DP sheet

1> Max product subarray

```
int maxProduct(const vector<int> &A) {
    int N = A.size();
    if(N == 0) return 0;
    else if(N == 1) return A[0];
    else {
        int max_ending_here = A[0], min_ending_here = A[0];
        int max_overall = A[0];

        for(int i = 1; i<N; i++){
            int temp = max_ending_here;
            max_ending_here = max({A[i], max_ending_here*A[i],
min_ending_here*A[i]});
            min_ending_here = min({A[i], temp*A[i], min_ending_here*A[i]});
            max_overall = max(max_overall, max_ending_here);
        }
        return max_overall;
    }
}
```

2> Longest Increasing subsequence

```
int CeilIndex(std::vector<int>& v, int l, int r, int key)
{
    while (r - l > 1) {
        int m = l + (r - l) / 2;
        if (v[m] >= key)
            r = m;
        else
            l = m;
    }

    return r;
}



    int lengthOfLIS(vector<int>& v) {
```

```
        if (v.size() == 0)
        return 0;

    std::vector<int> tail(v.size(), 0);
    int length = 1; // always points empty slot in tail

    tail[0] = v[0];
    for (size_t i = 1; i < v.size(); i++) {


        if (v[i] < tail[0])
            tail[0] = v[i];


        else if (v[i] > tail[length - 1])
            tail[length++] = v[i];

        else
            tail[CeilIndex(tail, -1, length - 1, v[i])] = v[i];
    }

     return length;
    }
```

> Return LIS elements

```
vector<int> increasingTriplet(vector<int>& nums) {
       int n=nums.size();

    vector<int>tails;

    tails.push_back(nums[0]);

    for(int i=1;i<n;i++){

        if(tails.back()<nums[i]){
            tails.push_back(nums[i]);
        }

        else{
```

```
            int idx=lower_bound(tails.begin(),tails.end(),nums[i])-tails.begin();
            tails[idx]=nums[i];
        }

    }

    return tails;
}
```

3> Maximum sum of any increasing subsequence

```
int maxSumIS(int arr[], int n)
        {
            // Your code goes here

            int dp[n];
             dp[0]=arr[0];
             for(int i=1;i<n;i++)
             {
                 dp[i]=arr[i];
                 for(int j=0;j<i;j++)
                 {
                     if(arr[j]<arr[i])
                     {
                         dp[i]=max(dp[i], dp[j]+arr[i]);
                     }
                 }
             }
            int ans=dp[0];
            for(int i=1;i<n;i++)
              ans=max(ans, dp[i]);
            return ans;
        }
```

4> Longest common subsequence

```
int longestCommonSubsequence(string text1, string text2) {
        int n=text1.size(), m=text2.size();

        int dp[n+1][m+1];

        dp[0][0]=0;

        for(int i=1;i<=n;i++)
            dp[i][0]=0;
        for(int i=1;i<=m;i++)
            dp[0][i]=0;

        for(int i=1;i<=n;i++)
        {
            for(int j=1;j<=m;j++)
            {
                if(text1[i-1]==text2[j-1])
                    dp[i][j]=(1+dp[i-1][j-1]);
                else
                {
                    dp[i][j]=max(dp[i-1][j], dp[i][j-1]);
                }
            }
        }

        return dp[n][m];
    }
```

to print it   // X, Y are strings, m, n their length, L is dp table

```
  int index = L[m][n];


  char lcs[index+1];// it stores printed string
  lcs[index] = '\0';
  int i = m, j = n;
  while (i > 0 && j > 0)
  {

     if (X[i-1] == Y[j-1])
     {
         lcs[index-1] = X[i-1];
         i--; j--; index--;
```

```
      }

      else if (L[i-1][j] > L[i][j-1])
         i--;
      else
         j--;
   }


   cout << "LCS of " << X << " and " << Y << " is " << lcs;
}
```

5> Longest repeating subsequene

```
int LCS(string A, string B){
    int n=A.size();
    int T[n+1][n+1];
    for(int i=0;i<n+1;i++)
     {  T[i][0]=0;
       T[0][i]=0;
     }

    for(int i=1;i<n+1;i++)
    {
        for(int j=1;j<n+1;j++)
        {
            if(A[i-1]==B[j-1] && i!=j)
              T[i][j]=T[i-1][j-1]+1;
            else
              T[i][j]=max(T[i-1][j], T[i][j-1]);
        }
    }
    return T[n][n];

}

int anytwo(string A) {
    if(LCS(A,A) > 1)
      return 1;
      return 0;
```

```
}
```

6> 0-1 knapsack

```
int knapSack(int W, int wt[], int val[], int n)
    {
       // Your code here
       int dp[n+1][W+1];

       for(int i=0;i<=n;i++)
         dp[i][0]=0;
       for(int j=1;j<=W;j++)
         dp[0][j]=0;

       for(int i=1;i<=n;++i)
       {
           for(int j=1;j<=W;j++)
           {
               if(wt[i-1]>j)
               {
                   dp[i][j]=dp[i-1][j];
               }
               else
               {
                   dp[i][j]=max(dp[i-1][j], val[i-1]+dp[i-1][j-wt[i-1]]);
               }
           }
       }

       return dp[n][W];
    }
```

7> Unbounded knapsack

```
int knapSack(int n, int W, int val[], int wt[])
    {
        // code here

       // Your code here
       int dp[n+1][W+1];

       for(int i=0;i<=n;i++)
         dp[i][0]=0;
       for(int j=1;j<=W;j++)
         dp[0][j]=0;

       for(int i=1;i<=n;++i)
       {
           for(int j=1;j<=W;j++)
           {
               if(wt[i-1]>j)
               {
                   dp[i][j]=dp[i-1][j];
               }
               else
               {
                   dp[i][j]=max(dp[i-1][j], val[i-1]+dp[i][j-wt[i-1]]);
               }
           }
       }

       return dp[n][W];

    }
```

8> Minimum cost to get a given total weight

```
int minimumCost(int cost[], int n, int W)
```

```
        {

    vector<int> val, wt;

    int size = 0;
    for (int i=0; i<n && i<W; i++)
    {
        if (cost[i]!= -1)
        {
            val.push_back(cost[i]);
            wt.push_back(i+1);
            size++;
        }
    }

    n = size;
    int min_cost[n+1][W+1];


    for (int i=0; i<=W; i++)
        min_cost[0][i] = MAX_INT;


    for (int i=1; i<=n; i++)
        min_cost[i][0] = 0;


    for (int i=1; i<=n; i++)
    {
        for (int j=1; j<=W; j++)
        {
            if (wt[i-1] > j)
                min_cost[i][j] = min_cost[i-1][j];


            else
                min_cost[i][j] = min(min_cost[i-1][j],
                     min_cost[i][j-wt[i-1]] + val[i-1]);

            //cout<<min_cost[i][j]<<" ";
        }
    }


    return (min_cost[n][W]==MAX_INT)? -1: min_cost[n][W];
}
```

9> Number of ways of getting a sum with repetiton allowed and different arrangements

```
int countWays(int arr[], int m, int N)
{
    int count[N + 1];
    memset(count, 0, sizeof(count));

    count[0] = 1;


    for (int i = 1; i <= N; i++)
        for (int j = 0; j < m; j++)

              if (i >= arr[j])
                count[i] += count[i - arr[j]];


    return count[N];

}
```

10> minimum no. of coins required to get a sum

```
#define ixt int_fast32_t


    int coinChange(vector<int>& c, int t) {
        sort(c.begin(),c.end());
            ixt *dp=new ixt[t+1];
        const int n=c.size();
        dp[0]=0;for(int i=1;i<=t;dp[i++]=1e9);
            for(int i=1;i<=t;i++){
                     for(int j=0,reg=0;j<n;j++){
                        reg=i-c[j]; // REG to optimize the exec time
                         if(reg>=0){
                                       dp[i]=min(dp[i],dp[reg]+1); // DP
```

```
                        } else {
                          continue;
            }
                }
        }

        int ans=dp[t];
        delete[] dp;
        if(ans>1e8){
                return -1;
        }
        return ans;
    }
```

9> subset with a sum possible or not

```
bool subsetsum(vector<int>& nums, int sum){
   bool dp[nums.size()+1][sum+1];

    dp[0][0]=1;

    for(int i=1;i<nums.size()+1;++i)
        dp[i][0]=1;
    for(int i=1;i<sum+1;i++)
        dp[0][i]=0;

    for(int i=1;i<nums.size()+1;++i)
    {
        for(int j=1;j<=sum;++j)
        {
            if(nums[i-1]<=j)
               dp[i][j]=dp[i-1][j] || dp[i-1][j-nums[i-1]];
            else
                dp[i][j]=dp[i-1][j];
        }
    }
    return dp[nums.size()][sum];
}
```

11> Minimum insertion, deletion or replace to convert string to another

```
    int minDistance(string A, string B) {
        int n=A.length();
int m=B.length();

int dp[n+1][m+1];

for(int i=0;i<=n;i++)
{
    for(int j=0;j<=m;j++)
    {
          if(i==0)
            {
            dp[i][j]=j;
            }
            else if(j==0)
            {
                dp[i][j]=i;
            }
        else if(A[i-1]==B[j-1])
        {
            dp[i][j]=dp[i-1][j-1];
        }
        else
        {
            dp[i][j]=1+min(dp[i-1][j],min(dp[i][j-1],dp[i-1][j-1]));
        }
    }
}
return dp[n][m];
    }
```

12> Matrix chain multiply

```
int dp[101][101]; //101 is max size of array, given in question


int matrixChainMemoised(int arr[], int i, int j)
```

```
{
    if (i == j)
    {
        return 0;
    }
    if (dp[i][j] != -1)
    {
        return dp[i][j];
    }
    dp[i][j] = INT_MAX;
    for (int k = i; k < j; k++)
    {
        dp[i][j] = min(
            dp[i][j], matrixChainMemoised(arr, i, k)
                     + matrixChainMemoised(arr, k + 1, j)
                       + arr[i - 1] * arr[k] * arr[j]);
    }
    return dp[i][j];
}
```

```
    int matrixMultiplication(int n, int arr[])
    {

        memset(dp,-1, sizeof(dp));
         int i = 1, j = n - 1;
       return matrixChainMemoised(arr, i, j);
    }
};
```

13> Rod cutting minimum cost

```
    int solve (vector<int> & cuts, int i, int j, int start, int end,
vector<vector<int>> & dp){

        if(start>end) return 0;

        if(dp[start][end]!=-1)
            return dp[start][end];
```

```
        int ans = INT_MAX;

        for(int k=start; k<=end; k++){

            int temp = solve(cuts, i, cuts[k], start, k-1, dp) + solve(cuts,
cuts[k], j, k+1, end, dp) + j-i;
            ans = min(ans, temp);
        }

        return dp[start][end] = ans;
    }


    int minCost(int n, vector<int>& cuts) {
        int len = cuts.size();


        sort(cuts.begin(), cuts.end());

        vector<vector<int>> dp (len, vector<int> (len, -1));
        return solve (cuts, 0, n, 0, len-1, dp);
    }
```

14> Egg drop

```
int T[101][10001];//100 is max no. of eggs, 10000 is max floors

int find(int A, int B){
    if(T[A][B] != -1)
       return T[A][B];
    if(B==0)
      {T[A][B]=0; return 0;}
    if(A==1 || B==1)
      { T[A][B]=B;
      return B;}
    int min=INT_MAX;
    for(int i=1;i<=B;i++)
    {
        int left,right;
```

```
        if(T[A-1][i-1] != -1)
           left=T[A-1][i-1];
        else
           left=find(A-1,i-1);

        if(T[A][B-i] != -1)
           right=T[A][B-i];
        else
           right=find(A,B-i);


        int mn=1+max(left,right);

        if(mn < min)
          min=mn;
    }
    T[A][B]=min;
    return min;
}

int Solutionsolved(int A, int B) {
    memset(T,-1,sizeof(T));
    //int min=INT_MAX;
    return find(A,B);
}

int solve(int k, int n) {
   vector<int> dp(k + 1, 0);
        int m;
        for( m=0;dp[k]<n;m++){
            for(int egg=k;egg>0;egg--)
                dp[egg]= dp[egg-1]+ dp[egg]+1;
        }
        return m;
}
```

15> all ways to break a string such that all parts are palindrome

```
bool isPalin(string s, int start, int end){
```

```
    while(start <= end){
        if(s[start++]!=s[end--])
            return 0;
    }
    return 1;
}

void func(int index,string s, vector<string> &path, vector<vector<string>> &res){
    if(index==s.size())
    {
        res.push_back(path);
        return ;
    }
    for(int i=index; i<s.size(); ++i){
        if(isPalin(s,index,i)){
            path.push_back(s.substr(index,i-index+1));
            func(i+1, s, path, res);
            path.pop_back();
        }
    }
    return ;
}


    vector<vector<string>> partition(string s) {
       vector<vector<string>> res;
        vector<string> path;
        func(0,s,path,res);
        return res;
    }
```

16> Job scheduling when endtime, starttime given, start of one can be end of other

```
struct Interval
{
    int start;
    int end;
    int profit;
    int maxProfit;  //the max profit we can reach with this task.
```

```
    bool operator<(const Interval &a) const  // sort the Interval with the end time
    {
        return this->end < a.end;
    };
};



    int jobScheduling(vector<int>& startTime, vector<int>& endTime, vector<int>&
profit) {
        vector<Interval> intervals;
        intervals.push_back({-1,-1,0,0});  // dummy start
        for(int i = 0 ; i < startTime.size() ; i++)
        {
            intervals.push_back({startTime[i], endTime[i], profit[i], -1});
        }

                //sort the interval based on the end time
        sort(intervals.begin(), intervals.end());

        for(int i = 1  ; i < intervals.size(); i++)
        {
            Interval dummy = {0, intervals[i].start,0,0};
            auto it = upper_bound(intervals.begin(), intervals.end(), dummy);
            int subProfit =prev(it)->maxProfit;
                        // do DP
            intervals[i].maxProfit = max(intervals[i-1].maxProfit, subProfit +
intervals[i].profit);
        }
        return intervals.back().maxProfit;
    }
```

17> Words from dictionary

```
int T[6501];
int rec(string &A,int l,unordered_set<string> &m)
{
    if(T[l]!=-1)
      return T[l];
    int n=A.size();
    if(l==n)
```

```
   {
    T[l]=1;
    return 1;
   }
   string s;
  for(int i=l;i<n;i++)
   {
    if(i-l>=20)
     {
       T[l]=0;
       return 0;
     }
     s.push_back(A[i]);
    if(m.find(s)!=m.end())
    {

   if(rec(A,i+1,m))
      {   T[l]=1;
          return 1;
      }
   }
  }
  T[l]=0;
  return 0;
}

    bool wordBreak(string A, vector<string>& B) {
       memset(T,-1,sizeof(T));

    unordered_set<string> m;

     for(auto x:B)
       m.insert(x);
   return rec(A,0,m);
    }
```

> Max rectangle with all 1

```
// Rows and columns in input matrix
int R,C;
```

```
int maxHist(vector<int> row)
{

    stack<int> result;

    int top_val;

    int max_area = 0;
    int area = 0;


    int i = 0;
    while (i < C) {

        if (result.empty() || row[result.top()] <= row[i])
            result.push(i++);

        else {

            top_val = row[result.top()];
            result.pop();
            area = top_val * i;

            if (!result.empty())
                area = top_val * (i - result.top() - 1);
            max_area = max(area, max_area);
        }
    }


    while (!result.empty()) {
        top_val = row[result.top()];
        result.pop();
        area = top_val * i;
        if (!result.empty())
            area = top_val * (i - result.top() - 1);

        max_area = max(area, max_area);
    }
    return max_area;
}
 int maximal(vector<vector<int> > &A) {

    int result = maxHist(A[0]);

    for (int i = 1; i < R; i++) {

        for (int j = 0; j < C; j++)
```

```
            if (A[i][j])
                A[i][j] += A[i - 1][j];


        result = max(result, maxHist(A[i]));
    }

    return result;

}
```

> Maximal square in binary matrix

```
 int maximalSquare(vector<vector<char>>& matrix) {
        int row = matrix.size(); int col = matrix[0].size();
        vector<vector<int>> dp(row+1, vector<int>(col+1,0));
        int maxSquare = 0;
        for(int i=1; i<=row; i++)
        {
            for(int j=1; j<=col; j++)
            {
                if(matrix[i-1][j-1] == '1')
                {
                    dp[i][j]= 1 + min({dp[i-1][j-1], dp[i-1][j], dp[i][j-1]}) ;
                    maxSquare = max(maxSquare, dp[i][j]);
                }
            }
        }
        return maxSquare*maxSquare;
    }
```

> Scramble string

```
class Solution {
public:

    map<pair<string,string>,bool> mp;

    bool check(string a, string b){

        if(mp.find({a,b})!=mp.end()) return mp[{a,b}];

        if(a==b) return mp[{a,b}]=true;
        if(a.length()<=1 || b.length()<=1) return mp[{a,b}]=false;\
        string c_a=a,c_b=b;
        sort(c_a.begin(),c_a.end());
        sort(c_b.begin(),c_b.end());
        if(c_a!=c_b) return mp[{a,b}]=false;

        int n=a.length();
        for(int i=1; i< a.length(); i++){
            if( check(a.substr(0,i),b.substr(0,i)) &&
check(a.substr(i,n-i),b.substr(i,n-i)) ){
                return true;
                break;
             }
            if( check(a.substr(0,i),b.substr(n-i,i)) &&
check(a.substr(i,n-i),b.substr(0,n-i))) {
                return true;
                break;
             }
        }
        return mp[{a,b}]=false;

    }

    bool isScramble(string s1, string s2) {
        if(s1.length()!=s2.length()) return false;
        return check(s1,s2);
    }
};
```