P1.

Solve the recurrence relation without using Master’s theorem:

\[ T(N) = 3T(N/2) + cN \]

Ans.:

Assume \( N = 2^k \)

\[ T(2^k) = 3 T(2^{k-1}) + c \cdot 2^k \]

After \( k \) steps

\[ = 3^k T(1) + c \left( 2^k + 3 \cdot 2^{k-1} + \frac{3}{2} \cdot 2^{k-2} + \ldots + \frac{3}{2} \right) \]

\[ = 3^k d + c \cdot 2^k \left( 1 + \frac{3}{2} + \left(\frac{3}{2}\right)^2 + \left(\frac{3}{2}\right)^3 + \ldots + \left(\frac{3}{2}\right)^{k-1} \right) \]

\[ = 3^k d + c N \left[ \frac{(3/2)^k - 1}{3/2 - 1} \right] \]

\[ = 3^k (d + 2c) - 2cN \]

Now, \( 3^k = (2^{\log_3 2})^k = (2^k)^{\log_3 2} = N^{\log_3 2} \)

\[ = N^{\log_3 2} (2c + d) - 2cN \]

\[ = \Theta (N^{\log_3 2}) \]

P2.

A and B are playing a guessing game where B first thinks up an integer \( X \) (positive, negative or zero, and could be of arbitrarily large magnitude) and A tries to guess it. In response to A’s guess, B gives exactly one of the following three replies:

a) Try a bigger number
b) Try a smaller number or
c) You got it!!

Design an efficient algorithm to minimize the number of guesses A has to make. An example (not necessarily an efficient one) below:

<table>
<thead>
<tr>
<th>A’s guess</th>
<th>B’s response</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Try a bigger number</td>
</tr>
<tr>
<td>20</td>
<td>Try a bigger number</td>
</tr>
<tr>
<td>30</td>
<td>Try a bigger number</td>
</tr>
<tr>
<td>40</td>
<td>Try a smaller number</td>
</tr>
<tr>
<td>35</td>
<td>You got it</td>
</tr>
</tbody>
</table>
Ans:

First find out whether $X$ is positive or negative by guessing if $X$ is ZERO. Say from the response you make out that it is positive. Then make the next guess “1”. As long as you keep getting the response of “try a bigger number”, keep on doubling your guess until you finally get the response “try a smaller number” for some guess $P$. Now you know that the number must lie between $P$ and $P/2$, so do a binary search between them. The complexity is $O(\log X)$.

P3:

There are $n$ students in a class. The test results are out and assume, for your convenience, that all the students had distinct grades (numbers). You can think of the test result as an unsorted integer array. A student $X$ has been told that his rank in the class is $R$ ($R$ is an integer and obviously, $1 \leq R \leq n$). He wants to find out the $k$ boys who are ranked closest to him ($k/2$ students below him, and $k/2$ students above). Devise an efficient algorithm to identify the scores of these $k$ boys.

Ans:

Since $X$’s rank is $R$, the $k$ boys’ rank must vary between $R-k$ and $R+k$. First find the $(R-k)^{th}$ and $(R+k)^{th}$ ranked element using Selection (linear time). Then do Partition twice to discard all the elements that fall out of the range ($(R-k), (R+k)$). It is linear.
**FEEDS:**

**P1.**
Prove that \( \log(N!) = \Omega(N \log N) \) without using Sterling’s formula.

**Ans.**

\[
\log(N!)
= \log N + \log(N-1) + \log(N-2) + \log 2 + \log 1
= \log N + \log(N-1) + \cdots + \log\left(\left\lfloor \frac{N}{2} \right\rfloor \right) + \log\left(\left\lfloor \frac{N}{2} \right\rfloor - 1 \right) + \cdots + \log 1
\geq \left(\left\lfloor \frac{N}{2} \right\rfloor \log\left(\left\lfloor \frac{N}{2} \right\rfloor \right) \right) \quad \text{as each log to the left of } \log\left(\left\lfloor \frac{N}{2} \right\rfloor \right) \text{ in the previous sum}
\]
\[= \left(\frac{N}{2} \right) \log\left(\frac{N}{2} \right)
= \left(\frac{N}{2} \right) (\log N - 1)
= \Omega(N \log N)
\]

**P2.**

MTU (Maximum Transfer Unit) defines the size of a packet that you can send to a target in Internet through a specific path without the packet getting fragmented into smaller packets. So, if you try to send a packet of size bigger than the MTU, you will get a message that your packet would be fragmented. This way, you can find the MTU of a specific path is by trying to send packets of increasing size until you get the fragmentation message.

Suppose you have to determine \( X \), which is the MTU of some path, and \( X \) could be potentially huge. Give an efficient algorithm to determine \( X \) and give its complexity in terms of \( X \). (Remember, you do not know the value of \( X \) beforehand).

**Ans.**

First try a packet of size 1. If you do not get any fragmentation message, then double your packet size and keep doing it till you get it (say for a packet size of \( P \)). Now, you know that \( X \) must lie between \( P \) and \( P/2 \) (the previous try for which you did not get any error message), so do a binary search. The overall complexity is \( O(\log X) \).

**P3.**

In a social gathering, there are \( b \) boys and \( g \) girls (\( b > g \)) of different ages. You have two unsorted arrays giving their ages (one for the boys, the other for the girls). Devise an efficient \( O(b \log g) \) algorithm to find out the ages that are common between both the boys and girls.

Example:
If \( \text{Array}_\text{boy} = \{10, 20, 11, 89, 23, 21\} \) and \( \text{Array}_\text{girl} = \{12, 30, 11, 20\} \),
Then  \( \text{Array}_{\text{common}} = \{11, 20\} \)

\textbf{Ans:}

Sort the smaller (girls’) array, then for each element in the larger array (boys), do a binary search on the girls’ array.