Sum of substrings of a number:

sum(123) = 1+2+3+12+23+123 = 164.

- for XY = 11X+2Y
- for XYZ = 111X+22Y+3Z
- for WXYZ = 1111W+222X+33Y+4Z

```
long sum(char s[])
{
    long sum=0, mul=1;
    for(i=strlen(s); i>0; i--)
    {
        sum+= (s[i-1]-'0')*mul*i;
        mul = mul*10+1;
    }
    return sum;
}
```

* for any number multiple of 4 have least two significant bits as zero.
* similarly for multiple of 8 we have least three " " "

```
bool isMultipleOf4(long long n)
{
    if(n==0)
        return 0;
    return ((n>>2)<<2)==n;
}
```

XOR from 1 to n

(a) find remainder of n by modding it with 4

(i) If rem = 0, xor will be same as n
(ii) If rem = 1, xor will be 1
(iii) If rem = 2, xor will be n+1
(iv) If rem = 3, xor will be 0

7-08-19

Euclidean algorithm for finding GCD

$$gcd(a,b) = \begin{cases} b & \text{if } a=0 \\ gcd(b\%a, a) & \text{otherwise} \end{cases}$$

Extended Euclidean Algorithm

The extended version also finds a way to represent GCD interms of a and b, (i.e) coefficient of x & y

$$a \cdot x + b \cdot y = gcd(a,b)$$

All we need is to figure out how the coefficients x & y change during the transition from (a,b) to $(b \bmod a, a)$

let us assume coefficients (x_1, y_1) for $(b \bmod a, a)$

$$(b \bmod a)\, x_1 + a \cdot y_1 = g \rightarrow ①$$

and we want to find pair (x,y) to (a,b)

$$a \cdot x + b \cdot y = g \rightarrow ②$$

we can represent $b \bmod a$ as

$$b \bmod a = b - \left\lfloor \frac{b}{a} \right\rfloor \cdot a$$

(floor)

substitute this eq ① gives us

$$g = (b \bmod a)\, x_1 + a \cdot y_1$$

$$= \left(b - \left\lfloor \frac{b}{a} \right\rfloor \cdot a\right) x_1 + a y_1$$

after rearraing terms

$$g = b x_1 + a\left(y_1 - \left\lfloor \frac{b}{a} \right\rfloor \cdot x_1\right)$$

We found values of x and y

$$x = y_1 - \left\lfloor \frac{b}{a} \right\rfloor x_1$$

$$y = x_1.$$

Implementation:

```
int gcd(int a, int b, int &x, int &y)
{
    if(a==0)
    { x=0;
      y=1;
      return b;
    }
    int x1, y1;
    int d = gcd(b%a, a, x1, y1);
    x = y1 - (b/a) * x1;
    y = x1;
    return d;
}
```

modular inverse by extended Euclidean

```
int x,y
int g = gcd(a,m,x,y)
if(g!=1)
    cout<< "NO solution";

else
{   x= (x%m+m)%m; // x may be negative, inorder to avoid negative we add +m.

    cout<<x<<endl;
}
```

26/08/19

Subset sum by dp:

```
bool isSubsetSum(int set[], int n, int sum)
{
    bool subset[n+1][sum+1];

    for (int i=0; i<=n; i++)
        subset[i][0]=true;

    for(int i=1; i<=sum; i++)
        subset[0][i]=false;

    for (int i=1; i<=n; i++)
    {
        for (int j=1; j<=sum; j++)
        {
            if (j<set[i-1])
                subset[i][j] = subset[i-1][j];
            if (j>=set[i-1])
                subset[i][j] = subset[i-1][j] ||
                               subset[i-1][j-set[i-1]];
        }
    }

    return subset[n][sum];
}

int main()
{
    int set[] = {3,34,4,12,5,2};
    int sum=9;
    int n= sizeof(set)/sizeof(set[0]);
    if (isSubsetSum(set,n,sum)==true)
        cout << "subset found";
    else cout<<"subset not found";
}
```

5/9/19

Next greater number with same set of digits

ex: 218765

output: 251678

algorithm:

1) from end of array traverse till arr[i-1] < arr[i] is reached
2) if found then from with i-1 traverse from end of array till arr[i-1] < arr[end] (then break)
3) now swap (end, i-1) elements, then reverse segment [end+1, n) (size of array)
4) print array

14/9/19

Tag: DP

minimum time to finish task without skipping two consecutive

Ex:
Ip: 10, 5, 7, 10
Op: 12 (skip first & last task)

```
int minTime (int arr[], int n)
{
    if(n<=0) return 0;
    int incl = arr[0];
    int excl = 0;
    for (i=1; i<n; i++)
    {
        int incl_new = arr[i] + min(incl, excl);
        int excl_new = incl; // when next task is not included current task must be included,
        incl = incl_new;
        excl = excl_new;
    }
    return min(incl, excl);
}
```

Ex:

24-09-2019

Longest incresing subsequence (LIS)

ex: arr = [3, 10, 2, 1, 20] (3, 10, 20)

length of LIS = 3

```
int lis(int arr[], int n)
{   int lis[n];
    lis[0]=1;
    for(i=1;i<n;i++)
    {   lis[i]=1;
        for(j=0;j<i;j++)
        {   if(arr[i]>arr[j] && lis[i]<lis[j]+1)
                lis[i] = lis[j]+1;
                sol[i]=j; // for printing LIS
        }
    }
    return *max_element(lis, lis+n);
    //returns max of array lis.
}
```

TO print LIS start with index which has max value in lis[] array and move back

for printing LIS

find max index in lis let it be "index" then

initialize sol[] with -1

```
ans.push_back(index);
for(i=index; i>=0; )
{   ans.push_back(sol[i]);
    i = sol[i];
}
for(i=ans.size()-1; i>=0; i--)
    cout << str[ans[i]] << " ";
```

25-09-19

cutting rod to maximize profit.

ex:	len	1	2	3	4	5	6	7	8
	price	1	5	8	9	10	17	17	20

output is 22

length 6 + length 2

let prices be the array, dp be solution array

```
for(i=1; i<=n; i++)
    dp[i]=prices[i];
```

```
dp[0] = 0;    dp[1] = prices[1];

for(k=2; k<=n; k++)
{
    for(i=1; i<=k; i++)
        dp[k] = max(dp[k], dp[i]+dp[i-k]);
}
cout<<dp[n];
```

25/10/19

Edit distance: minimum cost to convert str1 to str2 allowed operations are

(i) insert
(ii) remove
(iii) replace.

any operation cost is 1.

insert →

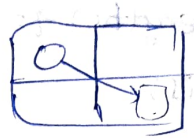
↓ Remove.

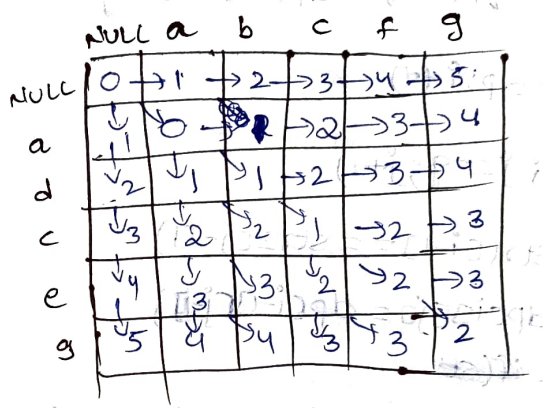
if (r≠c)

Replace ↘	Remove ↓
insert →	min(Replace, insert, Remove) +1

- If r==c

just copy diagonal element





```
long int minimum_edit_distance(string str1, string str2)
{
    long int i, j, p = str1.length(), q = str2.length();
    long int dp[p+1][q+1];

    for(i=0; i<=p; i++)  dp[i][0] = i;
    for(i=0; i<=q; i++)  dp[0][i] = i;

    for(i=1; i<=p; i++)
    {
        for(j=1; j<=q; j++)
        {
            if(str1[i-1] == str2[j-1])
                dp[i][j] = dp[i-1][j-1];
            else
                dp[i][j] = minimum(dp[i][j-1],
                dp[i-1][j-1], dp[i-1][j]) + 1;
        }
    }

    return dp[p][q];
}
```

26-10-19

Longest palindromic subsequence

This problem is similar to longest common subsequence (LCS), let str1 be given string, make str2 by reversing the given string then find LCS

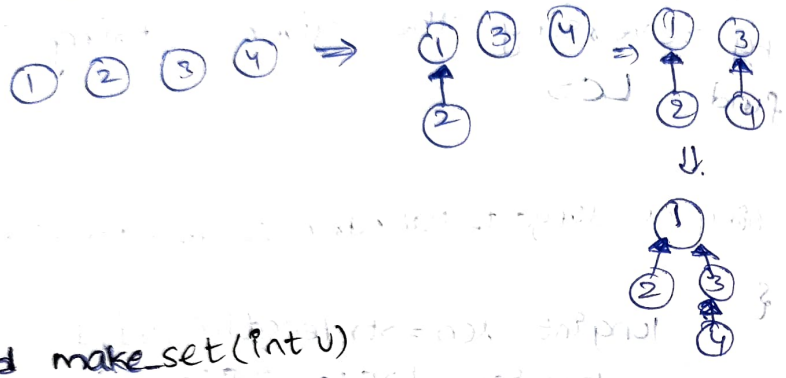
```
long int longest_palindromic_subsequence(string str)
{
    long int len = str.length(), i, j;
    long int dp[len+1][len+1];
    string rev = str;
    reverse(rev.begin(), rev.end());

    for(i=0; i<=len; i++)  dp[i][0] = 0;
    for(i=0; i<=len; i++)  dp[0][i] = 0;

    for(i=1; i<=len; i++)   // code fo LCS
    {
        for(j=1; j<=len; j++)
        {
            if(str[i-1] == rev[j-1])
                dp[i][j] = dp[i-1][j-1] + 1;
            else
                dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
        }
    }

    return dp[len][len];
}
```

12-11-19

refer CP algorithms

Disjoint ~~Union~~ set union:

At beginning every element starts as single element set



```
void make_set(int v)
{
    parent[v] = v;
    size[v] = 1;        // for weighted union
}

int find_set(int v)
{
    if(v == parent[v])
        return v;
    return parent[v] = find_set(parent[v]);
}
```

This is for path compression

```
void union_sets(int a, int b)
{
    a = find_set(a);
    b = find_set(b);
    if(a != b)
    {
        if(size(a) < size(b))
            swap(a,b);
        parent[b] = a;
        size[a] += size[b];
    }
}
```

; attaching lower weight tree to higher weight

* max sum of subsequence such that no ~~two~~ two adjacent elements are taken (Stackler thief)

$dp[i] = max(dp[i-2] + arr[i], dp[i-1])$

$dp[i-2] + arr[i]$ is when thief decised to rob house i, then we take profit till i-2, then add current profit (arr[i]),

dp[i-1] is case, when thief decided to not to stole the house.

```
int maxprofit(int arr[],int n)
{
    int dp[n];
    dp[0] = arr[0];  dp[1]= max(arr[0],arr[1]);
    for(i=2;i<n;i++)
        dp[i] = max(arr[i]+dp[i-2],dp[i-1]);
    return  dp[n-1];
}
```

28-11-19

partition problem DP-18

determine whether given set can be partitioned into two subsets such that sum of elements in both subsets is same,

* Find sum of elements in set, (named as sum) if sum is odd no solution exists
  else solution exists,
  So, try to find subset with sum/2
  If found return yes else no,
* It is similar to finding whether a subset with sum/2 exists or not by passing as argument to below function

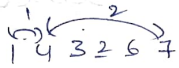
```
bool isSubsetSum( int  set[],int n,int sum)
{
    bool subset[n+1][sum+1];
    for( i=0;i<=n;i++)
        subset[i][0]= true;
    for(i=0; i<=sum;i++)
        subset[0][i]= false;
    subset[0][0]= true;
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=sum;j++)
        {
            if(j>=set[i-1])
                subset[i][j]=subset[i-1][j] ||
                    subset[i-1][j-set[i-1]];
            else
                subset[i][j] =subset[i-1][j];
        }
    }
return subset[n][sum];
}
```

7-12-19

minimum no. of jumps to reach end

Given an array where each element represents the max number of steps that can be made forward from that element, find minimum number of jumps to reach end of array, if an element is 0 then cannot move through that element.

Ex: 6
1 4 3 2 6 7



op: 2

```
int solve(int arr[],int n)
{
    if(n<=1) return 0;
    if(arr[0]==0) return -1;
    int max_index, jumps, steps;
      max_index = steps = arr[0];
       jumps=1;
    for(i=1; i<n; i++)
    {
        if(i==n-1) return jumps;
        max_index = max(max_index, i+arr[i]);
         steps--;
        if(steps==0)
        {   jumps++;
            if(i>=max_index)
                return -1;
            steps= max_index-i;
        }
    }
    return jumps;
}
```

11-12-19

frequency of array elements (1....N) in O(1) extra space and O(n) time,

$1 \le A_i \le N$

```
void printfrequency(int arr[],int n)
{
    for(j=0;j<n;j++)
        arr[j]=arr[j]-1;  // to make sure all elements are in (0,n-1)

    for(i=0;i<n;i++)
      arr[arr[i]%n] = arr[arr[i]%n] +n

    for(i=0;i<n;i++)
      cout << arr[i]/n << "\n";
}
```

19-12-19

lower_bound & upper_bound :

lower_bound(vec.begin(), vec.end(), num)
upper_bound(vec.begin(), vec.end(), num)

lower_bound:

(i) Single occurence of num returns pointer to position of num.
(ii) multiple occurence of num returns first position of num
(iii) no occurence of num returns position of next higher number than num

Subtracting the pointer by vec.begin() returns actual index (int index=itr-vec.begin())

upper_bound:

(i) Single occurence of num returns pointer to position of next higher number than num
(ii) multiple occurence of num. Returns pointer to first position of next higher number than last occurence of num,
(iii) no occurence of num returns pointer to position of next higher number than num.

18-12-19

0/1 Knapsack problem

```
int Knapsack(int W, int wt[], int val[], int n)
{
    if(n==0||W==0)
        return 0;

    int i,w, dp[n+1][W+1];

    for(i=0;i<=n;i++)
    {
        for(j=0;j<=W;j++)
        {
            if(i==0||j==0)
                dp[i][j]=0;
            else if(j>=wt[i-1])
                dp[i][j]=max(val[i-1]+dp[i-1][j-wt[i-1]], dp[i-1][j]);
            else
                dp[i][j]=dp[i-1][j];
        }
    }
    return dp[n][W];
}
```

0/1 knapsack with duplicates allowed

```
int unboundedKnapSack(int w, int n, int val[], int wt[])
{
    int dp[w+1];
    memset(dp, 0, sizeof(dp));
    int ans=0;

    for(int i=0; i<=w; i++)
        for(int j=0; j<n; j++)
            if (wt[j]<=i)
                dp[i] = max(dp[i], dp[i-wt[j]]+
                                    val[j]);

    return dp[w];
}
```

1-2-20

longest common substring:

ex: X = "GeeksforGeeks" Y = "GeeksQuiz"
output: 5

this is similar to LCS

```
int longest_Csubstr(string x, string y)
{
    int xlen = x.length(), ylen = y.length();
    int dp[xlen+1][ylen+1] = {0}, max_len=0;

    for(i=1; i<=xlen; i++)
    {
        for(j=1; j<=ylen; j++)
        {
            if(x[i-1]==y[j-1])
            {
                dp[i][j] = dp[i-1][j-1]+1;
                max_len = max(max_len, dp[i][j]);
            }
        }
    }
    return max_len;
}
```

4-2-20

shortest common supersequence,

Given two strings str1, str2 find the shortest string that has both str1, str2 as subsequences

ex: str1 = "geek" str2 = "eke"

output: geeke

```
int scs(string str1, string str2)
{
    int len1 = str1.length(), len2 = str2.length();

    for(i=1; i<=len1; i++)          // LCS code.
    {
        for(j=1; j<=len2; j++)
        {
            if(str1[i-1] == str2[j-1])
                dp[i][j] = dp[i-1][j-1]+1;
            else
                dp[i][j] = max(dp[i][j-1],
                               dp[i-1][j]);
        }
    }
    return (len1+len2-dp[len1][len2]);
}
```

It is similar to LCS.

5/2/2020

Word break problem: Given a input string and a dictionary of words. find out if the input string can be segmented into sequence of dictionary words.

Ex: ilikesamsung

{i, like, sam, sung, samsung, mobile, ice}

YES

```
t = int(input())
for _ in range(t):
    n = int(input())
    mp = list(input().split())
    str = input()
    len1 = len(str)
    dp = [ [False]*(len1+1) for _ in range(len1+1)]
    for i in range(len1+1):
        dp[i][0] = True
    for i in range(1, len1+1):
        for j in range(1, len1+1):
            if(str[i-1:j] in mp):
                dp[i][j] = dp[i][i-1] | dp[i-1][j]
            else:
                dp[i][j] = dp[i-1][j]
    if(dp[len1][len2]):
        print("YES")
    else:
        print("No")
```

7/2/20

Minimum cost to make string free of a Subsequence.

Given a string str, and an array arr, remove characters from the given string such that no subsequence forms the string "code", cost of removing str[i] is arr[i]. minimize cost and to do the task.

Ex: str = "ccoodddee" arr[] = {3,2,1,3,3,5,1,6}

output: 4 (Remove both 'e')

```
int findCost(string str, int arr[], int n)
{
    int costc, costo, costd, coste;
    costc = costo = costd = coste = 0;

    for(i=0; i<n; i++)
    {
        if(str[i]=='c')
            costc += arr[i];
        else if(str[i]=='o')
            costo = min(costC, costo+arr[i]);
        else if(str[i]=='d')
            costd = min(costo, costd+arr[i]);
        else if(str[i]=='e')
            coste = min(costd, coste+arr[i]);
    }
    return coste;
}
```

11/2/20

Seive of eratosthenes:

```
void Seive(int n)
{
    bool prime[n+1];
    memset(prime, true, sizeof(prime));

    for(int p=2; p*p<=n; p++)
    {
        if(prime[p]==true)
        {
            for(i=p*p; i<=n; i+=p)
                prime[i]=false;
        }
    }
}
```

Time complexity: $O(n*\log(\log(n)))$

27-02-20

Check whether a given graph is Bipartite or Not

```
bool isBipartite(int G[][MAX], int n)
{
    queue <int>q;
    int vis[n+1], col[n+1], i, j, v;

    memset(vis,0,sizeof(vis));
    memset(col,-1,sizeof(col));

for(j=0;j<n;j++)   // for disconnected components
{
    if(vis[j]==1)
        continue;

    q.push(j); vis[j]=1; col[j]=0;

    while(!q.empty())
    {
        v=q.front(); q.pop();
        for(i=0;i<n;i++)
        {
            if(G[v][i])
            {
                if(vis[i]==0)
                    q.push(i),vis[i]=1;
                if(col[i]==-1)
                    col[i]=!col[v];
                else if(col[i]==col[v])
                    return false;
            }
        }
    } // end while.
} // end for
return true;
} // end function
```

01-03-20 — Given undirected, connected, weighted graph

Kruskal's algorithm for minimal spanning tree.

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

bool cmp(pair<ll,pair<ll,ll>>&p1, pair<ll,pair<ll,ll>>&p2)
{
    if(p1.first < p2.first)
        return true;
    return false;
}

ll find_par(ll i, vector<ll>&par)
{
    if(i==par[i])
        return i;
    return par[par[i]]=find_par(par[i],par);
}

int main()
{
    ios_base::sync_with_stdio(false); cin.tie(0);
    ll t,i,n,m,u,v,c,cost,flag;
```

```
cin>>t;
while(t--)
{
    cin>>n>>m;    cost=0; flag=0;
    vector <ll>g[n+1];
    vector <pair<ll,pair<ll,ll>>> edges;
    vector <ll>par(n+1);
    for(i=0; i<=n;i++)  par[i]=i;

    for(i=0;i<m;i++)
    {   cin>>u>>v>>c;
        edges.pb({c,{u,v}});
        g[v].pb(u);
        g[u].pb(v);
    }
    sort(edges.begin(),edges.end(),cmp);

    for(auto itr:edges)
    {
        u=itr.S.F ; v=itr.S.S ;
        u= find_par(u,par);
        v= find_par(v,par);
        if(u!=v)
        {   par[v]=u;
            cost+=itr.F;
        }
    }
    cout<<"minimum spanning tree cost is "<<cost<<endl;
}
} // time complexity O(ElogE) or O(ElogV)
```

*All pair shortest path - Floyd Warshall algorithm:

Given weighted directed graph,

* all pair shortest path algorithm return shortest path between every vertices in graph that can contain negative edge weights (but no negative cycle)
* If graph contains only +ve edge weights solution would to run Dijsktra's algo V times, complexity is $O(V(E+V\log V))$
* If graph contains -ve edge weights then find all-pairs shortest path we can run Bellmanford once from each vertex, complexity would be. $O(V^2E)$ if graph dense then $E=V^2$, then complexity becomes $O(V^4)$

Pseudo code,

let dist[V][V] be contains, distances,

```
for(K=0;K<V;
for(K=0;K<V;k++)
{
    for(i=0;i<V;i++)
    {
        for(j=0;j<V;j++)
        {
            if( dist[i][K]+dist[K][j] < dist[i][j])
                dist[i][j]= dist[i][K]+dist[K][j];
        }
    }
}
```

detecting negative weight cycle: (i==j) diagonal

any element in diagonal of distance matrix (after execution of Algo) is negative, then, negative weight cycle is present

* because, initially length of path (i,i) is zero. A path [i,k...i] can only improve upon this if it has length less than zero (ie) denotes a negative cycle.

2003-20

Diameter of a tree (Diameter is the longest path between two nodes of a tree)

```
int maxD=-1,vis[1000];
int main( )
{
    int n,a,b;
    cin>>n;  vector<int> arr[n];
    for(i=0;i<n;i++)
        arr[a].push_back(b), arr[b].push_back(a);

    maxD=-1
    dfs(1,0);

    memset (vis,0,sizeof(vis));
    maxD=-1;
    dfs(maxNode,0);
    cout<<maxD;

}

void dfs( int node, int d)
{
    vis[node]=1;
    if(d>maxD)
        maxD=d, maxNode=node;

    for(int child:arr[node])
        if(vis[child]==0)
            dfs(child,d+1);

}
```

28-03-20

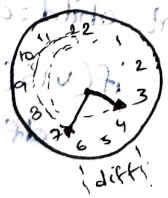
find angle between hour hand and minute hand.

We first find the angle of hour and minute w.r.t 12 o'clock then take difference. of them, we get angle between them, inorder to get smaller angle we print min(ans, 360-ans)

```
int main()
{
  int t;
  double h,m,ans;
  cin>>t;
  while(t--)
  {
    cin>>h>>m;
    if(m==60)
      m=0;
    double angle_h = h*30 + m*0.5;
    double angle_m = m*6;
    ans = fabs(angle_h - angle_m);
    cout<<int(min(ans,360-ans))<<"\n";
  }
return 0;
}
```

for 60 minutes ⇒ 30°
1 minute → 30/60

for 30 minute ⇒ 180°
1 minute = 6



3-4-20

Segment tree

Segment tree can perform update and range operations in O(log(n)), following code finds min in segment [L,R]
all inputs are 0 indexed.

```
ll tree[4*100005];
int main()
{
  cin>>n>>q;
  for(i=0;i<n;i++)
    cin>>arr[i];

  build(arr,1,0,n-1);
  while(q--)
  {
    cin>>type;
    if(type=='U')
    {
      cin>>x>>y;
      update(arr,1,0,n-1,x,y);
    }
    else
    {
      cin>>l>>r;
      cout<<range_min(arr,1,0,n-1,l,r)<<"\n";
    }
  }
return 0;
}
```

```
void build(ll arr[], ll v, ll l, ll r)
{
    if(l==r)
        tree[v]=arr[l];
    else
    {
        ll mid=(l+r)/2;
        build(arr, 2*v, l, mid);
        build(arr, 2*v+1, mid+1, r);
        tree[v]=min(tree[2*v]+tree[2*v+1]);
    }
}

void update(ll arr[], ll v, ll l, ll r, ll x, ll y)
{
    if(l==r)
    {
        tree[v]=y;
        arr[x]=y;
        return;
    }
    ll mid=(l+r)/2;
    if(l<=x && x<=mid)
        update(arr, 2*v, l, mid, x, y);
    else
        update(arr, 2*v+1, mid+1, r, x, y);

    tree[v]=min(tree[2*v], tree[2*v+1]);
}
```

```
ll range_min(ll arr[], ll v, ll s, ll e, ll l, ll r)
{
    if(s>r || l>e)
        return INT_MAX;
    if(l<=s && e<=r)
        return tree[v];
    ll mid=(s+e)/2;
    return min(range_min(arr, 2*v, s, mid, l, r),
    range_min(arr, 2*v+1, mid+1, e, l, r));
}
```

29-4-20

Reversing a linked list.

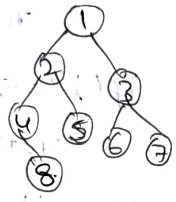
```
Node *reverse(Node *head)
{
    if(head==NULL) return NULL;
    Node *curr, *forward, *pre;
    curr=head->next;
    pre=head;
    head->next=NULL;
    while(curr!=NULL)
    {
        forward=curr->next;
        curr->next=pre;
        pre=curr;
        curr=forward;
    }
    return head=pre;
}
```

6-5-20

Construct a BinaryTree from Postorder and Inorder

in[] = {4,8,2,5,1,6,3,7}

Post[] = {8,4,5,2,6,7,3,1}



```
Node *buildTree(int in[], int post[], int n)
{   int idx = n-1;  // for preorder set idx=0;
    return construct(0, n-1, &idx, in, post);
}

Node *construct(int l, int r, int *idx, int in[], int post[])
{
    if(l>r) return NULL;
    int curr_root = post[*idx];
        (*idx)--;  // ++ if preorder
    Node *node = (struct Node*) malloc(sizeof(struct Node));
    node->data = curr_root;
    node->left = node->right = NULL;
    int curr_l = l;
    while(curr_l<=r && in[curr_l]!=curr_root)
        curr_l++;
    node->right = construct(curr_l+1, r, idx, in, post);
    node->left = construct(l, curr_l-1, idx, in, post);
    return node;
}
```

swap it if preorder

Reverse linked list in groups of size k

```
Node *reverse(Node *head, int k)
{   Node *current = head;
    Node *forward = NULL;
    Node *pre = NULL;
    int count=0;
    if(head==NULL)
        return NULL;
    while(current!=NULL && count<k)
    {
        forward = current->next;
        current->next = pre;
        pre = current;
        current = forward;
        count++;
    }
    head->next = reverse(forward, k);
    return pre;
}
```

8-5-20

merge overlapping intervals.

```
void merge_intervals(vector<pair<ll,ll>>&arr, int n)
{
    sort(arr.begin(),arr.end, cmp);

    pre_end= arr[0].S
    min_start = arr[0].F
    idx=0;

    for(i=1;i<n;i++)
    {   if(arr[i].F<=pre_end)
            pre_end = max(arr[i].S,pre_end);

        else
        {
            arr[idx].F= min_start;
            arr[idx].S = pre_end;
            idx++;
            pre_end=arr[i].S
            min_start = arr[i].F
        }
    }

    arr[idx].F=min_start;
    arr[idx].S = pre_end;

    for(i=0;i<=idx;i++)
        cout<<arr[i].F<<" "<<arr[i].S<<" ";

    cout<<"\n";
}
```

Ex: Ip: [[1,3],[2,4],[6,8],[9,10]]
Op: [1,4] [6,8], [9,10]

```
bool cmp(pair<ll,ll>&p1,pair<ll,ll>&p2)
{   if(p1.F<p2.F)
        return true;
    else if(p1.F==p2.F)
        return (p1.S<p2.S)
    return false;
}
```

or simply
return p1.F<p2.F

9-5-20 - GFG

Clone a linked list with next & random pointer in O(1) space.

```
Node *copyList(Node *head)
{   if(head==NULL)
        return NULL;

    Node *temp=head,*n,*clone_head,*clone;

    while(temp!=NULL)
    {
        n=(struct Node*)malloc(sizeof(struct Node));
        n->data = temp->data; // inserting new node
        n->next =temp->next;      after every node..
        n->arb = NULL;
        temp->next = n;

        temp = temp->next->next;
    }
```

```
temp = head;

while(temp!=NULL && temp->next != NULL)
{
    if(temp->arb!=NULL)        // assigning random pointers to clones nodes.
        temp->next->arb = temp->arb->next;
    else
        temp->next->arb = NULL;

    temp = temp->next->next;
}

clone_head = head->next;

temp=head; clone=clone_head;

while(clone!=NULL && temp!=NULL)
{
    if(temp->next!=NULL)
    {
        temp->next = temp->next->next;
        temp = temp->next;
    }
    if(clone->next!=NULL)
    {
        clone->next = clone->next->next;
        clone = clone->next;
    }
}

return clone_head;
}
```

18-5-20

Egg dropping puzzle

```
int egg_drop(int e, int n)        // e - eggs, n - floors
{
    int dp[e+1][n+1];
    for(i=1;i<=n;i++)   // if one egg, means no of trails = no of floors
        dp[1][i]=i;

    for(i=1;i<=e;i++)   // if one floor one trail,
        dp[i][1]=1, dp[i][0]=0;

    for(i=2;i<=e;i++)
    {
        for(j=2;j<=n;j++)
        {
            dp[i][j]=INT_MAX;
            for(k=1;k<=j;k++)
                dp[i][j]=min(dp[i][j],
                    1+max(dp[i-1][k-1],dp[i][j-k]));
        }
    }

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

matrix chain multiplication

for n, n-1 matrices will be there

40, 20, 30, 10, 30

A B C D

```
int MCM(int l, int r)
{
    if(l==r) return 0;
    if(dp[l][r]!=-1)
        return dp[l][r];
    int i,ans=INT_MAX,temp;
    for(i=l; i<r; i++)
    {
        temp = MCM(l,i)+mcm(i+1,r)+
               arr[l-1]*arr[i]*arr[r];
        if(temp<ans)
            ans= temp;
    }
    return dp[l][r]=ans;
}
```

ex: [40,20,30,10,30]

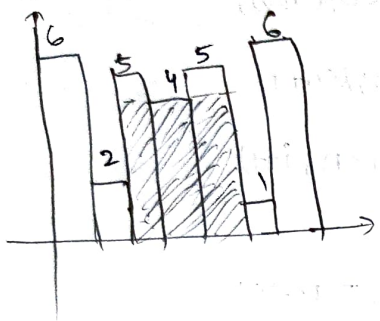
o/p: 26000

tabulation for matrix chain multiplication.

```
memset(dp,0,sizeof(dp));
for(len=2;len<n;len++)
{
    for(i=1; i+len-1<n; i++)
    {
        j=i+len-1;
        dp[i][j]=INT_MAX;
        for(k=i;k<j;k++)
        {
            curr_min=dp[i][k]+dp[k+1][j]+
                     arr[i-1]*arr[k]*arr[j];
            if(curr_min<dp[i][j])
                dp[i][j]= curr_min;
        }
    }
}
cout<<dp[1][n-1];
```

24-05-20

Largest Rectangle box Area in Histogram.




```
int max_area(int arr[], int n)
{
    int i=0, max_area=0, top;
    stack<int> s;

    while(i<n)
    {   if( s.empty() || arr[s.top()]<=arr[i])
            s.push(i++);
        else
        {
            top = s.top(); s.pop();
            curr_area = arr[top] *
                    (s.empty() ? i : i-s.top()-1);
            max_area = max(max_area, curr_area);
        }
    }
    while(!s.empty())
    {   top=s.top(); s.pop();
        curr_area = arr[top]*(s.empty()?i: i-s.top()-1);
        max_area = max(max_area, curr_area);
    }
    return max_area;
}
```

30-05-20

Coin change problem:

We have infinite supply of given coins, in how many ways we can obtain given amount.

```
int change(int amount, vector<int>&coins)
{   int i,j,n=coins.size(),coin;
    int dp[n+1][amount+1];

    if(n==0) return (amount==0);

    for(i=0;i<=amount;i++) // with zero denomination we can't cover any amount except, amount zero
        dp[0][i]=0;

    for(i=0;i<=n;i++)  // amount zero can be obtained from any denomination just taking none of them
        dp[i][0]=1;

    for(i=1;i<=n;i++)
    {   coin=coins[i-1];
        for(j=1;j<=amount;j++)
        {   if(coin>j) dp[i][j]=dp[i-1][j];
            else
                dp[i][j]=dp[i-1][j]+dp[i][j-coin];
        }
    }
    return dp[n][amount];
}
```

29-06-20

Celebrity problem

You are given grid[ ][ ] where grid[i][j] represent, whether person j is known to i. Celebrity, doesn't know anyone, and everyone knows celebrity.

```
int getId(int grid[MAX][MAX], int n)
{
    int candidate=0,i;

    for(i=0; i<n; i++)
        if(grid[candidate][i])
            candidate=i;

    // checking candidate is really celebrity
    for(i=0;i<n;i++)
    {   if(i==candidate) continue;

        if(grid[candidate][i] || !grid[i][candidate])
            return -1;

    }
    return candidate;
}
```