UDvorak's notebook (2017)

Contents

1 Snippets

1.1 Snippets Show

```cpp
#include <iostream>
#define show(x) cout << #x << " = " << x << endl;

using namespace std;

int main(){
    int e =32;
    show(e);
}
```

1.2 Snippets Time

```cpp
#include <chrono>
#include <iostream>
using namespace std;
int main(){

    auto start = chrono::high_resolution_clock::now();
    for(long long i = 0; i < 10000000; i++)
        continue;

    auto end = chrono::high_resolution_clock::now();
    chrono::duration<double> diff = end-start;
    cout << diff.count() << endl ;

    return 0;
}
```

1.3 Snippets Size

```cpp
#include <vector>
#include <string>
#include <iostream>
#define sz(a) ((int)(a).size())

using namespace std;

int main(){
    string t = "Hello, what's up";
    vector<int> c (10);
    cout << sz(t) << endl;
    cout << sz(c) << endl;
}
```

1.4 Snippets IsOdd

```cpp
#include <iostream>
#define isOdd(x) (x & 0x01)

using namespace std;

int main (){
    int a =57;
    int b= 32;
    cout << isOdd(a) << endl;
    cout << isOdd(b) << endl;
    return 0;
}
```

1.5 Snippets Assert

```cpp
#include <iostream>
#include <assert.h>
#define isOdd(x) (x & 0x01)
using namespace std;

void test(int num){
    assert(isOdd(num) == 0);
    cout << "Hello: " << num << endl;
}

int main(){
    int a=10, b=22, c=23, d=32;
    test(a);
    test(b);
    test(c);
    test(d);
}
```

1.6 Snippets StructPriorityQueue

```cpp
#include <iostream>
#include <queue>

using namespace std;

typedef priority_queue<edge> pq;

struct edge{
    int to, weight;
    edge(){}
    edge(int _to, int _weight){
        to = _to;
```

```
        weight = _weight;
    }
    bool operator < (edge e) const {
        return weight > e.weight;
    }
};

int main(){
    pq edges;

    edges.push(edge(1, 23));
    edges.push(edge(2, 3));
    edges.push(edge(3, 10));
    edges.push(edge(4, 11));
    edges.push(edge(5, 4));

    while(!edges.empty()){
        edge a = edges.top();
        edges.pop();
        cout << a.to << endl;
    }
}
```

1.7 Snippets utilities Directives1

```
#include <iostream>
using namespace std;
#define concat(a, b) a ## b

int main() {
   int xy = 100;
   cout << concat(x, y);
   return 0;
}
```

1.8 Snippets utilities Namespace1

```
#include <iostream>
using namespace std;

/*REFERENCE
 *https://www.tutorialspoint.com/cplusplus/cpp_namespaces.htm
 */

// first name space
namespace first_space{
   void func(){
      cout << "Inside first_space" << endl;
   }
}

// second name space
namespace second_space{
   void func(){
      cout << "Inside second_space" << endl;
   }
}

int main () {

   // Calls function from first name space.
   first_space::func();

   // Calls function from second name space.
   second_space::func();

   return 0;
}
```

1.9 Snippets utilities ClassPointers

```
#include <iostream>
using namespace std;

class Person {
   public:
      Person() {
         cout << "Constructor called!" <<endl;
      }

      ~Person() {
         cout << "Destructor called!" <<endl;
      }
};

int main( ) {
   Person* myBoxArray = new Person[4];
   delete [] myBoxArray; // Delete array
   return 0;
}
```

1.10 Snippets utilities Debug

```
#include <iostream>
using namespace std;
#define DEBUG

#define MIN(a,b) (((a)<(b)) ? a : b)

int main () {
   int i, j;
   i = 100;
   j = 30;

   #ifdef DEBUG
      cerr <<"Trace: Inside main function" << endl;
   #endif

   #if 0
      /* This is commented part */
      cout << MKSTR(HELLO C++) << endl;
   #endif

      cout <<"The minimum is " << MIN(i, j) << endl;

   #ifdef DEBUG
      cerr <<"Trace: Coming out of main function" << endl;
   #endif
      return 0;
}
```

1.11 Snippets utilities Directives2

```
#include<iostream>

using std::cout;
using std::cin;
using std::endl;
```

```
/*
 * g++ -E test.cpp  > sal.out
 * compile with that command and see how the compiler replace the
     constant
 */
#define PI 3.141516
#define MIN(a,b) (((a)<(b)) ? a : b)

int main(){

    cout << "The number PI is " << PI << endl;
    cout <<"The minimum is " << MIN(i, j) << endl;
    return 0;

}
```

1.12 Snippets utilities PointersDeclaration

```
#include <iostream>

using namespace std;

int main(){

    double * data;
    data = new double;

    *data = 123.34;

    cout <<  *data << endl;

    delete data;

    return 0;

}
```

1.13 Snippets utilities PredefinedMacros

```
#include <iostream>
using namespace std;

int main () {
   cout << "Value of __LINE__ : " << __LINE__ << endl;
   cout << "Value of __FILE__ : " << __FILE__ << endl;
   cout << "Value of __DATE__ : " << __DATE__ << endl;
   cout << "Value of __TIME__ : " << __TIME__ << endl;

   return 0;
}
```

1.14 Snippets utilities Template

```
#include <iostream>
#include <string>

/*REFERENCES
 * https://www.tutorialspoint.com/cplusplus/cpp_templates.htm
 */
using namespace std;

template <typename T>
inline T const& Max (T const& a, T const& b)  {
   return a < b ? b:a;
}

int main () {

   int i = 39;
   int j = 20;
   cout << "Max(i, j): " << Max(i, j) << endl;

   double f1 = 13.5;
   double f2 = 20.7;
   cout << "Max(f1, f2): " << Max(f1, f2) << endl;

   string s1 = "Hello";
   string s2 = "World";
   cout << "Max(s1, s2): " << Max(s1, s2) << endl;

   return 0;
};
```

1.15 Snippets utilities Namespace2

```
#include <iostream>
using namespace std;

/*REFERENCES
 *https://www.tutorialspoint.com/cplusplus/cpp_namespaces.htm
 */
// first name space
namespace first_space{
   void func(){
      cout << "Inside first_space" << endl;
   }
}

// second name space
namespace second_space{
   void func(){
      cout << "Inside second_space" << endl;
   }
}

using namespace first_space;
int main () {

   // This calls function from first name space.
   func();

   return 0;
}
```

1.16 Snippets utilities CommaOperator

```
#include <iostream>
using namespace std;

int main() {
   int i, j;

   j = 10;
   i = (j++, j+100, 999+j);

   cout << i;

   return 0;
}
```

1.17 Snippets utilities ArrayPointers

```
#include <iostream>

using namespace std;

inline void example_1(){
    char * name;
    name = new char[10];
    delete [] name;
}

inline void example_2(){
    int row = 4;
    int col = 3;

    //Allocate memory for rows
    double **pvalue  = new  double* [row];
    //Now allocate memory for columns
    for (int i=0; i<col; i++){
        pvalue[i] = new double[col];
    }


    //Now release memory
    for(int i = 0; i < row; i++) {
        delete [] pvalue[i];
    }
    delete [] pvalue;
}

int main(){
    example_1();
    example_2();


    return 0;

}
```

1.18 Snippets For

```
#include <iostream>

#define forn(i, n) for(int i = 0 ; (i) < (n) ; ++i)

using namespace std;

int main(){

    forn(_,10){
        cout << "with out variable" << endl;
    }

    forn(i,10){
        cout << "with variable: " << i << endl;
    }
    return 0;

}
```

1.19 Snippets Swap

```
#include <iostream>
#define swap(x,y) (x^=y, y^=x, x^=y)

using namespace std;

int main(){
    int x=324;
    int y=232;
    cout << x << " " << y << endl;
    swap(x,y);
    cout << x << " " << y << endl;
    return 0;
}
```

1.20 Snippets FreeOpen

```
#include <iostream>
#include <stdio.h>

using namespace std;

int main (){
    freopen("data.in", "r", stdin);
    freopen("data.out", "w", stdout);

    return 0;

}
```

1.21 Snippets StringStream

```
#include <iostream>
#include <sstream>
#include <string>
using namespace std;

int main(){
    string line;
    while (getline(cin, line)){
        stringstream ss(line);
        string word;
        int count = 0;
        while ( ss >> word) count ++;
        cout << endl << "# Words: " << count << endl;
    }
}
```

1.22 Snippets UpperLowerBound

```
// lower_bound/upper_bound example
#include <iostream>     // cout
#include <algorithm>    // lower_bound, upper_bound, sort
#include <vector>       // vector
using namespace std;
int main () {
  int myints[] = {10,20,30,30,20,10,10,20};
  vector<int> v(myints,myints+8);           // 10 20 30 30 20 10
      10 20

  sort (v.begin(), v.end());                // 10 10 10 20 20 20
      30 30

  vector<int>::iterator low,up;
  low=lower_bound (v.begin(), v.end(), 20); //          ^
  up= upper_bound (v.begin(), v.end(), 20); //                   ^
```

```
  cout << "lower_bound at position " << (low- v.begin()) << '\n';
  cout << "upper_bound at position " << (up - v.begin()) << '\n';
  /*
lower_bound at position 3
upper_bound at position 6
   */
  return 0;
}
```

1.23 Snippets CompareDoubles

```
#include <stdio.h>

using namespace std;
const double EPS = 1e-15;

/*
 * Return
 * -1   if x < y
 *  0   if x == y
 *  1   if x > y
 */
int cmp (double x, double y){
    return (x <= y + EPS) ? (x + EPS < y) ? -1 : 0 : 1;
}

int main(){
    double d1 = 0.00000000000212;
    double d2 = 0.00000000000213;
    int res = cmp(d1,d2);
    if (res == 0){
        printf("Equal \n");
    }else if(res == 1){
        printf("Greater\n");
    }else {
        printf("Less \n");
    }

}
```

2 BasicOperations

3 Structures

3.1 Structures RecoveryTree

```
#include <iostream>
using namespace std;
/**Build a binary tree form a
   inorder and preoder string **/

int preIndex = 0;
struct node {
    char key;
    node *left, *right;

    node(int k) {
        key = k;
        left = NULL;
        right = NULL;
    }
};
int search(string word, int b, int e, char c) {
    for(int i=b; i<=e; i++) {
        if(word[i] == c) return i;
    }
    return -1;
}
//Set preIndex to 0 to build another tree
node* build(string in, string pre, int b, int e) {
    if(b > e)
        return NULL;
    node *root = new node(pre[preIndex++]);
    if(b == e)
        return root;
    int inIndex = search(in, b, e, root->key);

    root->left = build(in, pre, b, inIndex - 1);
    root->right = build(in, pre, inIndex + 1, e);
    return root;
}

int main() {
    string pre, in;
    node *tree;
    while(cin >> pre >> in) {
        tree = build(in, pre, 0, pre.size() - 1);
        preIndex = 0;
    }
    return 0;
}
```

3.2 Structures Prim

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

using namespace std;
const int MAXN = 10005;
typedef pair <int, int> edge;
// Pareja (nodo, peso)
typedef pair <int, int> weight_node; // Pareja (peso, nodo)
vector <edge> g[MAXN];
// Lista de adyacencia
bool visited[MAXN];

// Retorna el costo total del MST
int prim(int n){ // n = n mero de nodos
    for (int i = 0; i <= n; ++i) visited[i] = false;
    int total = 0;
    priority_queue<weight_node, vector <weight_node>,
    greater<weight_node> > q;
    // Empezar el MST desde 0 (cambiar si el nodo 0 no existe)
    q.push(weight_node(0, 0));
    while (!q.empty()){
        int u = q.top().second;
        int w = q.top().first;
        q.pop();

        if (visited[u]) continue;
        visited[u] = true;
        total += w;
        for (int i = 0; i < g[u].size(); ++i){
            int v = g[u][i].first;
            int next_w = g[u][i].second;
            if (!visited[v]){
                q.push(weight_node(next_w, v));
            }
        }
```

```
    }
    return total;
}

int main(){
    //Nodo 0 se une al 1 con peso 1
    g[0].push_back(edge(1,1));
    //Nodo 0 se une al 2 con peso 2
    g[0].push_back(edge(2,2));
    //Nodo 0 se une al 3  con peso 3
    g[0].push_back(edge(3,3));

    g[1].push_back(edge(5,4));
    g[2].push_back(edge(4,5));
    g[3].push_back(edge(4,1));

    cout << prim(4);

    return 0;
}
```

3.3 Structures SegmentTree

```
#include <iostream>
#define left(x) x << 1
#define right(x) x << 1 | 1
#define ROOT 1
using namespace std;

void build(int *T, int *A, int node, int start, int end) {
    if(start == end) {
        T[node] = A[start];
    } else {
        int mid = (start + end) / 2;
        build(T, A, left(node), start, mid);
        build(T, A, right(node), mid + 1, end);
        // Merging the children
        T[node] = T[left(node)] + T[right(node)];
    }
}

void update(int *T, int *A, int node, int start, int end, int i,
    int val ) {
    if(start == end) {
        A[i] = val;
        T[node] = val;
    } else {
        int mid = (start + end) / 2;
        if(start <= i && i <= mid)
            update(T, A, left(node), start, mid, i, val);
        else
            update(T, A, right(node), mid + 1, end, i, val);
        T[node] = T[left(node)] + T[right(node)];
    }
}

int query(int *T, int node, int start, int end, int a, int b) {
    if(b < start || end < a)  // out of he boundaries
        return 0;
    if(a <= start && end <= b)
        return T[node];

    int mid = (start + end) / 2;
    int p1 = query(T, left(node), start, mid, a, b);
    int p2 = query(T, right(node), mid + 1, end, a, b);

    return p1 + p2;
}
int main() {
    int size = 5;

    int *a = new int[size];
    int *stree = new int[4 * size];

    for (int i = 0;  i < size ; i++) a[i] = i + 1;

    build(stree, a, ROOT, 0, size - 1);

    int from = 0, to = 4;
    cout << query(stree, ROOT, 0, size-1, from, to) << endl;
    return 0;
}
```

3.4 Structures FenwickTree

```
#include <bits/stdc++.h>
using namespace std;
#define flag(x) printf("[%d]\n", x)
typedef vector<int> vi;
struct fenwick_tree {
    vi data;
    fenwick_tree(int _n) : data(vi(_n + 1, 0)) {}
    void update(int i, int val)  {
        while(i < data.size()) {
            data[i] += val;
            i += i & (-i);
        }
    }
    int query(int i) {
        int sum = 0;
        while(i > 0) {
            sum += data[i];
            i -= i & (-i);
        }
        return sum;
    }
    int query_segment(int a, int b) {
        return query(b) - query(a - 1);
    }
};
int main() {
    int x[5] = {1, 2, 3, 4, 5};
    fenwick_tree *fq = new fenwick_tree(8);
    for(int i=0; i<5; i++)
        fq->update(i + 1, x[i]);
    //Node 0 -> dummy node
    for(int i=1; i<fq->data.size(); i++) {
        cout << fq->data[i] << ' ';
    }cout << endl;
    //Sum interval [1 - 4]
    flag(fq->query(4));
    //Sum interval [3 - 5]
    flag(fq->query_segment(3, 5));
    return 0;
}
```

3.5 Structures MaxMinPHeap

```
/**Utility STL Data Structures*/
/**Max Heaps*/
priority_queue <int> pq;
```

```
/**Min Heaps*/
priority_queue <int, vector<int>, greater<int> > pq;
```

3.6 Structures BinaryTree

```
#include <iostream>

using namespace std;

struct node {
    int val = 0;
    node * l = nullptr;
    node * r = nullptr;
};

inline node* build(node *head, int value){
    node *son = new node;
    son->val = value;
    if (head == nullptr) return son;
    node * aux = head, * nxt = head;
    while(nxt != nullptr){
        aux = nxt;
        if (value > nxt->val) nxt = nxt->r;
        else nxt = nxt->l;
    }
    if(value > aux-> val) aux->r = son;
    else aux->l = son;
    return head;
}

inline void show(node* head){
    if (head==nullptr) return;
    show(head->l);
    cout << head->val << endl;
    show(head->r);
}
int main(){
    node *head = new node;
    head->val = 5;

    head = build(head, 45);
    head = build(head, 20);

    show(head);
    return 0;
}
```

3.7 Structures Trie

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

using namespace std;

/*
 * Struct for a trie
 */
struct node {
        node * son[26];
        bool is_end;
        int num_times;

        node(){
                memset(son, 0, sizeof(son));
                is_end =false;
                num_times =0;
        }
};

/*
 * insert a word in the trie
 */
void insert(node* nd, char *s){
        if(*s){
        int pos = *s - 'a';
                if(!nd->son[pos]) nd->son[pos]=new node();
                insert(nd->son[pos], s+1);
        }else{
                nd->is_end = true;
        }
}

/*
 * Check if the word is in the trie
 */
int contains(node *nd, char *s){
        if(*s){
        int pos = *s - 'a';
                if(!nd->son[pos]) return false;
                return contains(nd->son[pos], s+1);
        }else{
                return nd->is_end;
        }
}

//This is just the driver program
int main(){
    node * trie = new node();

    string  a = "word";
    char *cstr = new char[a.length() + 1];
    strcpy(cstr, a.c_str());
    insert (trie, cstr);

    string b = "banani";
    strcpy(cstr, b.c_str());
    insert (trie, cstr);

    if (contains(trie, cstr)){
        cout << "ohh holly xx." << endl;
    }else{
        cout << "mother ..." << endl;
    }
        return 0;
}
```

4 NP$_{Problem}$

5 Combinatory

5.1 Combinatory BinomialPYTHON

```
import math
import sys

MAXN = 431
choose = []
for i in range (0, MAXN+5):
    choose.append([0]*(MAXN+5))

def binomial(N):
```

```
    for n in range (0, N+1):
        choose[n][0] = choose[n][n] = 1
    for n in range(1, N+1):
        for k in range(1, n):
            choose[n][k] = choose[n-1][k-1] + choose[n-1][k]
N = 431
binomial(N)
for line in sys.stdin:
    n, k = [int(y) for y in line.split()]
    val = choose[n][k]
    print(val)
```

6 Primes

6.1 Primes PollarRhoPYTHON

```
import random

def gcd( a, b):
    if(b == 0): return a;
    return gcd(b, a % b);

def pollardRho(N):
        if N%2==0:
                return 2
        x = random.randint(1, N-1)
        y = x
        c = random.randint(1, N-1)
        g = 1
        while g==1:
                x = ((x*x)%N+c)%N
                y = ((y*y)%N+c)%N
                y = ((y*y)%N+c)%N
                g = gcd(abs(x-y),N)
        return g

print(pollardRho(10967535067))
print(pollardRho(113))
```

6.2 Primes MillerTest

```
// C++ program Miller-Rabin primality test
#include <bits/stdc++.h>
using namespace std;

int power(long long x, unsigned long long y, long p){
        int res = 1;      // Initialize result
        x = x % p; // Update x if it is more than or
                                // equal to p
        while (y > 0){
                if (y & 1)
                        res = (res*x) % p;
                y = y>>1; // y = y/2
                x = (x*x) % p;
        }
        return res;
}

bool miillerTest(long long d, long n){
        long long a = 2 + rand() % (n - 4);

        long long  x = (long long)power(a, d, n);

        if (x == 1 || x == n-1)
                return true;
        while (d != n-1){
                x = (long long)(x * x) % n;
                d *= 2;
                if (x == 1)      return false;
                if (x == n-1) return true;
        }

        return false;
}
bool isPrime(long long n, long long k){
        // Corner cases
        if (n <= 1 || n == 4) return false;
        if (n <= 3) return true;

        // Find r such that n = 2^d * r + 1 for some r >= 1
        long long d = n - 1;
        while (d % 2 == 0)
                d /= 2;

        // Iterate given nber of 'k' times
        for (long i = 0; i < k; i++)
                if (miillerTest(d, n) == false)
                        return false;

        return true;
}

// Driver program
int main(){
        long k = 4; // Number of iterations

        long long n = 982451653;
        cout << isPrime(n, k) << endl;

        return 0;
}
```

6.3 Primes IsPrime

```
import java.math.BigInteger;
import java.util.Scanner;

public class prime {

        public static void main(String[] args) {
                BigInteger a = new BigInteger("1299827");
                //User miller rabin & Lucas Lehmer
                boolean res = a.isProbablePrime(10);
                System.out.println(res? "It's prime":"It's not
                    prime");
        }

}
```

6.4 Primes Sieve

```
#include <iostream>
#include <math.h>
#include <vector>
#define tam 1000

using namespace std;

typedef long long ll;
typedef vector< bool > vbool;

void show (vbool primes){
```

```
    int cap = primes.size();
    for(int i = 0; i< cap; i++){
        cout << i << " : " << primes[i] << endl;
    }
}
vbool sieve(ll n){
    vbool sieve (tam);
    for (int i = 0; i < tam; i++)
        sieve[i] = true;
    sieve [0] = sieve[1] = false;
    ll root = sqrt(n);
    for (int i = 2; i < root; i++){ //find primes
        if(sieve[i]){
            //removes all the multiples
            //of the current prime
            for (int k = i*i; k<= n; k+=i){
                sieve[k] = false;
            }
        }
    }

    return sieve;
}
int main(){
    //Initialize the array
    vbool primes = sieve(1000);
    show(primes);;
    primes.clear();
    return 0;
}
```

6.5 Primes PrimalyTest

```
#include <iostream>
#include <math.h>

using namespace std;
typedef long long ll;

bool is_prime(ll n){
    if (n < 2) return false;
    if (n < 4) return true;
    if (n % 2 == 0 || n % 3 == 0) return false;
    if (n < 25) return true;
    for(int i = 5; i*i <= n; i += 6){
        if(n % i == 0 || n % (i + 2) == 0)
            return false;
    }
    return true;
}

int main(){
    cout << is_prime(23234) << endl;
    cout << is_prime(2) << endl;
    cout << is_prime(7454) << endl;
    cout << is_prime(976) << endl;
    cout << is_prime(1973) << endl;
    return 0;

}
```

6.6 Primes Factorize

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

using namespace std;
const int MAXN = 1000000;
bool sieve[MAXN + 5];
vector <int> pri; //pri

void build_sieve(){
    memset(sieve, false, sizeof(sieve));
    sieve[0] = sieve[1] = true;
    for (int i = 2; i * i <= MAXN; i ++){
        if (!sieve[i]){
            for (int j = i * i; j <= MAXN; j += i){
                sieve[j] = true;
            }
        }
    }
    for (int i = 2; i <= MAXN; ++i){
        if (!sieve[i]) pri.push_back(i);
    }
}

vector <long long> fact(long long a){
    // Se asume que se tiene y
    // se llam  la funci n build_sieve()
    vector <long long> ans;
    long long b = a;
    for (int i = 0; 1LL * pri[i] * pri[i] <= a; ++i){
        int p = pri[i];
        while (b % p == 0){
            ans.push_back(p);
            b /= p;
        }
    }
    if (b != 1) ans.push_back(b);
    return ans;
}

int main(){
    build_sieve();
    long long num_to_fact;
    cin >> num_to_fact;
    vector < long long > vll = fact(num_to_fact);

    for (int x=0; x< vll.size(); x++){
        cout << vll[x] << " ";
    }
    cout << endl;
}
```

7 Search

8 Sorting

8.1 Sorting SelectionSort SelectionSortCPP

```
#include<iostream>
using namespace std;

int show(int array[], int length_array);
int sort(int array[], int length_array);


int show(int array[], int length_array){
    for (int index = 0 ; index < length_array; index++)
        cout << array[index] << " ";
```

```
    cout << endl;

}

int sort(int array[], int length_array){
    for (int index = 0; index < length_array; index++){
        int pos_smallest = index;
        for(int index_aux = index+1; index_aux < length_array;
            index_aux++){
            if (array[pos_smallest] > array[index_aux]){
                pos_smallest = index_aux;
            }
        }
        //Swap
        if( pos_smallest != index){
            int aux = array[index];
            array[index] = array[pos_smallest];
            array[pos_smallest] = aux;
        }
    }

}

int main(){
    int length_array = 7;
    int array [] = {58, -5, 10, 8, 78 ,234, 43};

    cout << "Original Array" << endl;
    show(array, length_array);

    sort(array, length_array);

    cout << "Sorted Array" << endl;
    show(array, length_array);
}
```

8.2 Sorting InsertionSort InsertionSortPYTHON

```
def show(array):
    for element in array:
        print(element, end = " ")
    print("")


def sort(array, length_array):
    for index in range(1,length_array):
        key = array[index]
        index_aux = index -1
        while index_aux >=0 and array[index_aux]>key:
            array[index_aux+1] = array[index_aux]
            index_aux = index_aux -1
        array[index_aux+1]=key


def main():
    array = [50, 885, 1 , -8 , 54 , 2, 54, 0]
    print("Original array")
    show(array)
    print("Sorted array")
    sort(array, len(array))
    print(array)
main()
```

8.3 Sorting InsertionSort InsertionSortCPP

```
#include <iostream>

using namespace std;

void show(int array[], int length_array);
void sort(int array[], int length_array);

void show(int array[], int length_array){
    for (int index = 0; index < length_array; index ++)
        cout << array[index] << " ";
    cout<< endl;
}


void sort(int array[], int length_array){
    for (int index = 1; index < length_array; index ++){
        int key = array[index];
        int index_aux = index - 1;
        while (index_aux >= 0 && array[index_aux] > key){
            array[index_aux + 1 ] = array[index_aux];
            index_aux = index_aux -1;
        }
        array[index_aux + 1] = key;
    }
}

int main(){
    int length_array = 8;
    int array[]  = {50, 885, 1 , -8 , 54 , 2, 54, 0};
    show(array, length_array);
    sort(array, length_array);
    show(array, length_array);
}
```

8.4 Sorting BubbleSort Bubble

```
#include <bits/stdc++.h>
#define forn(i,j,k) for (int i=j; i<k; i++)
using namespace std;

typedef long long ll;

inline void sort(ll *arr, int size){
    forn(i,0,size-1)
        forn(j,0, size-i-1)
            if (arr[j] > arr[j+1])
                swap(arr[j], arr[j+1]);
}


int main(){
    ios::sync_with_stdio(false);
    cin.tie(0);

    int size =8;
    ll *data  = new ll[size];

    forn(i, 0, size)
        scanf("%lld", &data[i]);

    sort(data, size);

    forn(i, 0, size)
        printf("%lld ",data[i]);


    return 0;
}
```

8.5 Sorting MergeSort MergeSortCPP

```
#include <stdio.h>
#include <iostream>
#include <vector>

using namespace std;

void show(int array[], int length_array);
void sort(int array[], int pos_ini, int pos_final);
void merge(int array[], int pos_ini, int pos_mid,int pos_final);


void show(int array [], int length_array){
    int index = 0;
    // cout<< "size : "<< array.size() <<endl;
    while(index<length_array){
        printf("%d ",array[index]);
        index = index +1;
    }
    printf("\n");
}

void sort(int array[], int pos_ini, int pos_final){

    /*In this condition the len of the array
      left and right half arrays will be
      of 1 element both */
    if(pos_final > pos_ini){
         //first calc the half point
        int pos_mid = (pos_ini+pos_final)/2;

        sort(array, pos_ini, pos_mid);
        sort(array, pos_mid +1, pos_final);
        merge(array, pos_ini, pos_mid, pos_final);

    }
}

void merge( int array[], int pos_ini, int pos_mid,int pos_final){


    int size_left = pos_mid - pos_ini + 1;
    int size_right =  pos_final - pos_mid;

    /* create temp arrays */
    int lefthalf[size_left], righthalf[size_right];

    for (int i = 0; i < size_left; i++)
        lefthalf[i] = array[pos_ini + i];
    for (int j = 0; j < size_right; j++)
        righthalf[j] = array[pos_mid + 1+ j];


    int index_right_half  = 0;
    int index_left_half = 0;
    int index = pos_ini;
    while (index_left_half < size_left && index_right_half <
        size_right) {

        if(lefthalf[index_left_half] <= righthalf[index_right_half
            ]){
            array[index] = lefthalf[index_left_half];
            index_left_half = index_left_half +1;
        }else{
            array[index] = righthalf[index_right_half];
            index_right_half  = index_right_half+1;
        }
        index = index +1;
    }
    //Copy the remaining elements if there is any
    while( index_left_half < size_left){
        array[index] =  lefthalf[index_left_half];
        index_left_half = index_left_half +1;
        index = index +1;
    }
    while( index_right_half < size_right){
        array[index] =  righthalf[index_right_half];
        index_right_half = index_right_half +1;
        index = index +1;
    }
}

int main(){
    int array[] = {-10, 37, 98 , 0 ,12, 192, 5};
    int length_array = sizeof(array)/ sizeof(array[0]);
    show(array, length_array);
    sort(array, 0, length_array  -1);
    show(array, length_array);
}
```

8.6 Sorting MergeSort MergeSortPY

```
def merge_sort(array):
    ##Stop when the len of the array is less or equal than one
    if len(array)>1:
        #Calc the mid of the array
        mid = len(array) // 2 # // mean integer division

        #Create two arrays left and right
        lefthalf = array[:mid]
        print(lefthalf)

        righthalf = array[mid:]
        print(righthalf)

        #Divide the subarrays left and right
        merge_sort(lefthalf)
        merge_sort(righthalf)

        ##I send the array as a argument to change the same array
            and not another
        merge(lefthalf, righthalf, array)

def merge(lefthalf, righthalf, array):
    index_array_left=0
    index_array_right=0
    k=0
    while index_array_left < len(lefthalf) and \
          index_array_right < len(righthalf):

        if lefthalf[index_array_left] < \
            righthalf[index_array_right]:

            #assign the less to the new array
            array[k]=lefthalf[index_array_left]

            """As the less was an element in the lefthalf we dont
            need to compare this again so we increase the index
            of the left array"""
            index_array_left=index_array_left+1

        else:
            array[k]=righthalf[index_array_right]

            """As the less was an element in the righthalf we dont
            need to compare this again so we increase the index
            of the left array"""
```

```
            index_array_right=index_array_right+1

        #It is necesary increase the pos of the original array
        k=k+1

    ##add the remaining elements
    while index_array_left < len(lefthalf):
        array[k]=lefthalf[index_array_left]
        index_array_left=index_array_left+1
        k=k+1

    while index_array_right < len(righthalf):
        array[k]=righthalf[index_array_right]
        index_array_right=index_array_right+1
        k=k+1

def main():
    array = [-10, 37, 98 , 0 ,12, 192, 5]
    print("Original Array")
    print(array)
    merge_sort(array)

    print("Sorted Array")
    print(array)
main()
```

8.7 Sorting StandardSort

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

using namespace std;

typedef long long ll;
typedef vector < ll > vl;
int main(){
    vl data = {234234LL, 2322LL,1LL, -1LL, 3454LL};

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

    for (int i=0; i< data.size(); i++)
        printf("%lld ", data[i]);
    return 0;
}
```

9 Strings

9.1 Strings Palindrome

```
#include <iostream>
#include <string>

using namespace std;

/*
 * i,j positions letters in the word
 */
inline bool evaluate(string word, int i, int j){
    if (i >= j)
        return true;
    else if (word[i] != word[j])
        return false;
    return evaluate(word, i+1, j-1);
}

inline bool is_palindrome(string word){
    int length = word.length();
    if (length == 1)
        return true;
    return evaluate(word, 0, length-1);
}

int main(){
    string word = "anamariaairamana";
    string word2 = "Thisssiss";
    cout << word << " ";
    cout << is_palindrome(word) << endl;
    cout << word2 << " ";
    cout << is_palindrome(word2) << endl;
    return  0;
}
```

9.2 Strings FunctionsOverChart

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

int main(){
  char a = 'a';
  cout << (isalnum(a)?"true":"false")  << endl;
  cout <<( isalpha(a)?"true":"false") << endl;
  cout << (isblank(a)?"true":"false")  << endl;
  cout << (isdigit(a)?"true":"false")  << endl;
  cout << (islower(a)?"true":"false")  << endl;
  cout << (ispunct(a)?"true":"false")  << endl;
  cout << (isupper(a)?"true":"false")  << endl;
  cout << (isxdigit(a) ?"true":"false") << endl;
  cout << (char)tolower(a) << endl;
  cout << (char)toupper(a) << endl;
  return 0;
}
```

9.3 Strings Split

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

using namespace std;
/*
 * Split by space
 */
int main(){
    string line;
    while(getline(cin, line)){
        stringstream ss;
        ss.str(line);
        string item;
        while (getline(ss, item,' ')) {
            cout << item << endl;
        }
    }
    return 0;
}
```

9.4 Strings Regex

```
#include <iostream>
#include <iterator>
#include <regex>
#include <string>
```

```
using namespace std;
int main(){
    string  s = "123daniel , jajaja, lol, 234234534, I am from
        Earth";
    regex tel("\\d{8},\\sI");

    auto words_begin = sregex_iterator(s.begin(), s.end(), tel);
    auto words_end = sregex_iterator();

    cout << "Found " << distance(words_begin, words_end)<< " words
        \n";

    const int N = 6;

    for (sregex_iterator i = words_begin; i != words_end; ++i) {
        smatch match = *i;
        string match_str = match.str();
        if (match_str.size() > N) {
            cout << "   " << match_str << '\n';
        }
    }

    return 0;
}
```

10 Math

10.1 Math NumberTheory GCD$_LCM$

```
#include<cstdio>
using namespace std;

int gcd(int a, int b){
    if(b == 0) return a;
    return gcd(b, a % b);
}

int lcm(int n1, int n2){
    return (n1 * n2) / gcd(n1,n2);
}

int main(){
    int n1=2366, n2=273;
    printf("gcd(%ld, %ld) = %ld\n",
            n1, n2, gcd(n1,n2));
    return 0;
}
```

10.2 Math NumberTheory DivisorsCPP

```
#include <algorithm>
#include <math.h>
#include <set>
#include <stdio.h>
using namespace std;

typedef set<int> si;
/* Get the divisors of a number */
si divisores(int n) {
    si d;
    int r = sqrt(n);
    for(int i = 1; i <= r; i++) {
        if(n % i == 0) {
            d.insert(i);
            d.insert(n / i);
        }
    }
    return d;
}

int main() {
    si divi = divisores(10);
    for (set<int>::iterator it=divi.begin();
            it!=divi.end(); ++it)
        printf("%d ", *it);
    printf("\n");
}
```

10.3 Math NumberTheory DivisorsPYTHON

```
import math
"""Get the divisors of a number"""
def listDivisors(n):
    divisors = set()
    lim = int(math.sqrt(n))

    for i in range(1, lim + 1):
        if n % i == 0:
            divisors.add(i)
            divisors.add(n // i)

    return divisors

def main():
    d = listDivisors(100)
    print(len(d))
    print(d)

main()
```

10.4 Math NumberTheory Josephus

```
//https://www.youtube.com/watch?v=uCsD3ZGzMgE
int j(int n, int k) {
  if (n == 1) return 0;
  if (n < k) return (j(n-1,k)+k)%n;
  int np = n - n/k;
  return k*((j(np,k)+np-n%k%np)%np) / (k-1);
}
```

10.5 Math Pow FastPow

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

using namespace std;

ll modular_pow(ll  base, int exponent, ll  modulus){
        ll result = 1;
        while (exponent > 0){
                /* if y is odd, multiply base with result */
                if (exponent & 1)
                        result = (result * base) % modulus;
                /* exponent = exponent/2 */
                exponent = exponent >> 1;
```

```
            /* base = base * base */
            base = (base * base) % modulus;
        }
        return result;
}

int main(){

}
```

10.6 Math NumberSystems ChangeBases

```
# coding=utf-8
""" CHANGE THE BASE OF A NUMBER
    ob -> origin base
    od -> destiny base
"""

chars = "0123456789ABCDEFGHIJKLMN OPQRSTUVWXYZ"

def changeBase(number, ob,od):
    if ob == 10:
        return tob(number, od)
    return tob(to10(number,ob),od)

""" FROM ANY BASE TO BASE 10
    b -> base of the number n
    pos -> location of a sub-number in n
"""
def to10(n, b, pos =0):
   if n == 0: return 0
   return (n % 10)* (b  ** pos) + to10(n / 10, b, pos+1)

"""FROM TEN BASE TO ANOTHER BASE"""
def tob(n, b):
    if n == 0: return ""
    return   tob(n // b, b) + chars[n % b]

def main():
    print ( tob(7,2))
    print ( tob(252,16))
    print ( tob(234,15))
    print ( to10(1000,2))
    print ( changeBase(111,2,10))
main()
```

11 Sequences

12 probability

13 Geometry

13.1 Geometry LineIntersect2

```
#include <iostream>
#include <stdio.h>
#include <vector>

using namespace std;

/*
 *BETTER VERSION
 * http://stackoverflow.com/questions/563198/how-do-you-detect-
     where-two-line-segments-intersect
 */

typedef long double ld;
typedef struct point {
    ld x;
    ld y;
} point;

typedef vector < point > vp;


/*
 * i ---> is the intersection
 */
bool get_line_intersection(point p0, point p1, point p2, point p3,
     point i){

    ld s1_x, s1_y, s2_x, s2_y;
    point AB, DC;

    AB.x = p1.x - p0.x; AB.y = p1.y -p0.y;
    DC.x = p3.x - p2.x; DC.y = p3.y - p2.y;

    ld s, t;

    s = (-AB.y * (p0.x - p2.x) + AB.x * (p0.y - p2.y))
        / (-DC.x * AB.y + AB.x * DC.y);
    t = ( DC.x * (p0.y - p2.y) - DC.y * (p0.x - p2.x))
        / (-DC.x * AB.y + AB.x * DC.y);

    if (s >= 0 && s <= 1 && t >= 0 && t <= 1){
        // Collision detected
        i.x = p0.x + (t * AB.x);
        i.y = p0.y + (t * AB.y);
        cout << " x = " << i.x << " y= "<< i.y << endl;
        return true;
    }

    return false; // No collision
}


int main(){

    vp p(4);
    point inter;

    //line 1
    p[0] = {0,1};
    p[1] = {2,3};

    //line 2
    p[2] = {3,0};
    p[3] = {0,3};


    bool is = get_line_intersection(p[0], p[1], p[2], p[3], inter)
        ;
    printf("%s\n", is ? "Las lineas chocan": " There is not
        collision");

    return 0;

}
```

13.2 Geometry PointInteriorBoundary

```
#include <iostream>
#include <algorithm>
#include <stdlib.h>      /* srand, rand */
#include <time.h>        /* time */
#include <vector>
#include <math.h>

#define magnitude(a) (sqrt(a.x*a.x + a.y*a.y))

using namespace std;
/*
 * This script helps to find if a point is inside, outside
 * or in the boundaries of a poligon
 */

typedef long double ld;

typedef struct point {
    ld x;
    ld y;
}point;

typedef struct vert {
    point o; //origin
    point d; //destiny
} vert;

typedef vector < point > verts;



/*
 * Check if a point is inside or outside of a figure
 */


/*Cross product*/
inline ld cross_product(point o, point d){
    ld cross  = (o.x * d.y)  - ( o.y * d.x);
    return cross>0? cross: cross *-1;
}

inline  vert r2(point o, point d){
    return {o, d} ;
}

inline  point r(point o, point d){
    return {d.x-o.x, d.y - o.y} ;
}

ld dist_to_point(point A, point B, point C){
    //First create vector AB   and AC

    point AB = r(A,B);
    point AC = r(A,C);

    ld cross = cross_product(AB, AC);
    ld distance1 = cross / magnitude(AB);
    ld distance2 = cross / magnitude(AC);

    return min(distance1, distance2);

}
```

```
/*
 * i ---> is the intersection
 */

bool segments_intersect(vert v0, vert v1){

    point p0 = v0.o;
    point p1 = v0.d;
    point p2 = v1.o;
    point p3 = v1.d;
    point i;

    ld s1_x, s1_y, s2_x, s2_y;
    point AB, DC;

    AB.x = p1.x - p0.x; AB.y = p1.y -p0.y;
    DC.x = p3.x - p2.x; DC.y = p3.y - p2.y;

    ld s, t;

    s = (-AB.y * (p0.x - p2.x) + AB.x * (p0.y - p2.y)) / (-DC.x *
        AB.y + AB.x * DC.y);
    t = ( DC.x * (p0.y - p2.y) - DC.y * (p0.x - p2.x)) / (-DC.x *
        AB.y + AB.x * DC.y);

    if (s >= 0 && s <= 1 && t >= 0 && t <= 1){
        // Collision detected
        i.x = p0.x + (t * AB.x);
        i.y = p0.y + (t * AB.y);
    //    cout << " x = " << i.x << " y= "<< i.y << endl;
        return true;
    }

    return false; // No collision
}



inline void test_point(verts v, point pun){
    int cant = v.size();

    /*
     * Create a imaginary point to create a ray between the point
         and it.
     */

    point p;
    int cont = 0; // times that the point intersects
    p.x = rand() % 1000000 + 1;
    p.y = rand() % 1000000 + 1;

    if  ( cant > 0){
        for ( int i= 0 ; i < cant ; i ++ ){
            if (dist_to_point (v[i],v[(i+1)%cant], pun) == 0){
                cout << "The point is in the boundaries"<< endl;
            }
            if (segments_intersect(r2(v[i],v[(i+1)%cant]), r2(pun,
                 p))){
                cont  = cont +1;
            }
        }
    }
    if (cont % 2 == 0){
        cout << "The point is an exterior point " << endl;
    }else {
        cout << "The point is an interior point " << endl;
    }

}
```

```
int main(){

    /*The vertex of the polygon*/
    verts v(3);
    v[0] = {0,0};
    v[1] = {10,0};
    v[2] = {0,10};


    /* Point to check the program */

    point p1 = {4,5};
    point p2 = {5,5};

    test_point( v, p1);
    test_point( v, p2);

    return 0;

}
```

13.3 Geometry PickTheorem

```
#include <stdio.h>

using namespace std;

/*
 *  Pick's theorem is a useful method for determining the area of
     any polygon whose
 *  vertices are points on a lattice, a regularly spaced array of
     points.
 */

typedef double d;

/*
 * b boundary point : a lattice point on the polygon including
     vertices
 * i interior point : a lattice points on the polygon's interior
     region
 */
d area_poligon(d b, d i){
    return (b/2) + i -1;
}

int main(){
    printf("%f",area_poligon(5,5));
    return 0;
}
```

13.4 Geometry EulerFormule

```
#include <stdio.h>

using namespace std;

typedef long long ll;

b is_a_poligon(ll V, ll E, ll F){
   return V - E + F == 2;
}

int main(){
    printf("%s\n",
            is_a_poligon(3, 34,5) ? "true" : "false");
    //This is a cube
    printf("%s\n",
            is_a_poligon(8,12,6) ? "true" : "false");
    return 0;
}
```

13.5 Geometry LineIntersect1

```
#include <iostream>
#include <algorithm>
#include <limits>

using namespace std;

typedef long double ld;
typedef pair < ld, ld> point;
const ld INF = 9000000000000000000;

typedef struct line{
    // vy(y) + vx (x) = c; vy = 1
    ld vy;//value y
    ld vx;//value x
    ld c;// value coeficient
    point origin; //In the case of a segment
    bool segment;
    point destiny; //In the case of a segment
} line;

inline ld det(line l1, line l2){
    ld dete = (l1.vx * l2.vy ) - (l1.vy * l2.vx);
    return dete;
}

inline point intersect( line l1, line l2){
    //if det == 0 lines are parallel
    ld dete = det(l1,l2);
    if (dete != 0){
        //Apply crammer for compute the intersection
        ld x  = ((l1.c * l2.vy) - (l2.c * l1.vy )) / dete;
        ld y =  ((l1.vx * l2.c) - (l2.vx * l1.c)) / dete;
        return make_pair(x,y);
    }
    return make_pair(INF, INF);

}

/*
 *Return the negative slope
 */
inline ld slope(line l1){
    ld m = (l1.destiny.second - l1.origin.second ) / ( l1.destiny.
        first - l1.origin.first);
    return -1* m;
}

/*
 * Compute the independent coeffiecient
 */
inline ld coeficient(line l1){
    //y -mx = b
    ld c = l1.origin.second + l1.vx*l1.origin.first;
    return c;
}


/*
 * Based on the intersection and the lines check if
 * the intersection is valid, checking the boundaries
 */
```

```
inline bool mintersect(point inter, line l1, line l2){
  ld x= inter.first;
    ld y= inter.second;

    if( x != INF ){
        if(l1.segment && l2.segment){
            if( x >= min(l1.origin.first, l1.destiny.first) &&
                x >= min(l2.origin.first, l2.destiny.first) &&
                x <= max(l2.origin.first, l2.destiny.first) &&
                x <= max(l1.origin.first, l1.destiny.first) &&
                y >= min(l1.origin.second, l1.destiny.second) &&
                y >= min(l2.origin.second, l2.destiny.second) &&
                y <= max(l2.origin.second, l2.destiny.second) &&
                y <= max(l1.origin.second, l1.destiny.second)  ){

                cout << "x= " << x<< " y= " <<y <<endl;
                return true;
            }else{
                cout << "The lines  does not collide";
                return false;
            }
        }else{
            cout << "x= " << x<< " y= " <<y <<endl;
            return true;
        }
    }else{
        cout<<"Lines are parallel \n";
        return false;
    }

}

int main(){
    line l1, l2;
    l1 = l2 = {}; //its valid only with -std=c++11


    l1.origin = make_pair(0,0);
    l1.destiny = make_pair(-2,2);
    l1.vx  = slope(l1);
    l1.vy = 1;
    l1.segment =true;
    l1.c = coeficient(l1);

    l2.origin = make_pair(0,0);
    l2.destiny = make_pair(-3,2);
    l2.vx  = slope(l2);
    l2.vy = 1;
    l2.segment = true;
    l2.c = coeficient(l2);

    point inter = intersect(l1, l2);

    mintersect(inter, l1, l2);


}
```

13.6 Geometry Line2Point

```
#include<iostream>
#include<vector>
#include<math.h>
#include<algorithm>

#define magnitude(x) (sqrt(x.first*x.first + x.second*x.second))

using namespace std;

typedef long double ld;
typedef pair< ld, ld>  point;

/* Difference between two points */
inline  point r(point o, point d){
    return make_pair(d.first-o.first, d.second - o.second) ;
}

/*Cross product*/
inline ld cross_product(point o, point d){
    ld cross  = (o.first * d.second)  - ( o.second * d.first);
    return cross>0? cross: cross *-1;
}

/*
 *First find cross product
 */
ld distance(point A, point B, point C){
    //First create vector AB   and AC
    point AB = r(A,B);
    point AC = r(A,C);

    ld cross = cross_product(AB, AC);
    ld distance1 = cross / magnitude(AB);
    ld distance2 = cross / magnitude(AC);

    return min(distance1, distance2);
}

/*
 *Find the distance from the segment AB to C
 */

int main(){
    //Fast input and output
    ios::sync_with_stdio(false);
        cin.tie(NULL);

    point  A,B,C;

    A = make_pair(0,0);
    B = make_pair(0,2);
    C = make_pair(5,0);

    cout << distance(A,B,C);
    return 0;
}
```

13.7 Geometry PolygonArea

```
#include <iostream>
#include <vector>
#define f first
#define s s

using namespace std;

typedef long double ld;
typedef pair <ld, ld> point;
typedef vector < point > polygon;

inline  point r(point o, point d){
    return make_pair(d.f-o.f, d.s - o.s) ;
}

inline ld cross_product(point o, point d){
    ld cross  = (o.f * d.s)  - ( o.s * d.f);
    return cross > 0 ? cross :  cross * -1;
}
```

```
inline ld area(polygon p){

    int num_points = p.size();
    ld area = 0;
    for (int i = 1; i < num_points -1 ; i++){
        point l1 = r(p[0],p[i]);
        point l2 = r(p[0],p[i+1]);
        area += cross_product(l1,l2);
    }
    return area < 0 ?( -1 * area)/2:
                     (area/2 );
}

int main(){
    //Fast input and output
    ios::sync_with_stdio(false);
    cin.tie(NULL);

    point A, B, C, D;
    A = make_pair(1,0);
    B = make_pair(2,1);
    C = make_pair(1,2);
    D = make_pair(0,1);

    polygon p(4);
    p[0] = A;
    p[1] = B;
    p[2] = C;
    p[3] = D;

    cout << area(p);

    return 0;
}
```

13.8 Geometry ConvexHull

```
#include <iostream>
#include <math.h>
#include <vector>

using namespace std;
/* CONVEX HULL: Minimun Convex Polygon
 * Convex hull is the smallest set of points that containss tho
     set X.
 */


/*
 * Steps for the algorithm
 * 1- loop through all of the points and find the most left point
 *    if there is a tie, pick the highest point
 * 2- from the most left we are going to use cross product for
    finding
 *    the further clockwise from the current possition
 * 3- If there is no colinear points the code is straighforward.
 */

typedef long double ld;
const ld INF = 9000000000000000000;
typedef struct point {
    ld x;
    ld y;
} point;

typedef vector< point > points;
#define magnitud(p) (sqrt(p.x * p.x + p.y * p.y))
#define crossproduct(a,b) (a.x * b.y - a.y * b.x)
#define dist(a,b) (sqrt(pow((b.x - a.x),2) + pow((b.y * a.y),2)))
/*
 * Return the index of the most left point
 */
inline int left_most(points p){
    int cant = p.size();
    if(cant > 0){

        int ref = 0;
        for( int i = 1; i < cant; i++){
            if (p[i].x < p[ref].x ){
                ref = i;
            }
        }
        return ref;
    }
    return -1;
}

/* Create a vector based on two points
 */
inline point cv(point a, point b){
    return {b.x - a.x, b.y - a.y};
}

/*
 * lm -> stands out the left most point
 * more_points -> if is true use as many points as possible
 *                for the convex hull otherwise use as few
 *                as possible
 */
inline void convex_hull(points p, int lm, bool more_points){
    int cant = p.size();

    vector< bool > used (cant, false);
    /*
     * Just to clarify
     */
    int start = lm;


    do
    {
        int n = -1;
        ld dist = more_points?INF:0;


        cout << "Left most is " << lm << ": x= " << p[lm].x << " y
            = "<< p[lm].y << endl;

        for (int i = 0; i < cant ; i ++){

            //Do not go back to the same point
            if (i == lm) continue;

            //Do not reuse
            if(used[i])continue;


            //Set N
            if ( n==-1) {
                n =i;

                continue; //if I do not put this continue, the
                          //program will do a cross product
                          //with the same line
            }

            ld cross = crossproduct(cv(p[i],p[lm]), cv(p[n],p[lm])
                );
```

```
            ld d = dist(p[i], p[lm]);
            if (cross < 0 ){ //This is the magic
                n = i;
                dist = d;
            }else if (cross == 0){
                //In this case, both N and X are in the
                //same direction.  If more_points is true, pick
                    the
                //closest one, otherwise pick the farthest one.
                if(more_points && d < dist){
                    dist = d;
                    n = i;
                }else if(!more_points && d > dist){
                    dist = d;
                    n = i;
                }
            }

        }

        lm = n; //change the most left;
        used[lm] = true;
    }while(start != lm);
}

int main(){
    ios::sync_with_stdio(false);
    cin.tie(NULL);

    points p(6);

    p[0] = {0,2};
    p[1] = {3,5};
    p[2] = {4,3};
    p[3] = {3,0};
    p[4] = {3,3};
    p[5] = {4,6};
    convex_hull(p, left_most(p), false);

    return 0;

}
```

13.9 Geometry CircleCenter

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

using namespace std;
// Constants
const double PI = acos(-1);
struct point {
    double x;
    double y;
    point (){}
    point (double _x, double _y){
        x = _x;
        y = _y;
    }
};
inline point get_center(point A, point B, point C){

    float yDelta_a = B.y - A.y;
    float xDelta_a = B.x - A.x;
    float yDelta_b = C.y - B.y;
    float xDelta_b = C.x - B.x;

    point center;

    float aSlope = yDelta_a/xDelta_a;
    float bSlope = yDelta_b/xDelta_b;

    center.x = ( aSlope * bSlope * (A.y - C.y)
                + bSlope*(A.x + B.x)
                - aSlope*(B.x+C.x) )/(2* (bSlope-aSlope) );

    center.y = -1*(center.x - (A.x+B.x)/2)/aSlope +  (A.y+B.y)/2;

    return center;
}
```

14 Arrays

14.1 Arrays PermutationJAVA

```
public class Permutation{

    public static void main(String[] args){
        int[] array={3,5,0};
        permute(0, array);
    }

    public static void swap(int [] array, int i, int j){
        int temp = array[i];
        array[i] = array[j];
        array[j] = temp;
    }

    public static void show(int[] input){
        for(int x: input){
            System.out.print(x);
        }
        System.out.println("");
    }

    public  static void permute(int start, int[] input ) {
        if (start == input.length) {
            show(input);
            return;
        }
        for (int i = start; i < input.length; i++) {
            swap(input, i, start);
            permute(start + 1, input);
            swap(input, i, start);
        }
    }

}
```

14.2 Arrays Operations

```
from itertools import permutations, combinations

def pers(array):
    ps = permutations(array)
    for p in ps:
        print (p)

def combs(array, r=2):
    cmb = combinations(array, r)
    for c in cmb:
        print (c)
```

```
if __name__ == "__main__":
    pers([1,2,3,4,5])
    combs([1,2,3,4,5])
```

14.3 Arrays maximun$_{s}ubarray_{p}roblemKadane$

```
#include <bits/stdc++.h>
#define forn(i,j,k) for(int i=j; i<k; i++)
using namespace std;

typedef long long ll;

/*
 * Largest Sum Contiguous Subarray
 * Kadane Algorithm
 * Complexity O(n)
 */
inline ll get_max_sum(ll * data, int size){
    ll max_so_far= data[0];
    ll max_ending_here = data[0];
    forn(i, 1, size){
        max_ending_here = max(data[i], \
                data[i] + max_ending_here);
        max_so_far = max(max_so_far, max_ending_here);
    }
    return max_so_far;
}

int main(){
    //Faster input and output
    ios::sync_with_stdio(false);
    cin.tie(0);

    int size = 8;
    ll *data  = new ll[size];

    forn(i, 0, size)
        scanf("%lld", &data[i]);

    ll res = get_max_sum(data, size);
    printf("The max sum that can be done with \n \
            Contiguous elements is: %lld \n", res);

    return 0;
}
```

14.4 Arrays MapFunctions

```
"""
    Apply different function over an array
"""

def square(num):
    return num ** 2

def cube(num):
    return num ** 3

def is_pair(num):
    return num % 2

functions = [square, cube, is_pair]

array = range(0,20)

for elemn in array:
    value = map(lambda x: x(elemn), functions)
    print (elemn, value)
```

14.5 Arrays Combination

```
"""
    reference: http://www.geeksforgeeks.org/print-all-possible-
        combinations-of-r-elements-in-a-given-array-of-size-n/
"""

def combination(array, data, start, end, index, r):
    if (index == r):
        print (data)
        return
    for i in range(start, end+1):
        """ "end-i+1 >= r-index" makes sure that
             including one element  at index will
             make a combination with remaining
             elementsat remaining positions
        """
        if (end - i + 1 >= r - index):
            data[index] = array[i]
            combination(array, data, i+1, end, index + 1, r)

def get_combinations(array, r, n):
    combination(array, [0] * r, 0, n-1,  0, r)


if __name__ == "__main__":
    array = [0,1,2,3,4,5]
    r = 5
    n  = len(array)
    get_combinations(array, r, n)
```

14.6 Arrays PermutationCPP

```
#include <stdio.h>
#include <algorithm>
#include <iterator>
#include <vector>

using namespace std;

typedef vector <int > vi;

inline void show(vi &data, int &size){
    for (int i=0; i<size; i++)
            printf("%d \t", data[i]);
        printf("\n");
}

inline void permutation(vi data, int size){
    sort(data.begin(), data.end());
    do {
        show(data, size);
    }while(next_permutation(data.begin(), data.end()));
    show(data, size);
}

int main(){
    int size = 3 ;
    int data[] = {1,4,-1};
    vi vals(begin(data), end(data));
    permutation(vals, size);
    return 0;
}
```

15 Graphs

15.1 Graphs Traverse DFSJAVA

```
import java.util.*;

public class DFS {

    public static void dfs(node  start){
        ArrayDeque<node> s = new ArrayDeque<node>();

        s.push(start);
        while(s.isEmpty() == false){
            node top = s.pop();

            if(top.visited == false){
                top.visited = true;
                System.out.println("Visit " + top.name);
                ArrayList<node> n;
                n = top.neighbors;
                for (node a: n){
                    s.push(a);
                }
            }


        }
    }

    public static void main(String args[]){
        node a = new node("A");
        node b = new node("B");
        node c = new node("C");
        node d = new node("D");

        ArrayList<node> la = new ArrayList<node>();
        la.add(b);
        la.add(c);
        ArrayList<node> lc = new ArrayList<node>();
        lc.add(d);

        a.neighbors = la;
        c.neighbors = lc;

        dfs(a);
    }

}

class node {
   ArrayList<node> neighbors;
   String name;
   boolean visited ;
   public node(String name){
        this.name = name;
        this.visited = false;
        this.neighbors = new ArrayList<node>();
   }
};
```

15.2 Graphs Traverse DFSPY

```
class node:
    def __init__(self,n):
        self.neighbors = []
        self.visited = False
        self.name = n

def dfs(start):
    stack = []
    stack.append(start)

    while (stack != []):
        top = stack.pop()

        if (top.visited == False):

            top.visited = True
            print(top.name)
            #In this part in the termination condition
            n = top.neighbors
            for i in n:
                stack.append(i)


a = node("a")
b = node("b")
c = node("c")
d = node("d")

a.neighbors = [b,c]
c.neighbors = [d]
dfs(a)
```

15.3 Graphs Traverse DFSCPP

```
#include <bits/stdc++.h>
#define NUM_NODES 20

using namespace std;
vector < int > g[NUM_NODES];
int vis[NUM_NODES];
enum {WHITE, GRAY, BLACK};
void dfs(int o){
    vis [o] = GRAY; //semi-visited
    for (int i = 0; i < g[o].size(); i++){
        int v = g[o][i];
        if (vis[v] == GRAY)
            cout << "There is a cycle. to " << o <<  endl;
        // visit neighbors
        else if (vis[v] == WHITE) dfs(v);
    }
    cout << o << endl;
    vis[o] = BLACK; //visited;
}
int main(){

    g[0].push_back(1);
    g[0].push_back(2);
    g[0].push_back(3);
    g[1].push_back(4);
    g[1].push_back(5);
    g[2].push_back(6);
    g[3].push_back(7);
    g[4].push_back(0);
    g[6].push_back(0);

    dfs(0);
    return 0;
}
```

15.4 Graphs BestPath DijkstraHeap

```
#include <iostream>
#include <queue>
#include <vector>
using namespace std;

#define forn(i,a) for (int i=0; i<a ; i++)
#define INF 2e7

struct edge{
        int to, weight;
        edge(){}
        edge(int _to, int _weight){
                to = _to;
                weight = _weight;
        }
        bool operator < (edge e) const {
                return weight > e.weight;
        }
};

typedef vector < edge > ve;
typedef vector < ve > vve;
typedef vector < int > vi;
typedef priority_queue< edge> pq;

inline void show(vve &adj, int nodes){
    forn(i,nodes){
        cout << " Node:" << i << endl;
        forn (j, adj[i].size()){
            cout << "\t" << adj[i][j].to << endl;
        }
    }
}

inline void dijkstra(vve &adj, int src, int num_nodes){
    vi dist = vi(num_nodes+1,INF);
        pq  q;
    //by default
    q.push(edge(src,0));
    dist[src] = 0;
    //apply bfs
    while(!q.empty()){
        edge top = q.top();
        q.pop();
        int u = top.to;
        for(int i=0;i<adj[u].size();i++){
            int v = adj[u][i].to;
            if(dist[u] + adj[u][i].weight < dist[v]){
                dist[v] = dist[u] + adj[u][i].weight;
                q.push(edge(v,dist[v]));
            }
        }
    }
    //Show results of distances
    cout << "Distancias desde el origen ";
    cout << src << endl;
    forn(i, num_nodes){
        cout <<"Costo al nodo: " << i;
        cout << " ="<< dist[i] << endl;
    }
}

int main(){
    int nodes, vertex, from, to, weight;
    cin >> nodes >> vertex;
    vve adj(nodes);
    //Read the connections
    forn(i, vertex){
        cin >> from >> to >> weight;
        adj[from].push_back(edge(to, weight));
    }
    // show(adj, nodes);
    int src = 1;
    dijkstra(adj, src, nodes);
    return 0;
}
```

15.5 Graphs BestPath Dijkstra

```
#include <stdio.h>
#include <limits.h>
#define V 9

inline  int show_sol(int dist[], int n){
   printf("Vertex   Distance from Source\n");
   for (int i = 0; i < V; i++){
      printf("%d \t %d\n", i, dist[i]);
   }
}

int min_dis(int dist[], bool is_set[]){
   int min = INT_MAX, min_index;
   for (int v = 0; v < V; v++){
        if (is_set[v] == false && dist[v] <= min){
            min = dist[v], min_index = v;
        }
   }
   return min_index;
}

inline  void dijkstra(int graph[V][V], int src){
     int dist[V];
     bool is_set[V];
     for (int i = 0; i < V; i++){
        dist[i] = INT_MAX, is_set[i] = false;
     }
     dist[src] = 0;
     for (int count = 0; count < V-1; count++){
       int u = min_dis(dist, is_set);
       is_set[u] = true;

       for (int v = 0; v < V; v++){
         if (!is_set[v] && graph[u][v]
                 && dist[u] != INT_MAX
                 && dist[u]+graph[u][v] < dist[v])
            dist[v] = dist[u] + graph[u][v];
       }
     }
     show_sol(dist, V);
}
int main(){
   int graph[V][V] = {{0, 4, 0, 0, 0, 0, 0, 8, 0},
                      {4, 0, 8, 0, 0, 0, 0, 11, 0},
                      {0, 8, 0, 7, 0, 4, 0, 0, 2},
                      {0, 0, 7, 0, 9, 14, 0, 0, 0},
                      {0, 0, 0, 9, 0, 10, 0, 0, 0},
                      {0, 0, 4, 14, 10, 0, 2, 0, 0},
                      {0, 0, 0, 0, 0, 2, 0, 1, 6},
                      {8, 11, 0, 0, 0, 0, 1, 0, 7},
                      {0, 0, 2, 0, 0, 0, 6, 7, 0}
                     };
```

```
    //distances from all points to 1
    dijkstra(graph, 1);
    return 0;
}
```

15.6 Graphs BestPath BellmanFord

```
#include <cstdio>
#include <vector>
#define f first
#define s second
#define pb push_back
using namespace std;

#define MAX 2e9

typedef vector<int> vi;
typedef pair<int, int> pii;
typedef vector<pii> vpii;
typedef vector<vpii> vvpii;

void init(vi &distances, int s) {
    for(int i=0; i<distances.size(); i++)
        distances[i] = MAX;
    distances[s] = 0;
}

void bellman_ford(vvpii &graph, vi &dist) {
    for(int i=0; i<graph.size() - 1; i++) {
        for(int u = 0; u < graph.size(); u++) {
            for(pii v : graph[u]) {
                dist[v.f] =
                    min(dist[v.f], v.s + dist[u]);
            }
        }
    }
}

int main() {

    vvpii adjList(5);
    vi d(5);
    init(d, 0);
    adjList[0].pb({1, 6});
    adjList[0].pb({3, 7});
    adjList[1].pb({2, 5});
    adjList[1].pb({3, 8});
    adjList[1].pb({4, -4});
    adjList[2].pb({1, -2});
    adjList[3].pb({2, -3});
    adjList[3].pb({4, 9});
    adjList[4].pb({0, 2});
    adjList[4].pb({2, 7});

    bellman_ford(adjList, d);

    for(int i=0; i<d.size(); i++) {
        printf("%d ", d[i]);
    }
    printf("\n");

    return 0;
}
```

15.7 Graphs BestPath FloydWarshal

```
#include<iostream>
#include<stdio.h>
using namespace std;
/*
 * Floyd-Warshall gives us the shortest paths
 * from all sources to all target nodes.
 */
#define V 4  //number of vertex
#define INF 9999999

void print_sol(int dist[][V]){
    printf ("shortest distances \n");
    for (int i = 0; i < V; i++){
        for (int j = 0; j < V; j++){
            if (dist[i][j] == INF)
                printf("%7s", "INF");
            else
                printf ("%7d", dist[i][j]);
        }
        printf("\n");
    }
}
void floyd (int graph[][V]){
    int dist[V][V], i, j, k;
    for (i = 0; i < V; i++)
        for (j = 0; j < V; j++)
            dist[i][j] = graph[i][j];

    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];
            }
        }
    }
    print_sol(dist);
}
int main(){
    int graph[V][V] = { {0,   5,  INF, 10},
                        {INF, 0,   3, INF},
                        {INF, INF, 0,   1},
                        {INF, INF, INF, 0}
                      };
    floyd(graph);
    return 0;
}
```