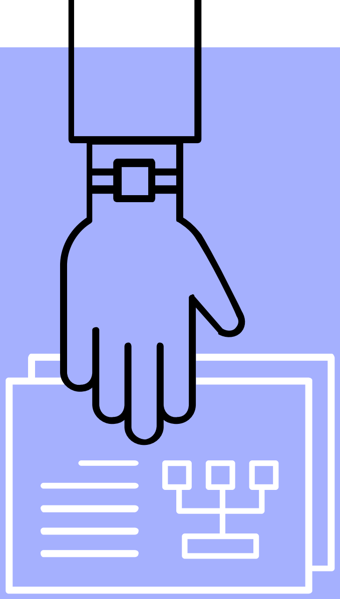
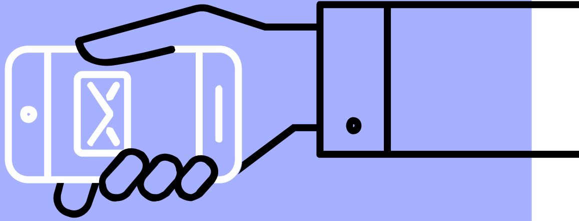
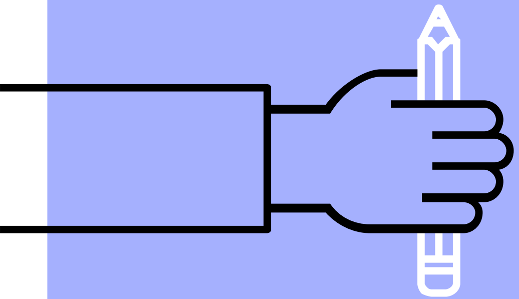
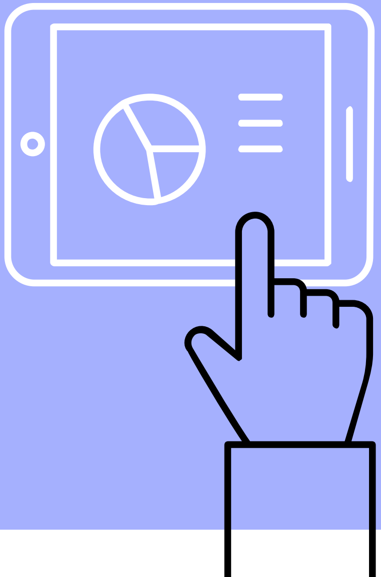
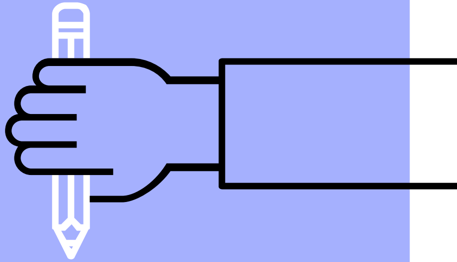



DAA ASSIGNMENT GROUP-14

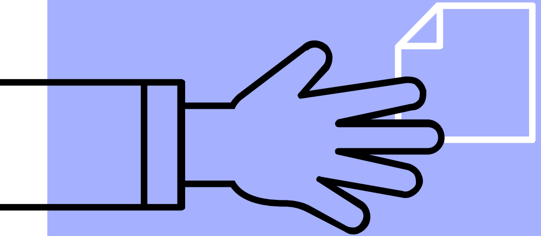
Group Members



Kishan Tripathi - IIT2019225

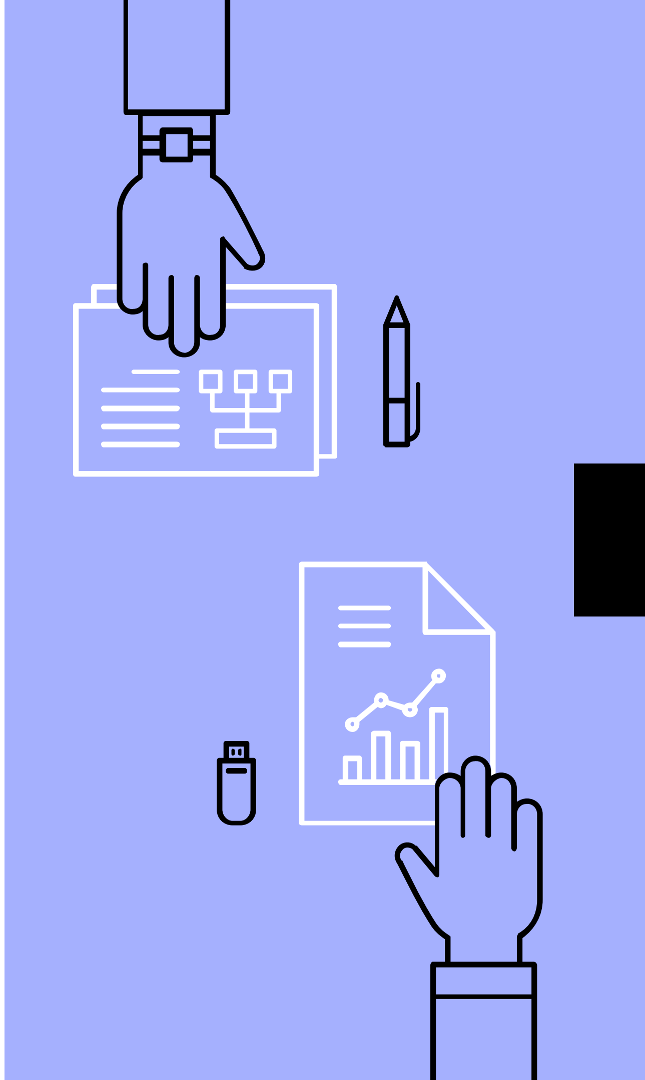
Mukul Mohmare - IIT2019226

Anshuman Bhradwaj - IIT2019227



Abstract :

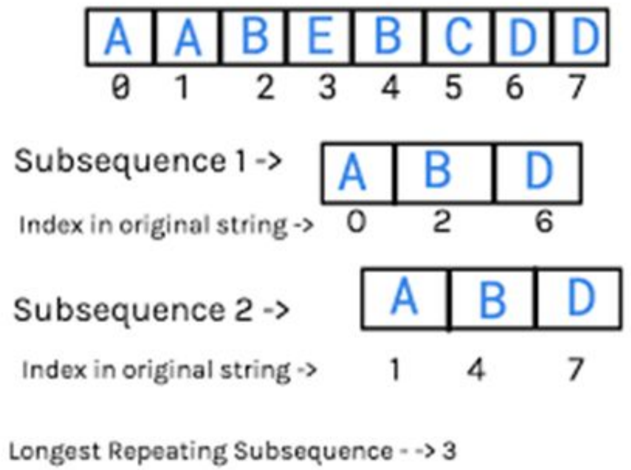
Given a string, print the longest repeating subsequence such that the two subsequences don't have same string character at same position, i.e., any j'th character in the two subsequences shouldn't have the same location index in the given original string. Solve using dynamic programming.



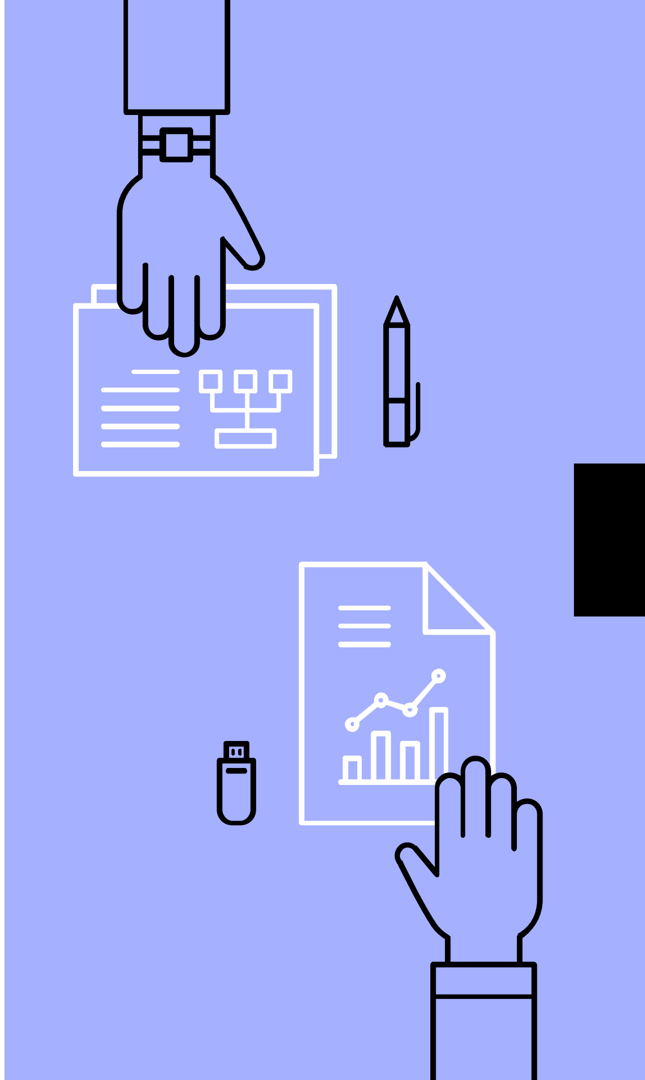
Longest Repeating Subsequence :

LRS is classic variation of LCS problem .The idea is to find LCS of the given string with itself but keeping in mind whenever characters match , they shouldn’t have same index.

Let's see an example……



A A B E B C D D
0 1 2 3 4 5 6 7
Subsequence 1 -> A B D
Index in original string -> 0 2 6
Subsequence 2 -> A B D
Index in original string -> 1 4 7
Longest Repeating Subsequence - -> 3



Dynamic Programming

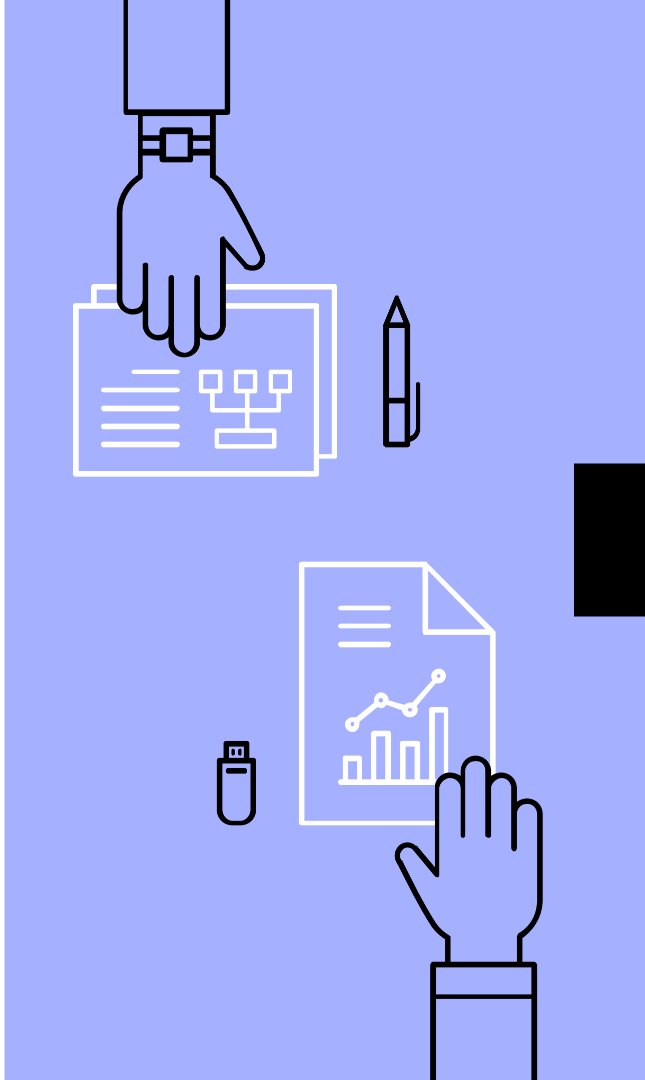
Dynamic Programming is optimisation over plain recursion. The idea is to simply store the results of subproblems, so that we do not have to re-compute them when needed later.

Any Dynamic Programming problem has two properties

1. Optimal Substructure
2. Overlapping

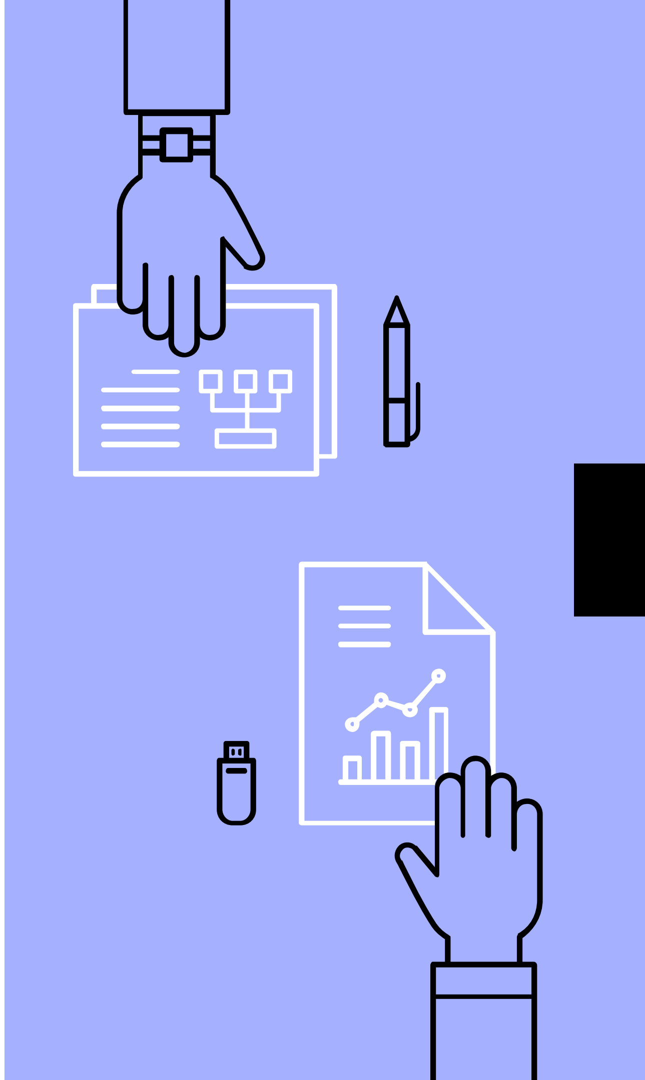
We used **Two** approaches to solve this problem.....

- ★ Bottom up approach
- ★ Top down approach



Bottom Up Approach

Analyze the problem and see the order in which the sub-problems are solved and start solving from the trivial subproblem, up towards the given problem.

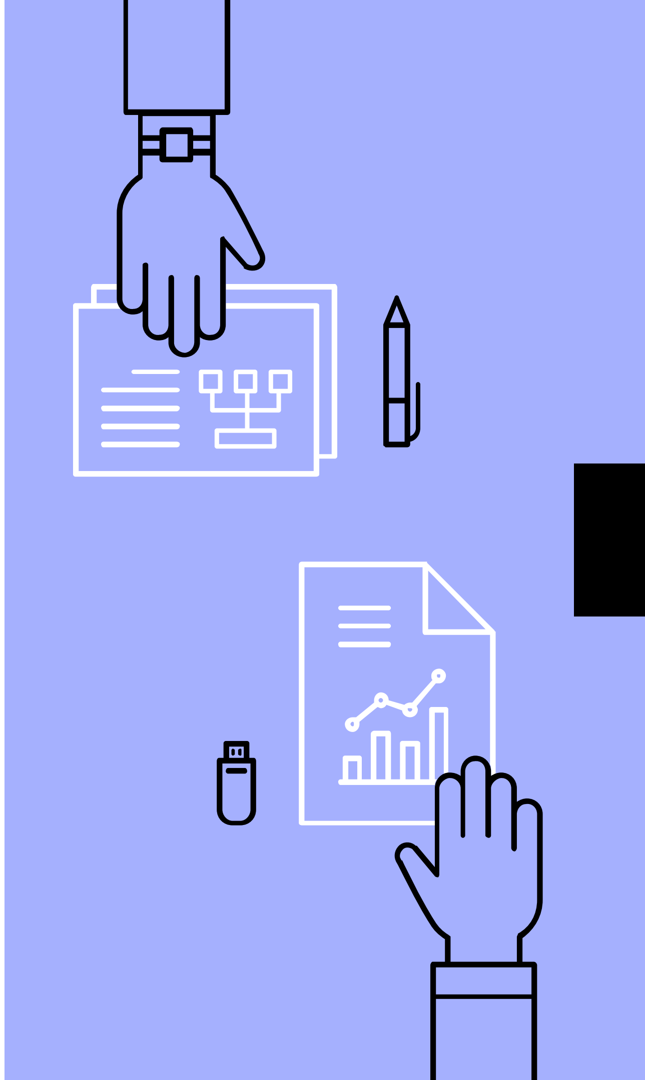


Logic Behind Bottom Up Approach :

We create a dp table of size [n+1][n+1]

Where n is the length of the string

```
    if (str[i-1] == str[j-1] && i != j)
            dp[i][j] =  1 + dp[i-1][j-1];
     else
            dp[i][j] = max(dp[i][j-1], dp[i-1][j]);
```



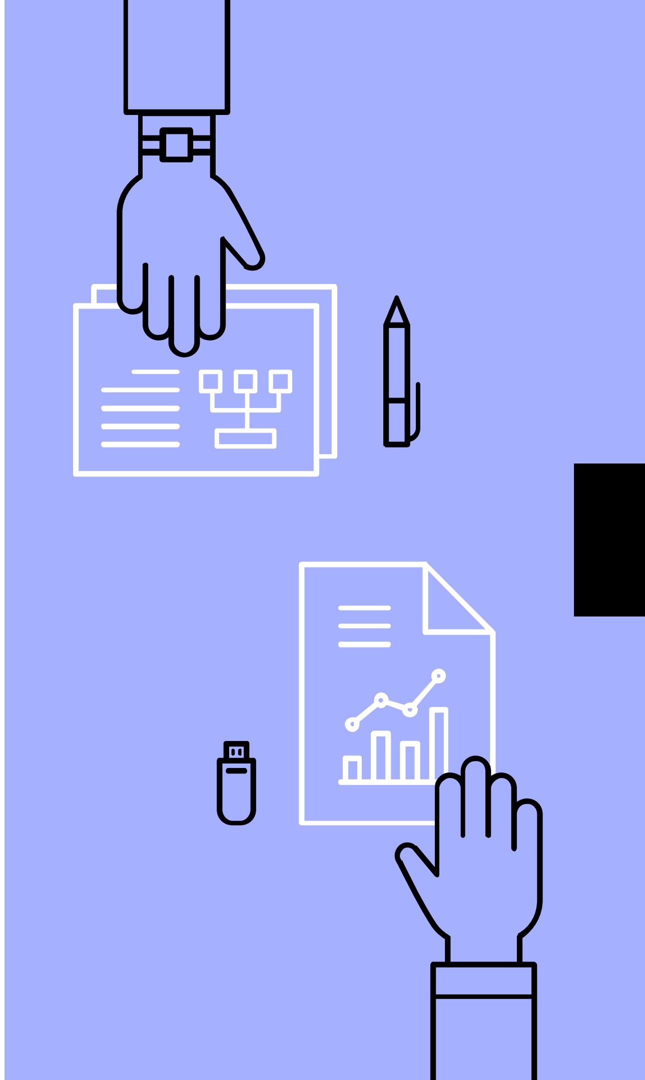
Top Down Approach

We solved the given problem by first breaking it down into simpler subproblems. If the problem is already solved then return the saved answer for that otherwise solve the subproblem and save the answer. This method is very intuitive.

Also referred as **Memoization**.

Logic Behind Top Down Approach :

```
        if (dp[n][m]!=0)
                return dp[n][m];
        if (n==0||m==0)
                return 0;
        else if (str[n-1]==str[m-1]&& n != m)
                return
dp[n][m]=1+longestRepeatedSubSeq(str,n-1,m-1);
        else
                return
dp[n][m]=max(longestRepeatedSubSeq(str,n-1,m),long
estRepeatedSubSeq(str,n,m-1));
```



Sample Input :

Input : “AABCBDC”

Output : “ABC”

A	A	B	C	B	D	C
0	1	2	3	4	5	6

First

A	B	C
0	2	3

Second

A	B	C
1	4	6

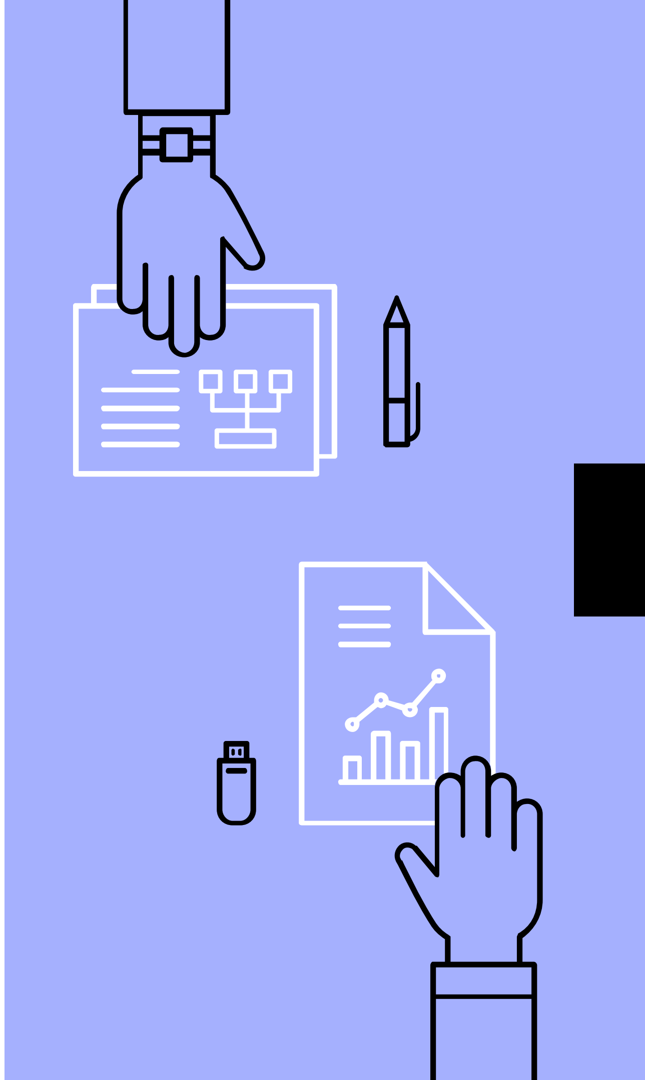
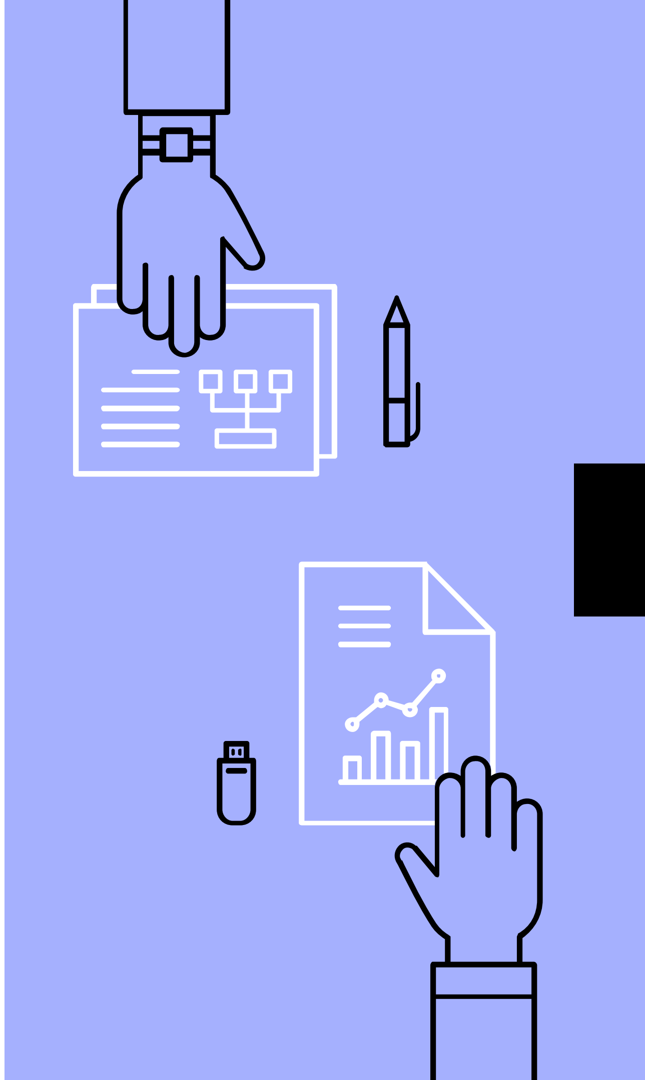


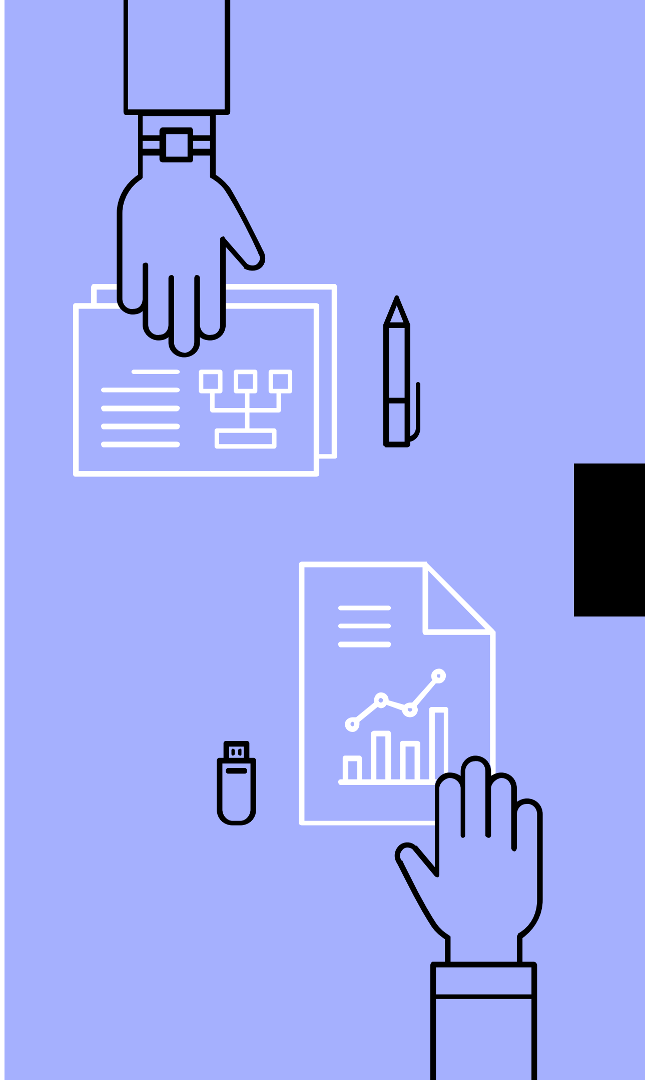
Table for storing answer...

		A	A	B	C	B	D	C
	0	0	0	0	0	0	0	0
A	0	0	1	1	1	1	1	1
A	0	1	1	1	1	1	1	1
B	0	1	1	1	1	2	2	2
C	0	1	1	1	1	2	3	3
B	0	1	1	2	2	2	3	3
D	0	1	1	2	3	3	3	3
C	0	1	1	2	3	3	3	3



Complexity Analysis

- Time complexity for both the algorithm is $O(n^2)$.
- Space complexity for both algorithms is$O(n^2)$.



THANK YOU