template sheet

NUMBER THEORY:

1)Sum of all divisors up to n in o(n) time:

```
int snod=0;
for(i=1 to n ){
snod+=(n/i);
 }
```

2)Sieve: o(n)

```
prime[2]=1;
for(int i=3;i<=n;i+=2)prime[i]=1;

for(int i=3;i<=sqrt(n);i+=2){
   if(prime[i]==1){
        for(int j=i*i;j<=n;j+=i){
                prime[j]=0;
        }
   }
}
for(int i=2;i<=n;i++) if(prime[i]==1){v.push_back(i);}
```

3)prime factorization:

```
vector<int>v;
for(auto p: primes){
if(1LL*p*p>n){break;}
        if(n%p==0){
                while(n%p==0){
                n=n/p;
                v.push_back(p);}
        }
}

if(n>1){
        v.push_back(n);
}
```

4)Number of divisors of a number:

```
vector<int>v;
int nod=1;
for(auto p: primes){
if(1LL*p*p>n){break;}
```

```
        if(n%p==0){
                cnt=1;
                v.push_back(p);

                while(n%p==0){
                  n=n/p;
                  cnt++;
                }
                nod*=p;
        }
}

if(n>1){
        v.push_back(n);
        nod*=2;
}
```

5) **Divisor summatory function: O(sqrt(n))**

```
int snod=0;
int root=sqrt(n);
for(i=1 to root ){
snod+=((n/i)-i);
 }
snod*=2;
snod+=sq;
```

6) SOD using LCM,GCD:

```
vector<int>v;
int sod=1;
for(auto p: primes){

if(1LL*p*p>n){break;}
        int a=1;
        sum=1;
        if(n%p==0){
                while(n%p==0){
                a*=p;
                sum+=a;
                n=n/p;
                //v.push_back(p);
                }
```

```
        sod*=sum;
}

        }

if(n>1){
        sod*=(1+n);
        //v.push_back(n);
    }
}
```

6) PHI function(1 to n koita number ase which are relatively prime to n):

O(m*n)=O(m)*o(n)

If P is a prime… O(p)=p-1.//as oita chara baki ektar satheo prime na.

1 to p^a porjonto number divisible by p or 1 to p^a porjonto number jader gcd is more than one = p^(a-1). So O(p^a)=p^a - p^(a-1).
= p^a * (p-1)/p

Code:

```
vector<int>v;
Int phi =n;
for(auto p: primes){
if(1LL*p*p>n){break;}
        if(n%p==0){
                while(n%p==0){
                n=n/p;
                }
        phi/=p;
        phi*=(p-1);
        }
}

if(n>1){
        phi/=n;
        phi*=(n-1);

}
```

1 to n range er shob int er phi value ber korte hole in O(nloglogn):

```
void phi_1_to_n(int n) {
    vector<int> phi(n + 1);
   phi[0] = 0;
   phi[1] = 1;
   for (int i = 2; i <= n; i++)
       phi[i] = i - 1;

   for (int i = 2; i <= n; i++)
       for (int j = 2 * i; j <= n; j += i)
           phi[j] -= phi[i];
}
```

As phi(p^k) = p^k - p^(k-1) if p is a prime number and k > 0

SYNTAX
Precision:
cout<<setprecision(3)<<x;

N! er prime factorization is: 2^(p1) * 3^(p2) ……

P1 = floor(n/2) + floor(n/4) +floor(n/8).....until value becomes zero
P2 = floor(n/3) + floor(n/9) +floor(n/27).....until value becomes zero
…..for all prime factors of N

Generating all divisors using prime factors recursively:

```
void setDivisors(int n, int i) {// factor hcche pair er array….factor and their power
        int j, x, k;
        for (j = i; j<factors.size(); j++) {
            x = factors[j].first * n;
            for (k = 0; k<factors[j].second; k++) {
                divisors.push_back(x);
                 setDivisors(x, j + 1);
                 x *= factors[j].first;
            }
        }
}
```

Kadane's algorithm:

```
int maxSubarraySum(vector<int> &arr) {

    // Stores the result (maximum sum found so far)
    int res = arr[0];

    // Maximum sum of subarray ending at current position
    int maxEnding = arr[0];

    for (int i = 1; i < arr.size(); i++) {

        // Either extend the previous subarray or start
        // new from current element
        maxEnding = max(arr[i], maxEnding + arr[i]);

        // Update result if the new subarray sum is larger
        res = max(res, maxEnding);
    }
    return res;
}
```

Binary exponentiation:

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

long long power(long long A, long long B)
{
    if (B == 0)
        return 1;

    long long res = power(A, B / 2) % 9; // ensure recursive result is mod 9

    if (B % 2)
        return (res * res % 9) * (A % 9) % 9;
    else
        return (res * res) % 9;
}

int main()
{
    cout << power(2, 12) % 9 << "\n"; // final result also mod 9
    return 0;
}
```

dp

DP:

1D:

```
vector<int> dp(n,-1);
  dp[0]=0;
  for(int ind=1;ind<n;ind++){
      int jumpTwo = INT_MAX;
        int jumpOne= dp[ind-1] + abs(height[ind]-height[ind-1]);
        if(ind>1)
            jumpTwo = dp[ind-2] + abs(height[ind]-height[ind-2]);

        dp[ind]=min(jumpOne, jumpTwo);
  }
```

2D:

```
int minPathSum(vector<vector<int>>& grid) {
        int m=grid.size();
        int n=grid[0].size();
        vector<vector<int>> dp(m, vector<int>(n, 0));
        dp[0][0]=grid[0][0];

        // first row
        for(int j = 1; j < n; j++)
            dp[0][j] = grid[0][j] + dp[0][j-1];

        // first column
        for(int i = 1; i < m; i++)
            dp[i][0] = grid[i][0] + dp[i-1][0];

        for(int i=0;i<m;i++){
            for(int j=1;j<n;j++){
                if(i>0 && j>0)
                {
                    dp[i][j] = grid[i][j] + min(dp[i-1][j], dp[i][j-1]);

                }
            }
        }
        return dp[m-1][n-1];
    }
```

```
int ninjaTraining(int n, vector<vector<int>>& points)
{
  vector<vector<int>> dp(n, vector<int>(4, 0));
// dp[i][j] represents the maximum points at day i, considering the last activity as j
  dp[0][0] = max(points[0][1], points[0][2]);
  dp[0][1] = max(points[0][0], points[0][2]);
  dp[0][2] = max(points[0][0], points[0][1]);
  dp[0][3] = max(points[0][0], max(points[0][1], points[0][2]));

  for (int day = 1; day < n; day++) {
    for (int last = 0; last < 4; last++) {
      dp[day][last] = 0;
      // Iterate through the tasks for the current day
      for (int task = 0; task <= 2; task++) {
        if (task != last) {
          int activity = points[day][task] + dp[day - 1][task];
          dp[day][last] = max(dp[day][last], activity);
        }
      }
    }
  }

  // The maximum points for the last day with any activity can be found in dp[n-1][3]
  return dp[n - 1][3];
}
```

binary search

```
1. int rotated_search(vector<int>& nums, int target)
 {
     int left=0, right=nums.size()-1;
     int mid;
     while(left<=right)
     {
         mid = (left+right)/2;
         if(nums[mid]>=nums[left] )
         {

             if(target>=nums[left] && target<=nums[mid])//sorted
             {
                 right=mid;
                 mid=(right+left)/2;

                 if(nums[mid]<target)
                 {
                     left=mid+1;
                 }
                 else if (nums[mid]>target)
                 {
                     right=mid-1;
                 }
                 else
                     return mid;

             }
             else{left=mid+1;}

         }
         else
         {
             if(target>=nums[mid] && target<=nums[right])//sorted
             {
                 left=mid;
                 mid=(right+left)/2;

                 if(nums[mid]<target)
                 {
                     left=mid+1;
                 }
                 else if (nums[mid]>target)
```

```
                    {
                        right=mid-1;
                    }
                    else
                        return mid;

                }
                else{right=mid-1;}
            }

        }
        return -1;

    }
```

2.

sub array

Subarray: consecutive ordered elements of an array
Subsequence: consecutive or nonconsecutive ordered elements of an array

```
// optimal for array with negative elements, better for array with non-negative elements
int getLongestSubarray(vector<int>& a, long long k) {
    int n = a.size(); // size of the array.

    map<long long, int> preSumMap;
    long long sum = 0;
    int maxLen = 0;
    for (int i = 0; i < n; i++) {
        //calculate the prefix sum till index i:
        sum += a[i];

        // if the sum = k, update the maxLen:
        if (sum == k) {
            maxLen = max(maxLen, i + 1);
        }

        // calculate the sum of remaining part i.e. x-k:
        long long rem = sum - k;

        //Calculate the length and update maxLen:
        if (preSumMap.find(rem) != preSumMap.end()) {
            int len = i - preSumMap[rem];
            maxLen = max(maxLen, len);
        }

        //Finally, update the map checking the conditions:
        if (preSumMap.find(sum) == preSumMap.end()) {
            preSumMap[sum] = i;
        }
    }

    return maxLen;
}

// optimal for array with non-negative elements
int getLongestSubarray(vector<int>& a, long long k) {
    int n = a.size(); // size of the array.

    int left = 0, right = 0; // 2 pointers
    long long sum = a[0];
    int maxLen = 0;
```

```
    while (right < n) {
        // if sum > k, reduce the subarray from left
        // until sum becomes less or equal to k:
        while (left <= right && sum > k) {
            sum -= a[left];
            left++;
        }

        // if sum = k, update the maxLen, i.e., answer:
        if (sum == k) {
            maxLen = max(maxLen, right - left + 1);
        }

        // Move forward the right pointer:
        right++;
        if (right < n) sum += a[right];
    }

    return maxLen;
}
```

graph

Taking input
1. Adjacency list

```
int n, m;
cin >> n >> m;

vector<vector<int>> adj(n + 1);

for (int i = 0; i < m; i++) {
    int u, v;
    cin >> u >> v;
    adj[u].push_back(v);
    adj[v].push_back(u); // remove if directed
}
```

2. Adjacency matrix

```
int n;
cin >> n;

vector<vector<int>> mat(n, vector<int>(n));

for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
        cin >> mat[i][j];
```

3. Edge List

```
int n, m;
cin >> n >> m;

vector<tuple<int,int,int>> edges;

for (int i = 0; i < m; i++) {
    int u, v, w;
    cin >> u >> v >> w;
    edges.emplace_back(u, v, w);
}
```

4. Implicit Graph (grid graphs)

```
int n, m;
cin >> n >> m;

vector<string> grid(n);
for (int i = 0; i < n; i++)
    cin >> grid[i];
```

syntax

```
 unordered_set < int > s;
 s.insert(i);

for (auto it = s.begin(); it != s.end(); it++) {
   cout << * it << " ";
 }
 if (s.find(2) != s.end())
   cout << n << " is present in unordered set" << endl;
distance(v.begin(), it)
 s.erase(s.begin()); //single element or range
 s.size()
s.empty() == false //boolean
 s.clear();

vector:
v.insert(v.begin(), 5);//position,value

count() – gives the count of a particular element in the multiset.
multiset<int> s;// multiple Same element

Map:
mp.insert({1,10});
for (auto it = mp.begin(); it != mp.end(); it++) {
   cout << it -> first  << it -> second << endl;
 }
multimap<string, int> marks;

   marks.insert({"Alice", 85});
   marks.insert({"Bob", 90});
   marks.insert({"Alice", 92});


Stack,queue;
q.push(v);
deque.push_front(v); dq.push_back(v);
```

Max,min pq:

```
priority_queue<int> pq;
priority_queue<int,vector<int>,greater<int>> pq;
```

```
*min_element(v.begin(),v.end())
```

Merge sort + inversion sort:

```
void or long long merge(vector<int>& arr, int low, int mid, int high) {
        vector<int> temp;
        int left = low, right = mid + 1; long long invCount = 0; // count of inversions
        while (left <= mid && right <= high) {
            if (arr[left] <= arr[right]) temp.push_back(arr[left++]);
            else { temp.push_back(arr[right++]);
                    invCount += (mid - left + 1);
            }
       }
        while (left <= mid) temp.push_back(arr[left++]);
        while (right <= high) temp.push_back(arr[right++]);
        for (int i = low; i <= high; i++) arr[i] = temp[i - low];
          return invCount;
    }
    void or long long mergeSort(vector<int>& arr, int low, int high) {
        if (low >= high)  return 0;
        int mid = (low + high) / 2;  long long invCount = 0;
        invCount +=mergeSort(arr, low, mid);
        invCount +=mergeSort(arr, mid + 1, high);
        invCount +=merge(arr, low, mid, high);
       return invCount;
    }
```

Vector: `vector<int> arr = {5, 3, 3, 4, 1};`

base syntax

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

using namespace std;

#ifdef LOCAL
#include "algo/debug.h"
#else
#define debug(...) 42
#endif
#define ll long long
#define llm LONG_LONG_MAX

void solve()
{
}

int main() {
  ios::sync_with_stdio(false);
  cin.tie(0);
  ll tt,cse=1;
  cin >> tt;
  while (tt--) {
    //cout<< "Case " << cse << ": ";
    //cse++;
    solve();
  }
  return 0;
}
```

shortcuts

```
next_permutation(begin, end);//just next

__builtin_popcount(int num);

sort(begin, end)
sort(arr, arr+3, greater<int>())
bool sortbysec(const pair<int,int> &a,const pair<int,int> &b)
{
    return (a.second < b.second);
}
```

math_tricks

```
int cnt = (int)(log10(n)+1);
```

Repeatedly subtract the smaller number from the larger number until one of them becomes 0. Once one of them becomes 0, the other number is the GCD of the original numbers.and this is also similar to taking remainder until 0.

```
int findGcd(int a, int b) {
    while(a > 0 && b > 0) {
        if(a > b) {
            a = a % b;
        }
        else {
            b = b % a;
        }
    }
    if(a == 0) {
        return b;
    }
    return a;
}
```