Homework 2 (Due: June 20)

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.
Question 1. (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 2. (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 3. (10 points) Longest divisible subsequence
Given an array of positive integers, \([a_1, a_2, \ldots, a_n]\), a subsequence, \([a_{i_1}, a_{i_2}, \ldots, a_{i_k}]\), is called a divisible subsequence if

\[ a_j | a_{j+1} \quad \forall j \in \{1, \ldots, k-1\} \]

Give an algorithm which takes an array and gives back the length of the longest divisible subsequence in \( O(n^2) \).

Question 4. (10 points) Potholes on the road
Given a 2-D matrix \( M \) of size \( n \times n \), with each entry being 0 or 1, a path in this matrix is defined as a way of going from the top left corner, \((1, 1)\), to the bottom right corner, \((n, n)\), but with the constraint that the movement is restricted to moving only down or right. For any such path, we define its weight to be equal to the number of 1’s in the path.

Give an algorithm which takes a binary matrix of size \( n \times n \) and gives back a path of least weight in \( O(n^2) \).

Question 5. (25 points) Summing up numbers

1. (10 points) Given an array, \([a_1, a_2, \ldots, a_n]\), we want to build a database which can calculate \( \text{CubeSum}(l, r) = \sum_{i=l}^{r} a_i^3 \) for queries of the form \((l, r)\), \(1 \leq l \leq r \leq n\).

Give an algorithm which takes an array and builds a database in \( O(n) \), where the database lets us answer each query in \( O(1) \).

**Hint 5.1.** Try to find a function \( f \) such that \( \text{CubeSum}(l, r) \) can be expressed in terms of \( f(r) \) and \( f(l) \).

2. (5 points) Give an algorithm which finds the subarray \((l, r)\) with maximum value of \( \text{CubeSum}(l, r) \) in \( O(n^2) \).

**Hint 5.2.** Use the database that we made in the previous part.

3. (10 points) Give an algorithm which finds the subarray \((l, r)\) with maximum value of \( \text{CubeSum}(l, r) \) in \( O(n) \).

**Hint 5.3.** For each index \( r \), try to find the index \( l_r \), which maximizes the \( \text{CubeSum}(l_r, r) \). Use the index \( l_{r-1} \) to find the \( l_r \).
Question 6. (15 points) Looking for stability

An array of integers is called almost stable if the difference between consecutive values is less than or equal to 1 and it is called stable if the difference between the maximum and the minimum of the array is less than or equal to 1.

Give an algorithm which takes an array, \([a_1, a_2, \ldots, a_n]\), which is almost stable and returns the length of the longest stable subarray in \(O(n)\).

Hint 6.1. First try to solve this problem when no two consecutive elements are the same.

For each index \(r\) try to find the previous index \(l_r\) which maximizes the length of the stable subarray ending at \(r\). Use the index \(l_{r-1}\) to find \(l_r\).

Question 7. (20 points - Extra credit) Recursing on XOR

1. (10 points) For an array of integers, \([a_1, a_2, \ldots, a_n]\), we define a recursive function, \(RXOR\), on the array as follows:

\[
RXOR([a_1, a_2, \ldots, a_n]) = \begin{cases} 
  a_1 & \text{if } n = 1 \\
  RXOR([a_1 \oplus a_2, a_2 \oplus a_3, a_3 \oplus a_4, \ldots, a_{n-1} \oplus a_n]) & \text{otherwise}
\end{cases}
\]

where \(\oplus\) is the bitwise XOR function.

Give an algorithm which takes an array and returns the value of the function \(RXOR\) for each subarray, \([a_l, a_{l+1}, \ldots, a_r]\), \(1 \leq l \leq r \leq n\), in \(O(n^2)\).

Hint 7.1. Instead of the normal recursion where we would be doing recursion on pairs of \((l, r)\), where \(l\) and \(r\) are the endpoints of the subarray, try to think of doing recursion on pairs \((i, l)\), where \(i\) denotes the starting index of the subarray and \(l\) denotes the length of the subarray. Each subarray \((l, r)\) in the original format can be represented as \((l, r-l+1)\) in the new format.

The advantage of the new representation is that we can express the recursion more intuitively by calculating \((i, l)\) in terms of \((i, l-1)\) and \((i+1, l-1)\).

Hint 7.2. You might need the fact that

\[
x \oplus y = y \oplus x, \quad x \oplus y \oplus y = x
\]

2. (10 points) We define another function, \(RXOR_{\max}\), based on the previous function, as follows:

\[
RXOR_{\max}([a_1, a_2, \ldots, a_n]) = \max_{1 \leq l \leq r \leq n} RXOR([a_l, a_{l+1}, \ldots, a_r])
\]

Give an algorithm which takes an array and returns the value of the function \(RXOR_{\max}\) for each subarray, \([a_l, a_{l+1}, \ldots, a_r]\), \(1 \leq l \leq r \leq n\), in \(O(n^2)\).

Hint 7.3. Using the representation for recursion as defined previously, try to find a recursive method for calculating \(RXOR_{\max}\) on the array \((i, l)\) in terms of \(RXOR_{\max}\) of \((i, l-1)\) and \((i+1, l-1)\) and \(RXOR\) of \((i, l)\).