n) + nT(n)$$

$$(n+1)T(n+1) = 2n + 2 + (n + 1)T(n)$$

$$T(n+1) = \frac{2n}{n+1} + \frac{2}{n+1} + T(n)$$

$$T(n+1) = 2 + T(n)$$

$$T(n) = 2 + T(n - 1)$$

$$= 2 + 2 + T(n - 2)$$

$$= 2 + 2 + 2 + T(n - 3)$$

$$= 2k + T(n - k)$$ →Identify the pattern

Choose k = n – 1:

$$T(n) = 2(n - 1) + T(1)$$

$$= 2n - 2 + 2$$

$$= 2n$$

Therefore T(n) = 2n

3.

a.

We need to traverse each vertex's adjacency list once to count the number of adjacent vertices. This requires **Θ(V)** time. Additionally, counting the total number of adjacent vertices gives us an extra **Θ(E)** time complexity. Therefore, the traversal for each vertex and edge once would give the time complexity for computing outdegree of **Θ(V + E).** For the indegree of each edge, add one to the in-degree counter for the vertex appearing in the vertex we are currently visiting. Therefore, the traversal for each vertex and edge once would give the time complexity for computing outdegree of **Θ(V + E).**

b.

The expected lookup time is O(L),where L is the load factor. The load factor can also be expressed as V/n, where V is the number of vertices and n is the total entries of the hashtable. The disadvantages of this scheme is that this implementation requires more space complexity than that of the linked list implementation as each key in the hash table needs additional space to store the pointers to the linked lists. Also, when the load factor of the hashtable has exceeded a certain threshold, the resizing of that hashtable would need extra time to move the elements between the old and new hash tables. While the Hashtable provide a fast look-up time for the edges, it requires more space complexity

4.

Assume the stack has the following functions:

isEmpty(stack): check whether the stack is empty

push(val): push a new value to the top of the stack

pop( ): pop a value off the top of the stack and return it

top( ): return the top value of the stack

def DFS(G) :

for $u \in G.V$:

color[u] ← White

π[u] ← NIL

time←0 #global

for $u \in G.V$:

```
                if color[u] = White :

                        DFS_VISIT_STACK(G, u)

def DFS_VISIT_STACK(G, u):

        stack ← []     # initialize empty stack

        time ← time +1
        color[u] ← GRAY

        stack.push(u)

        d[u] ← time # ‘d’ is for discovery

        while not isEmpty( stack ):

                top_val ← top( stack )

                neighbor ←  NIL

                for v ∈  G.A(u) :

                        if color[v] = WHITE:  # set neighbor as the FIRST white adjacent vertex

                                neighbor ← v

                                break

                if neighbor = NIL:  # all the adjacent vertices have been explored

                        time ← time +1
                        color[top_val] ← BLACK

                        f[top_val] ← time       # ‘f’ is for discovery

                        stack.pop( )

                else:     # not all the adjacent vertices have been explored

                        time ← time +1
                        d[neighbor] ← time

                        π[neighbor] ← top_val

                        color[neighbor] ← GRAY
```

```
stack.push( neighbor )
```