You may answer the questions in any order. Dishonest behaviors and attempts will be most seriously punished. When you are asked to justify, prove, or disprove your answers, you may directly use anything that we have shown in class in a "black-box" manner.

**Problem 1** (10 points)
State the master method.

**Problem 2** (15 points)
An array $A[1 \ldots n]$ is strongly heapified if
- $A[i] \leq A[i/2], i = 2, 3, \ldots, n$ and
- $A[2j] \geq A[2j + 1], j = 1, 2, \ldots, \lfloor (n - 1)/2 \rfloor$.

What is the worst-case asymptotic time complexity of problem of reordering any arbitrary array $A$ into a strongly heapified array. Namely, is it $\Theta(n)$, $\Theta(n \log n)$, $\Theta(n^2)$ or something else? Justify your answer.

**Problem 3** (15 points)
Let $S$ be a sequence of $n$ distinct numbers. $S$ is monotonic if $S$ is either increasing or decreasing. $S$ is tritonic if $S$ consists of three disjoint monotonic subsequences. Note that each disjoint subsequence could be empty and the “direction” (i.e., increasing or decreasing) of two consecutive subsequences could be the same. (Therefore, a monotonic sequence is also tritonic.)

Now, suppose that $S$ is tritonic. Prove or disprove that the maximum of $S$ can be determined in $O(\log n)$ time.

**Problem 4** (15 points)
Let $T$ be a rooted tree of $n$ weighted nodes. The node weights are given an array $w$, where $w[i]$ denotes the weight of node $i$, for each $i = 1, \ldots, n$. The structure of $T$ is given by an array $p$, where $p[i]$ denotes the parent of node $i$. The unique node $r$ with $p[r] = r$ is the root of $T$.

We are interested in finding a set of non-adjacent nodes whose sum of weights is maximized. Prove or disprove that the problem can be solved in $O(n)$ time.

**Problem 5** (15 points)
In class we see how to analyze the amortized cost of each operation for dynamic tables using the following potential function:

$$
\Phi_i = \begin{cases} 
2 \cdot num_i - size_i & \text{if } num_i / size_i \geq 0.5; \\
size_i / 2 - num_i & \text{if } num_i / size_i < 0.5,
\end{cases}
$$

where $num_i$ (respectively, $size_i$) is the number of elements (respectively, table size) right after the $i$-th operation. You are now asked to show that when $num_{i-1} / size_{i-1} \geq 0.5$ and the $i$-th operation is $DELETE$, the amortized cost for the $i$-th operation is $O(1)$.

As an alternative, you can provide an implementation for dynamic tables such that each operation (including, $CREATE$, $INSERT$ and $DELETE$) takes worst-case $O(1)$ time. Justify your answer.
Problem 6  (15 points)

As we mentioned in class, a 2-3-4 tree is a B-tree with \( t = 2 \). Please show the resulting 2-3-4 tree after inserting the following numbers in order into an empty tree.

\[
5, 3, 7, 9, 10, 8, 1, 2, 4, 6, 11, 12.
\]

No need to show the intermediate steps. Only the final tree counts.

Problem 7  (15 points)

In class we study Tarjan’s implementation for disjoint sets using forests with two heuristics (path compression and weighted union). In a sequence of \( n \) operations for disjoint sets, the amortized cost for each operation (MAKE-SET, UNION, or FIND-SET) is \( O(\alpha(n)) \).

Now, we keep weighted-union but discard path-compression. That is, the implementation becomes the following.

```
MAKE-SET(x)
1  p[x] ← x
2  rank[x] ← 0

UNION(x, y)
1  LINK(FIND-SET(x), FIND-SET(y))

LINK(x, y)
1  if rank[x] > rank[y]
2    then p[y] ← x
3  else p[x] ← y
4  if rank[x] = rank[y]
5    then rank[y] ← rank[y] + 1

FIND-SET(x)
1  while p[x] ≠ x
2    x ← p[x]
3  return x
```

Show that the amortized cost of each operation is \( O(\log n) \).