Please read the instructions carefully:

1. There are two stages to each assignment:

   (a) For the first stage, you need to submit your homework solution as a single pdf file to NYU Classes. It can be either a scanned copy of your written solution or photos joined into a pdf. You are also allowed to typeset the solutions in MS Word/LaTeX. In case of scanned copies it is your responsibility that the solution is readable. There are no late submissions accepted, as sample solution outlines (to be used for the second stage of the homework) will be given out on the day of the submission of the first stage of the homework. You should leave around 5 to 6 cm of space between each part of the answers for the second stage of the homework.

   (b) For the second stage of the assignment, you are required to critique your own solutions and analyze the mistakes that you made. Based on the outlines provided, you need to write the self criticism in the empty space that you left in your homework. Do not rewrite your solution, you need to understand and point out the mistakes that you made when writing the original solutions.

      A good analysis of your solution should summarize the key steps, ideas or techniques that you previously missed. It should prompt you to reorganize your thoughts and think about what you can do to improve the solutions. Even in the case where you completely missed the solution it is insightful to point out why the approach adopted by you does not work and identify the step where it fails. This will allow you to get full credit on the self evaluation part of the homeworks and of course help you in understanding the question.

      Mark correct solutions in green, incorrect solutions in red and if you are unsure about any solutions write it in blue.

2. For each question where you are required to design an algorithm, you need to clearly explain the algorithm and prove the running time complexity as asked in the question.

3. You can assume $O(1)$ complexity of arithmetic of numbers (e.g. multiplication/division of numbers and calculations of $n!$) unless otherwise stated in the question.
HW 1

Question 1. (5 points) AVL tree mechanics

1. (3 points) Draw an AVL tree (initially empty) at each step when inserting the following numbers in order:
   
   \[ 1, 2, 5, 4, 6, 3, 10, 9, 7, 8 \]

2. (2 points) Draw the above AVL tree at each step when deleting the following numbers in order (assuming that the substitution on deleting a node is done by replacing it with the minimum in the right subtree):
   
   \[ 4, 5, 6 \]

Question 2. (5 points) Heap mechanics

Draw a min-heap (initially empty) at each step when inserting the following numbers in order:

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

Question 3. (5 points) Fibonacci recursion

Prove by induction, that the \( n \)'th Fibonacci number can be found by the formula

\[ F_n = \frac{\Phi^n - \phi^n}{\sqrt{5}} \]

where \( \Phi = \frac{1+\sqrt{5}}{2}, \phi = \frac{1-\sqrt{5}}{2} \).

Question 4. (10 points) A heaping challenge

Given a min heap containing \( n \) data items, along with a data item \( x \) and a positive integer \( k \), our task is to design an algorithm that runs in time \( O(k) \) and answers the following question: are there at least \( k \) items in the heap that are less than \( x \)? Of course, we could go through the entire heap and just count the number of items that are less than \( x \), but this would take time proportional to \( n \). The challenge is to design an algorithm whose running time is \( O(k) \) by somehow using the heap property.

Question 5. (10 points) Very sparse numbers

A positive integer is called sparse if its binary representation does not contain any consecutive 1’s, e.g.

\[ n = 69 \rightarrow 1000101 \]

In this problem, we want to calculate the number of sparse integers with bit length less than or equal to \( k \), denoted by \( s_k \).

E.g. for \( k = 3 \), the only sparse numbers of bit length bounded by 3 are \( 4 \rightarrow 100, 5 \rightarrow 101, 2 \rightarrow 10, 1 \rightarrow 1 \), which gives us that \( s_3 = 4 \).

Give an algorithm to calculate \( s_k \) in \( O(\log k) \).

Note 5.1. 0 is not a sparse number.

Question 6. (10 points) Recursive permutations

For every positive integer \( n \), there are \( n! \) permutations of \( \{1, 2, \cdots, n\} \). There is an ordering on the
permutations defined by the lexicographic (dictionary) order on lists, e.g. for $n = 3$ we have the following ordering of the permutations:

1, 2, 3  
1, 3, 2  
2, 1, 3  
2, 3, 1  
3, 1, 2  
3, 2, 1

1. (5 points) Give a recursive algorithm which takes two positive integers $n$ and $k \in \{1, 2, \cdots, n!\}$, which then returns the $k^{th}$ permutation of $\{1, 2, \cdots, n\}$ in $O(n^2)$. E.g. for $n = 3$, $k = 4$ the algorithm should output $2, 3, 1$

**Hint 6.1.** Try to find the first number in the permutation, by counting the permutations.

2. (5 points) Give a recursive algorithm which takes a permutation of $\{1, 2, \cdots, n\}$ and returns the position of the permutation in the lexicographic ordering of all permutations in $O(n^2)$. E.g. for the input $(3, 1, 2)$, the algorithm should output 5.

**Hint 6.2.** Try to reverse the above procedure to get the position.

**Question 7. (10 points) Sweeping the plane**

Given a set of points $\{(x_i, y_i)\}_{i=1}^{n} \in \mathbb{R}[2]$, give an algorithm to find the number of points in the lower left quadrant of each point, in $O(n \log n)$. E.g. for the following set of points:
The algorithm should have the following output:

\[(A : 0), (B : 1), (C : 2), (D : 3), (E : 3), (F : 5), (G : 6), (H : 7), (I : 7), (J : 9)\]

**Note 7.1.** Assume that the points are given in left to right ordering.

**Hint 7.1.** Use an AVL tree to store the points, iterating from left to right (what ordering is there on the points?). Show how we can use this to find the number of points below each point. What augmentations do we need the tree to have if we want points which are both to the left and below our current point?

**Question 8.** (10 points) Linear equations

1. (8 points) Solve the following congruence equations
   
   (a) (2 points) \(13x + 5 = 6x + 15 \pmod{20}\)
   (b) (2 points) \(13x + 15 = 6x + 5 \pmod{20}\)
   (c) (2 points) \(13x + 7 = 8x + 12 \pmod{20}\)
   (d) (2 points) \(13x + 7 = 3x + 12 \pmod{20}\)

2. (2 points) Find \(7^{59} \pmod{100}\).
Question 9. (10 points) Cutting down costs
Given a list of positive integers $a_1, a_2, \ldots, a_n$, we can perform the following operation on the list any number of times

- If $a_i > a_j$, then $a_i \leftarrow a_i - a_j$

What is the minimum value of $\sum_{i=1}^{n} a_i$ that we can achieve in this manner?
E.g. for the list $4, 2, 4$

We can do the following operations

\begin{align*}
a_1 &= a_1 - a_2 \\
a_3 &= a_3 - a_2
\end{align*}

to get $2, 2, 2$

As we can’t reduce the sum any further the minimum is 6.

**Note 9.1.** For this question we are not required to give an algorithm. Instead we need to just find the minimum possible value of this sum.
We still need to prove our claim about the value that we suggest.

**Hint 9.1.** Try to find similarities to Euclid’s algorithm for finding the GCD.

Question 10. (15 points, BONUS) Uneven tree updates
Given a tree with $n$ nodes, where each node has a key, value, we want to perform the following operations on it:

- Increment($k, v$): Increase the value at all nodes in the subtree of the node with the key $k$ by $v$, in $O(\log n)$.
- Find($k$): Return the value of associated with the key $k$, in $O(\log n)$.

**Note 10.1.** Note that this is not necessarily a binary tree nor is it a search tree. The tree structure is not going to change throughout these procedures.
Each node has a unique key associated with it.

An example of an input tree is:

```
   C : 5
  /   \
E : 4  A : 10
    /   \
   D : 20 \\  B : 2
```

**Note 10.2.** For either of these operations we first need to locate a node with key $k$. As this is not a search tree the only way to search is to go through the whole tree. A workaround is to store the locations of the nodes in a dictionary (can be implemented using any data structure, e.g. AVL
tree) and go straight to these locations. After we locate $k$, to update the subtree, we need to go to each descendant and update them as well. A workaround for this is to store a carry as we did in the range update for the AVL tree. This makes the update operation $O(1)$. But if we do this, then, to get the value associated with a key, we would need to travel all the way to the root to add the carry values. This is still $O(n)$ as we are not guaranteed to have a height balanced tree. So these augmentations are not enough to do both operations in $O(\log n)$.

1. (4 points) Show how to convert this tree to an array, where each Increment($\cdot$, $\cdot$) operation of the tree corresponds to incrementing the values of a contiguous subarray.

   **Hint 10.1.** Try to use the preordering of the given tree.

2. (5 points) Describe the augmentations needed to implement both of the operations on the list using an AVL Tree.

   **Hint 10.2.** We need to store nodes by position. Remember that we implemented an array interface using AVL trees

3. (6 points) Show how to implement both of the operations using the AVL tree as this list. Prove that the time complexity for each operation is $O(\log n)$