HW#4

1.

Each PUSH and POP operation needs $O(1)$ cost.

Because the stack size never exceeds k, so that each COPY operation needs at most $O(k)$ cost.

Since k PUSH and POP operations occur between two consecutive COPY operations. It needs $O(k)$ cost for k PUSH and POP operations. And one COPY operation need at most $O(k)$ cost.

Totally, k PUSH and POP operations and one COPY operation need $O(k)$ cost.

So the amortized cost of each operation is $O(1)$.

Then we can show that the total cost of n operations is $O(n)$

2.

a)

Let S_1 be the enqueue stack, S_2 be the dequeue stack.

ENQUEUE(x)	DEQUEUE()

ENQUEUE(x){
    S1.push(x)
}
	DEQUEUE(){
	    if S2 is not empty
	        return S2.pop();
	    else
	        while S1 is not empty
	            S2.push(S1.pop())
	        return S2.pop();
	}

b)

operation	actual cost	amortized cost	
ENQUEUE	1	3	$O(1)$ amortized cost
DEQUEUE	1	1	

Intuition: When ENQUEUEing an object, pay $3

- $1 pays for the actual cost to push an element into S1:
  S_1.push(x)
- $2 is the prepayment for it being moved from S1 to S2:
  S_2.push(S1.pop())

And the DEQUEUE pay $1 for itself.

Amortized cost:

Each object has credit≥0, so total amortized cost is $O(n)$ for n operations, $O(1)$ for each operation.

c)

Define the potential cost $\Phi(D_i)$ = # of the elements in the enqueue stack, denoted as m_i in the following statement..

$\Phi(D_0)$ = 0, the enqueue stack is empty initially.

- cost of operation ENQUEUE:

  $$\begin{aligned}\hat{c_i} &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &= 1 + m_i - m_{i-1} \\ &= 1 + (m_{i-1} + 1) - m_{i-1} \\ &= 1 + 1 = 2 = O(1)\end{aligned}$$

- cost of operation DEQUEUE – dequeue stack is not empty:

  $$\begin{aligned}\hat{c_i} &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &= 1 + m_i - m_{i-1} \\ &= 1 + (m_{i-1}) - m_{i-1} \\ &= 1 + 0 = 1 = O(1)\end{aligned}$$

- cost of operation DEQUEUE – dequeue stack is empty:

  $$\begin{aligned}\hat{c_i} &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &= (m_{i-1} + 1) + m_i - m_{i-1} \\ &= (m_{i-1} + 1) + 0 - m_{i-1} \\ &= 1 = O(1)\end{aligned}$$

So, amortized cost of each operation is $O(1)$.

Amortized cost of a sequence of n operations is $O(n)$

※The definition of potential cost above is not the only solution.

3.

a)

First, do MAKE-SET on the n elements.

Then create a single set whose depth is $\lg n$ with n-1 UNIONs:

$\text{UNION}(x_1, x_2), \text{UNION}(x_3, x_4), \ldots \text{UNION}(x_{n-1}, x_n)$

$\text{UNION}(x_2, x_4), \text{UNION}(x_6, x_8), \ldots \text{UNION}(x_{n-2}, x_n)$

...

$\text{UNION}\left(x_{n/2}, x_n\right)$

Last, perform m – 2n + 1 FIND-SETs on the deepest node in the tree. Since the tree's depth is $\lg n$, the running time of each FIND-SET is $\boldsymbol{\Omega}(\lg n)$.

The total time for the entire sequence is bounded below by the time to perform all the FIND-SET operations. So letting $m \geq 3n$, we have more than m/3 FIND-SET operations, so the total running time is $\boldsymbol{\Omega}(m \lg n)$.

b)

We will prove by induction that any node of rank r has at least $f(r) \geq 2^r$ descendants (including itself).

When r = 0, the node has 1 descendant.

Assume $f(r) \geq 2^r$ for r < k. For r = k, the rank k node first becomes rank k when two nodes of rank k – 1 are linked. By the inductive assumption, each of these two nodes has at least 2^{k-1} descendants. Therefore the node with rank k has at least $2^{k-1} + 2^{k-1} = 2^k$ descendants.

This proves that any node of rank r has at least $f(r) \geq 2^r$ descendants. It follows that every node has rank at most $\lfloor \lg n \rfloor$.

c)

We only need to show that every operation takes $O(\lg n)$ time. This is easy given the result of 21.4-2.

MAKE-SET and LINK takes $O(1)$ time.

Recall that the rank of a node is an upper bound on the height of the node.

This implies that every tree has height at most $\lfloor \lg n \rfloor$. FIND-SET in a tree of height at most $\lfloor \lg n \rfloor$ takes $O(\lg n)$ time.

UNION is a combination of two FIND-SET and one LINK, which takes $O(\lg n)$ time in total. This finishes the proof that a sequence of m operations take $O(m \lg n)$ time.

4.

1)

Suppose that for every cut of G, there is a unique light edge crossing the cut.

Let's consider two minimum spanning trees *T* and *T'* of G.

Consider any edge $(u, v) \in T$. If we remove (u, v) from *T*, then *T* becomes disconnected, resulting in a cut $(S, V - S)$. The edge (u, v) is a light edge crossing the cut $(S, V - S)$.

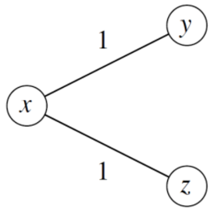
Now consider the edge $(x, y) \in T'$ that crosses $(S, V - S)$. It is also a light edge crossing this cut.

Because the light edge crossing $(S, V - S)$ is unique, so that the edges (u, v) and (x, y) are the same edge.

Then we can show that every edge in *T* is also in *T'*.

2)

Here's a counterexample for the converse:



The graph is unique minimum spanning tree.

Consider the cut ({x}. {y, z}). Both of the edges (x, y) and (x, z) are light edges crossing the cut , and they are both light edges.