1. (10pts, 1 page)
Prove that every node has rank at most $\lfloor \log n \rfloor$ in Union-Find algorithm with two heuristics (1) union by rank and (2) path compression.

Sol:
By Induction:

Claim:
A node with rank $r$, then it is the root of a subtree of size at least $2^r$.

Base case:
A node of rank 0 is the root of a subtree that contains at least itself (and so is of size = at least 1).

Inductive case:
A node $X$ can have rank $(r + 1)$ only if, at some previous stage, it had a rank $r$ and it was the root of a tree that was joined with another tree whose root had rank $r$. Then $X$ became the root of the union of the two trees. Each tree, by inductive hypothesis is of size at least $2^r$, and so now $X$ is the root of a tree of size at least $2^r + 2^r = 2^{r+1}$.

Now the number of nodes in the forest is $n$ and we have at least $2^r$ nodes in every tree with rank $r$. So,

$$n \geq 2^r \Rightarrow r \leq \lfloor \log n \rfloor$$

2. (15pts, 1/2 page per part)
Consider the following code segment:

```plaintext
for i = 1 to n
   makeset(i)
for i = 1 to n - 1
   union(find(i), find(i + 1))
for i = 1 to n
   find(i)
```

For each of the following cases, analyze the time complexity of the above sequence:

(a) when no weighted union and no path compression heuristic is used.
   (assuming union($p, q$) always makes $p$ a child of $q$.)

(b) when only weighted union heuristic is used.

(c) when both weighted union and path compression heuristic are used.
Sol:
There are $n$ Makeset, $2n$ Find and $n$ Union.

(a) The Time complexity is $O(4n + n^2) = O(n^2)$.

(b) The Time complexity is $O(4n + n \lg n) = O(n \lg n)$.

(c) The Time complexity is $O(4n \alpha(n)) = O(n \alpha(n))$.

Note: ‘Union by Rank’ also give you the same complexity.

3. (10pts, 1 page)
A priority queue is essentially a list of items in which each item has associated with it a priority. Items are withdrawn from a priority queue in order of their priorities starting with the highest priority item first. If the maximum priority item is required, then a heap is constructed such than priority of every node is greater than the priority of its children.

Design such a heap where the item with the middle priority are withdrawn first. If there are $n$ items in the heap, then the number of items with the priority smaller than the middle priority is $\frac{n}{2}$ if $n$ is odd, else $\frac{n}{2} \pm 1$. Specifically, you should explain how the withdraw and insert operations work, calculate their complexity, and how the data structure is constructed.

Sol:

You can use one min heap and one max heap such that root of the min heap is larger than the root of the max heap. The size of the min heap should be equal or one less than the size of the max heap. So the middle element is always the root of the max heap.

For the insert operation, if the new item is less than the root of max heap, then insert it into the max heap, else insert it into the min heap. After the withdraw or insert operation, if the size of heaps are not as specified above than transfer the root element of the max heap to min heap or vice-versa.

By this implementation, insert and withdraw operation will be in $O(\lg N)$ time.

4. (15pts, 1 page)
Given the vertex coordinates of a set $X$ of $n$ axis-parallel rectangles in the plane (see figure), each containing the origin $(0, 0)$ somewhere on its boundary, compute the union of rectangles in $X$ such that the union is represented as a polygon whose vertices are listed in counterclock-wise order. Prove that solving this problem takes $\Omega(n \log n)$ time.
Sol:
This is a lower bound argument. Sorting can be reduced to this problem, and thus the lower bound of sorting $\Omega(n \ lg \ n)$ will be transferred to this problem. W.L.O.G, assume we have a non-zero sequence of distinct positive numbers - $a_1$, $a_2$, $a_3$,... $a_n$, now for each $a_i$, take a rectangle with the following co-ordinates:

\[ [(a_i, 0), (0, 0), (a_i, \frac{1}{a_i}), (0, \frac{1}{a_i})]\]

If we give these co-ordinates to our oracle, it will return the final co-ordinates(counterclock-wise) of the union of rectangles as -

\[ (x_1, y_1), (x_2, y_2), ... (x_{2n+1}, y_{2n+1}), (0, 0)\]

Now consider the sequence $(x_{2i}, y_{2i}) \forall i = 1 to n$

As our mapping(from number to rectangle) was unique, we can map back each point $(x_{2i}, y_{2i})$ to the actual numbers $a_i$ - and they will be in descending order. Also mapping takes $O(n)$ time which is $o(n \ lg \ n)$ (small-o).

NOTE:
If we have a mix of positive and negative numbers we will give the oracle two sets - one containing the positive ones and another set containing absolute value of the negative numbers, and do the same thing for each set and combine appropriately.