Date : __ / __ / ____

10th June

1. Maximum Meetings in One Room:

Approach (Greedy):
→ Sort meetings by their end times
→ select a meeting if its start time is after the last selected meeting's end time.

Steps: → Create a list of pairs (start[i], end[i]).
→ Sort the list by end[i]
→ Initialize count = 0, lastEndTime = 0.
→ Iterate : if start[i] > lastEndTime.
count++;
update LastEndTime = end[i];
→ Return count.

Java Code:

```
Public class MaxMeetings {
Public static int maxMeetings ( int n, int[] start, int[] end) {
    class Meeting {
      int start, end;
      Meeting ( int s, int e) {
       start = s;
       end = e; }
    }
    Meeting[] meetings = new Meeting [n];
   for(int i=0; i<n; i++) {
      meetings[i] = new Meeting ( start[i], end[i]);
   }
Arrays. sort (meetings, Comparator. comparingInt (m -> m.end))
```

Date : __/__/____

```
int count = 0;
int LastEndTime = 0;
for(int i=0; i<n; i++){
  if(meetings[i].start > lastEndTime){
    count++;
    lastEndTime = meetings[i].end;
  }
}
return count;
}
}
```

Dry Run:

Meeting 1: 1 > 0 → count = 1, lastEndTime = 2.
Meeting 2: 3 > 2 → count = 2, lastEndTime = 4.
Meeting 3: 0 > 4 ✗
Meeting 4: 5 > 4 → count = 3, lastEndTime = 7.
Meeting 5: 8 > 7 → count = 4, lastEndTime = 9.
Meeting 6: 5 > 9 ✗

Final count = 4.

Meeting	start	End
1	1	2
2	3	4
3	0	6
4	5	7
5	8	9
6	5	9

Date : __ / __ / ____

2. Number of Ways to Evaluate Expression to true.

Approach → Recursive + Memoization.
→ Break the string around each operator.
→ Recursively count ways for left and right parts to be True and False.
→ Combine based on operator.
→ Use memoization to avoid redundant computations.

Java Code:

```
static map< String, Integer > memo = new HashMap<>();
public static int countWays (string s, int i, int j, boolean isTrue)
    if (i > j) return 0;
    if (i == j) {
      if (isTrue)
        return s.charAT(i) == 'T' ? 1 : 0;
      else return s.charAT(i] == 'F' ? 1 : 0;
    }
string key = i + " " + j + " " + isTrue;
  if (memo.containskey (key))
     return memo.get(key);
     int ways = 0;
for (int k = i+1; k <= j-1; k+=2) {
  char op = S.charAT (k);
  int lt = countWays (S, i, k-1, true);
  int lf = countWays (s, i, k-1, false);
  int rt = CountWays (s, k+1, j, true);
  int rf = countWays (S, k+1, j, false);
```

Date : __/__/____

```
if (op == '&') {
    if (isTrue)
        ways += lt * rt;
    else
        ways += lt * rt + lf * rt + lf * rf;
} else if (op == '|') {
    if (isTrue)
        ways += lt * rt + lt * rf + lf * rt;
    else
        ways += lt * rt + lf * rf;
    }
}
memo.put(key, ways);
return ways;
}
}
```

Dry Run:

S = "T|F&T^T"

countWays (S, 0, 6, true)

Time complexity : $O(N^3)$

Space Complexity : $O(N^2)$