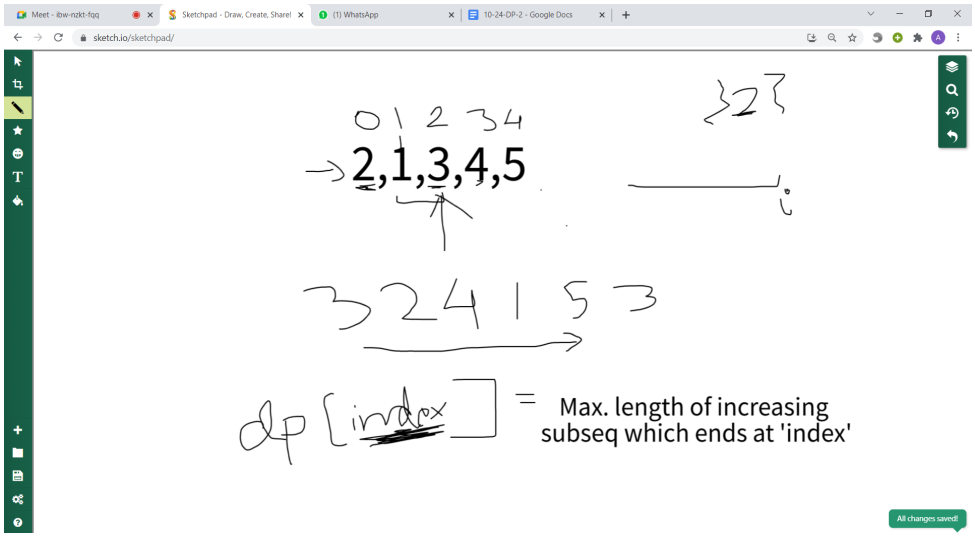
Longest Increasing Subsequence



2,13,3,4,5

1....(i-1) i

------>
dp[i]= max(1,1+dp[j]) where aj<=ai and j<i

O(n*n)

dp[0]=1;
dp[1]=1
dp[2]=2
dp[3]=3
dp[4]=4

Code:

```
#include<bits/stdc++.h>
using namespace std;
// 12 10 9 1
 //  0  1  2 3
int LIS(vector<int> v){
  int n=v.size();
  int dp[n+1];

  for(int i=0;i<n;i++){
```

```
        dp[i]=1;
        for(int j=i-1;j>=0;j--){
            if(v[i]>=v[j])
            dp[i]=max(dp[i],1+dp[j]);
        }
    }
    int ans=*max_element(dp,dp+n);
    return ans;
}

int main(){
    int n;
    cin>>n;
    vector<int>v(n);

    for(int i=0;i<n;i++)
    cin>>v[i];

    cout<<LIS(v);

    return 0;
}
```

Time Complexity: **O(n^2)**

Longest Strictly Increasing Subsequence:

Hint: Just change v[i]>=v[j] to v[i]>v[j] in LIS function.

Longest Increasing Subarray/Substring:

Note: A substring needs to be continuous.

```
#include<bits/stdc++.h>
using namespace std;
// 12 10 9 1
 //  0  1  2 3
int LIS(vector<int> v){
    int n=v.size();
    int dp[n+1];

    for(int i=0;i<n;i++){
```

```
      dp[i]=1;
      if(i>0&&v[i]>=v[i-1])
      dp[i]=1+dp[i-1];
  }
  int ans=*max_element(dp,dp+n);
  return ans;
}

int main(){
    int n;
    cin>>n;
    vector<int>v(n);

    for(int i=0;i<n;i++)
    cin>>v[i];

    cout<<LIS(v);

    return 0;
}
```

Time Complexity: O(n)
Note: Can be solved with two pointers as well in linear time.

Longest Common Subsequence (LCS)

LCS:

dp[i][j]:

// base case

if(a[i]==b[j])
dp[i][j]=1+dp[i-1][j-1] // dp[1][0]=1
else
dp[i][j]=max(dp[i][j-1],dp[i-1][j]); //

Code:

```cpp
#include<bits/stdc++.h>
using namespace std;

int LCS(vector<int> a,vector<int> b){
 int n=a.size();
 int m=b.size();

 int dp[n+1][m+1];  // 1 based indexing

 // int dp[n];  dp[0] -> [0......n-1]
 // int dp[n+1] -> 0.1,2...n

 dp[0][0]=0;

  // a: 1...i
  // b: empty
 for(int i=1;i<=n;i++){
     dp[i][0]=0;
 }

 // a: empty
 // b: 1...j
 for(int j=1;j<=m;j++)
 dp[0][j]=0;

 for(int i=1;i<=n;i++){
     for(int j=1;j<=m;j++){
         if(a[i-1]==b[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];
}

int main(){
    int n;
```

```
    cin>>n;

    vector<int>a(n);

    for(int i=0;i<n;i++)
    cin>>a[i];

    int m;
    cin>>m;

    vector<int>b(m);

    for(int i=0;i<m;i++)
    cin>>b[i];

    cout<<LCS(a,b);
    return 0;
}
```

Time Complexity: O(n*m)

Coin change

Watch this video is you have doubt ->
https://www.youtube.com/watch?v=ZI17bgz07EE

```
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

ll dp[1005][1005];

ll get_length(ll amount, ll pos, vector<ll> &coins) {
```

```
    if(pos >= coins.size() || amount<0){
        if(amount==0){
            return 0;
        }
        return INT_MAX; // we will not be counting invalid case this way as answer will
//always be less than infinity
    }

    if(dp[amount][pos]!=-1){
        return dp[amount][pos];
    }

    // unvisited state-> -1
    // visited state -> 0-infinity

    ll ans= min({get_length(amount, pos+1, coins),
                1+get_length(amount-coins[pos], pos, coins),
                   1+get_length(amount-coins[pos], pos+1, coins)});

    // three choices at every state
    // 1) we don't take the coin and move to next postion
    // 2) we take the coin and don't move to next postion
    // 3) we take the coin and move to next postion

    // dp[amount][pos]=ans;
    // return dp[amount][pos];
    return dp[amount][pos]=ans;
}

int main(){
    memset(dp,-1,sizeof(dp));// making every element of dp array -1

    ll amount,n; cin>>amount>>n;
    vector<ll>coins(n);
    for(ll i=0;i<n;i++){
        cin>>coins[i];
```

```
    }
    cout<<get_length(amount,0,coins);
}
```

Complexity-> amount * size of coin's array

Home work problem-

https://www.hackerrank.com/challenges/ctci-coin-change/problem?h_r=internal-search