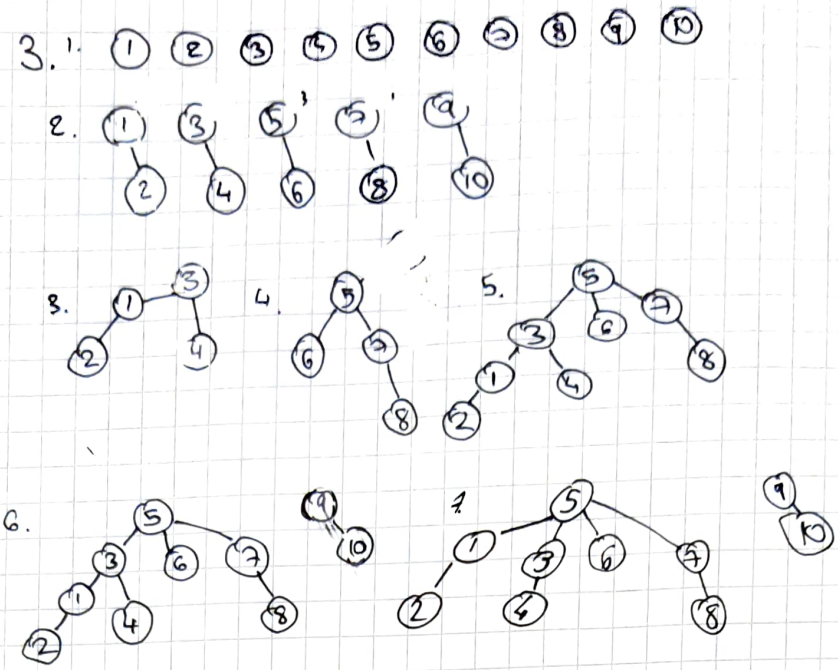
Leonard Blam
900086
Yoseß Tyberg
3247 10144

HW1

1. After 0 unions, the largest set has 1 element. In order to get the larger possible set with any one union, the largest set and the second largest set will have to be joined. So the first union will make one set with 2 elements and the rest will still have 1 element, and all subsequent unions will take a set with 1 element and add it to the largest set.
So if it any point, after v unions, there are v+1 elements in the largest set, then after v+1 unions, after adding a set with one element to the largest set, there will be v+2 elements in the largest set.
That is a proof by induction.

2. The worst case is n makeset and then n-1 unions - so total of 2n-1 operations.
With n sets, each element's respesentive pointer is updated at most logn - times. It's only updated if its set size is ≤ to the set joining, so after the kth update an element's set has at least 2^k elements and since $2^k \leq n$, $k \leq \log n$. Since there are n elements in total there exist a case where every union required the max # of updates. A total of $\Omega(n \log n)$
m-operations are not given in the question, so I assume that we have a separate operation of $\Omega(m)$ times
Then the result together is $\Omega(m + n\log n)$

3.



2 trees : 1 with the height of max 2
and another one with height 1

4a. we make n-times make set. Then we go over each edge what is n-1. And we go over if find set(1) ≠ find set(2) then does n-1 Unions. What in worset case is $O(n! - (n-1))$

b. $O(n! \quad , +n)$

(could be the inverse Ackermann function and become linear of $O(n)$)

5.
```
Template <type T>
void make_set(T n) {
T list = new <T> list;
list->head = n;
list->next = n->tail;
list->tail = 0;
int length = (n->length)+1;
n->tail = list;      }

node<T>* Find_set(T n){
    return &n->head;  }

void Union(T n1, T n2 ) {
node<T> first = find_set(n1);
node<T> second = Find_set(n2);
node<T> largerset , smallerset;
if (first->length ≥ second->length ) {
    larger_set = first;
    smaller_set = second;  }
else {
    larger_set = second;
    smaller_set = first;  }
    larger_set->last->next = smaller_set->first;
    smaller_set->head->last = smaller_set->last;
for (auto i = smaller_set->begin(); i < smaller_set->end(); ++i){
    (*i)->head = larger_set->head;  }
delete small_set;
}
```