Contents

1 Info About Memory and Time Limits

$O(f(n))$	Limite
$O(n!)$	$10, \ldots, 11$
$O(2^n n^2)$	$15, \ldots, 18$
$O(2^n n)$	$18, \ldots, 21$
$O(n^4)$	100
$O(n^3)$	500[1]
$O(n^2 \log^2 n)$	1000
$O(n^2 \log n)$	2000
$O(n^2)$	1e4[2]
$O(n \log^2 n)$	3e5
$O(n \log n)$	1e6
$O(n)$	1e8[3]

2 C++ Cheat Sheet

2.1 Headers

```cpp
#pragma GCC optimize("Ofast")
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;
typedef unsigned long long ull;
typedef pair<int, int> ii;
typedef tuple<int, int, int> iii;
typedef vector<int> vi;
typedef vector<ll> vll;
typedef vector<ii> vii;

typedef vector<vi> graph;
typedef vector<vii> wgraph;

#ifndef declaraciones_h
#define declaraciones_h

#define rep(i, n) for (int i = 0; i < (int)n; i++)
#define repx(i, a, b) for (int i = a; i < (int)b; i++)
#define invrep(i, a, b) for (int i = b; i-- > (int)a;)

#define pb push_back
#define eb emplace_back
#define ppb pop_back
```

[1] Este caso esta justo en el limite de tiempo, además en 256 MB cabe a los una matriz de 400^3 ints

[2] En general solo funciona hasta 6e3

[3] En general solo funciona hasta 4e7

```

#define lg(x) (31 - __builtin_clz(x))
#define lgg(x) (63 - __buitlin_clzll(x))
#define gcd __gcd

#define INF INT_MAX

#define umap unordered_map
#define uset unordered_set

#define debugx(x) cerr << #x << ": " << x << endl
#define debugv(v)          \
    cerr << #v << ":";     \
    for (auto e : v)       \
    {                      \
        cerr << " " << e; \
    }                      \
    cerr << endl
#define debugm(m)                                          \
    cerr << #m << endl;                                    \
    rep(i, (int)m.size())                                  \
    {                                                      \
        cerr << i << ":";                                  \
        rep(j, (int)m[i].size()) cerr << " " << m[i][j]; \
        cerr << endl;                                      \
    }
#define debugmp(m)                                                    \
    cerr << #m << endl;                                               \
    rep(i, (int)m.size())                                             \
    {                                                                 \
        cerr << i << ":";                                             \
        rep(j, (int)m[i].size())                                      \
        {                                                             \
            cerr << " {" << m[i][j].first << "," << m[i][j].second << "} \
                "; \
        }                                                             \
        cerr << endl;                                                 \
    }
#define print(x) copy(x.begin(), x.end(), ostream_iterator<int>(cout,
    "")), cout << endl

template <typename T1, typename T2>
ostream &operator<<(ostream &os, const pair<T1, T2> &p)
{
    os << '(' << p.first << ',' << p.second << ')';
    return os;
}

#endif
```

2.2 Cheat Sheet

```
#include "../headers/headers.h"

// Note: This Cheat Sheet is by no means complete
// If you want a thorough documentation of the Standard C++ Library
// please refer to this link: http://www.cplusplus.com/reference/

/* ================== */
/* Reading from stdin */
/* ================== */
// With scanf
scanf("%d", &a);            //int
scanf("%x", &a);            // int in hexadecimal
scanf("%llx", &a);          // long long in hexadecimal
scanf("%lld", &a);          // long long int
scanf("%c", &c);            // char
scanf("%s", buffer);        // string without whitespaces
scanf("%f", &f);            // float
scanf("%lf", &d);           // double
scanf("%d %*s %d", &a, &b); //* = consume but skip

// read until EOL
//  - EOL not included in buffer
//  - EOL is not consumed
//  - nothing is written into buffer if EOF is found
scanf(" %[^\n]", buffer);

//reading until EOL or EOF
//  - EOL not included in buffer
//  - EOL is consumed
//  - works with EOF
char *output = gets(buffer);
if (feof(stind))
{
} // EOF file found
if (output == buffer)
{
} // succesful read
if (output == NULL)
{
} // EOF found without previous chars found
//example
while (gets(buffer) != NULL)
{
    puts(buffer);
    if (feof(stdin))
    {
```

```
        break;
    }
}

// read single char
getchar();
while (true)
{
    c = getchar();
    if (c == EOF || c == '\n')
        break;
}

/* ================== */
/* Printing to stdout */
/* ================== */
// With printf
printf("%d", a);            // int
printf("%u", a);            // unsigned int
printf("%lld", a);          // long long int
printf("%llu", a);          // unsigned long long int
printf("%c", c);            // char
printf("%s", buffer);       // string until \0
printf("%f", f);            // float
printf("%lf", d);           // double
printf("%0*.*f", x, y, f);  // padding = 0, width = x, decimals = y
printf("(%.5s)\n", buffer); // print  at most the first five characters
    (safe to use on short strings)

// print at most first n characters (safe)
printf("(%.*s)\n", n, buffer); // make sure that n is integer (with long
    long I had problems)
//string + \n
puts(buffer);

/* ==================== */
/* Reading from c string */
/* ==================== */

// same as scanf but reading from s
int sscanf(const char *s, const char *format, ...);

/* ================== */
/* Printing to c string */
/* ================== */
// Same as printf but writing into str, the number of characters is
    returned
// or negative if there is failure
int sprintf(char *str, const char *format, ...);
//example:
int n = sprintf(buffer, "%d plus %d is %d", a, b, a + b);
printf("[%s] is a string %d chars long\n", buffer, n);

/* ==================== */
/* Peek last char of stdin */
/* ==================== */
bool peekAndCheck(char c)
{
    char c2 = getchar();
    ungetc(c2, stdin); // return char to stdin
    return c == c2;
}

/* ================ */
/* Reading from cin */
/* ================ */
// reading a line of unknown length
string line;
getline(cin, line);
while (getline(cin, line))
{
}

// Optimizations with cin/cout
ios::sync_with_stdio(0);
cin.tie(0);
cout.tie(0);

// Fix precision on cout
cout.setf(ios::fixed);
cout.precision(4); // e.g. 1.000

/* ==================== */
/* USING PAIRS AND TUPLES */
/* ==================== */
// ii = pair<int,int>
ii p(5, 5);
ii p = make_pair(5, 5)
    ii p = {5, 5};
int x = p.first, y = p.second;
// iii = tuple<int,int,int>
iii t(5, 5, 5);
tie(x, y, z) = t;
tie(x, y, z) = make_tuple(5, 5, 5);
get<0>(t)++;
get<1>(t)--;

/* ================================ */
/* CONVERTING FROM STRING TO NUMBERS */
/* ================================ */
//-----------------------
// string to int
// option #1:
int atoi(const char *str);
// option #2:
sscanf(string, "%d", &i);
//-------------------------
// string to long int:
// option #1:
long int strtol(const char *str, char **endptr, int base);
```

```cpp
// it only works skipping whitespaces, so make sure your numbers
// are surrounded by whitespaces only
// Example:
char szNumbers[] = "2001 60c0c0 -110111010011010010000 0x6fffff";
char *pEnd;
long int li1, li2, li3, li4;
li1 = strtol(szNumbers, &pEnd, 10);
li2 = strtol(pEnd, &pEnd, 16);
li3 = strtol(pEnd, &pEnd, 2);
li4 = strtol(pEnd, NULL, 0);
printf("The decimal equivalents are: %ld, %ld, %ld and %ld.\n", li1, li2
     , li3, li4);
// option #2:
long int atol(const char *str);
// option #3:
sscanf(string, "%ld", &l);
//--------------------------
// string to long long int:
// option #1:
long long int strtoll(const char *str, char **endptr, int base);
// option #2:
sscanf(string, "%lld", &l);
//--------------------------
// string to double:
// option #1:
double strtod(const char *str, char **endptr); //similar to strtol
// option #2:
double atof(const char *str);
// option #3:
sscanf(string, "%lf", &d);

/* ========================== */
/* C STRING UTILITY FUNCTIONS */
/* ========================== */
int strcmp(const char *str1, const char *str2);             // (-1,0,1)
int memcmp(const void *ptr1, const void *ptr2, size_t num); // (-1,0,1)
void *memcpy(void *destination, const void *source, size_t num);

/* ============================ */
/* C++ STRING UTILITY FUNCTIONS */
/* ============================ */
// read tokens from string
string s = "tok1 tok2 tok3";
string tok;
stringstream ss(s);
while (getline(ss, tok, ' '))
    printf("tok = %s\n", tok.c_str());

// split a string by a single char delimiter
void split(const string &s, char delim, vector<string> &elems)
{
    stringstream ss(s);
    string item;
    while (getline(ss, item, delim))
        elems.push_back(item);
}

// find index of string or char within string
string str = "random";
std::size_t pos = str.find("ra");
std::size_t pos = str.find('m');
if (pos == string::npos) // not found

    // substrings
    string subs = str.substr(pos, length);
string subs = str.substr(pos); // default: to the end of the string

// std::string from cstring's substring
const char *s = "bla1 bla2";
int offset = 5, len = 4;
string subs(s + offset, len); // bla2

// ------------------------
// string comparisons
int compare(const string &str) const;
int compare(size_t pos, size_t len, const string &str) const;
int compare(size_t pos, size_t len, const string &str,
            size_t subpos, size_t sublen) const;
int compare(const char *s) const;
int compare(size_t pos, size_t len, const char *s) const;

// examples
// 1) check string begins with another string
string prefix = "prefix";
string word = "prefix suffix";
word.compare(0, prefix.size(), prefix);

/* ==================== */
/*  OPERATOR OVERLOADING */
/* ==================== */

//--------------------------
// method #1: inside struct
struct Point
{
    int x, y;
    bool operator<(const Point &p) const
    {
        if (x != p.x)
            return x < p.x;
        return y < p.y;
    }
    bool operator>(const Point &p) const
    {
        if (x != p.x)
            return x > p.x;
        return y > p.y;
    }
    bool operator==(const Point &p) const
    {
```

```
        return x == p.x && y == p.y;
    }
};

//--------------------------
// method #2: outside struct
struct Point
{
    int x, y;
};
bool operator<(const Point &a, const Point &b)
{
    if (a.x != b.x)
        return a.x < b.x;
    return a.y < b.y;
}
bool operator>(const Point &a, const Point &b)
{
    if (a.x != b.x)
        return a.x > b.x;
    return a.y > b.y;
}
bool operator==(const Point &a, const Point &b)
{
    return a.x == b.x && a.y == b.y;
}

// Note: if you overload the < operator for a custom struct,
// then you can use that struct with any library function
// or data structure that requires the < operator
// Examples:
priority_queue<Point> pq;
vector<Point> pts;
sort(pts.begin(), pts.end());
lower_bound(pts.begin(), pts.end(), {1, 2});
upper_bound(pts.begin(), pts.end(), {1, 2});
set<Point> pt_set;
map<Point, int> pt_map;

/* ================== */
/* CUSTOM COMPARISONS */
/* ================== */
// method #1: operator overloading
// method #2: custom comparison function
bool cmp(const Point &a, const Point &b)
{
    if (a.x != b.x)
        return a.x < b.x;
    return a.y < b.y;
}
// method #3: functor
struct cmp
{
    bool operator()(const Point &a, const Point &b)
    {
        if (a.x != b.x)
            return a.x < b.x;
        return a.y < b.y;
    }
};
// without operator overloading, you would have to use
// an explicit comparison method when using library
// functions or data structures that require sorting
priority_queue<Point, vector<Point>, cmp> pq;
vector<Point> pts;
sort(pts.begin(), pts.end(), cmp);
lower_bound(pts.begin(), pts.end(), {1, 2}, cmp);
upper_bound(pts.begin(), pts.end(), {1, 2}, cmp);
set<Point, cmp> pt_set;
map<Point, int, cmp> pt_map;

/* ======================== */
/* VECTOR UTILITY FUNCTIONS */
/* ======================== */
vector<int> myvector;
myvector.push_back(100);
myvector.pop_back(); // remove last element
myvector.back();     // peek reference to last element
myvector.front();    // peek reference to first element
myvector.clear();    // remove all elements
// sorting a vector
vector<int> foo;
sort(foo.begin(), foo.end());
sort(foo.begin(), foo.end(), std::less<int>());    // increasing
sort(foo.begin(), foo.end(), std::greater<int>()); // decreasing

/* ==================== */
/* SET UTILITY FUNCTIONS */
/* ==================== */
set<int> myset;
myset.begin();  // iterator to first elemnt
myset.end();    // iterator to after last element
myset.rbegin(); // iterator to last element
myset.rend();   // iterator to before first element
for (auto it = myset.begin(); it != myset.end(); ++it)
{
    do_something(*it);
} // left -> right
for (auto it = myset.rbegin(); it != myset.rend(); ++it)
{
    do_something(*it);
} // right -> left
for (auto &i : myset)
{
    do_something(i);
} // left->right shortcut
auto ret = myset.insert(5);  // ret.first = iterator, ret.second =
    boolean (inserted / not inserted)
int count = mysert.erase(5); // count = how many items were erased
if (!myset.empty())
```

```
{
}
// custom comparator 1: functor
struct cmp
{
    bool operator()(int i, int j) { return i > j; }
};
set<int, cmp> myset;
// custom comparator 2: function
bool cmp(int i, int j) { return i > j; }
set<int, bool (*)(int, int)> myset(cmp);

/* ==================== */
/* MAP UTILITY FUNCTIONS */
/* ==================== */
struct Point
{
    int x, y;
};
bool operator<(const Point &a, const Point &b)
{
    return a.x < b.x || (a.x == b.x && a.y < b.y);
}
map<Point, int> ptcounts;

// --------------------
// inserting into map

// method #1: operator[]
// it overwrites the value if the key already exists
ptcounts[{1, 2}] = 1;

// method #2: .insert(pair<key, value>)
// it returns a pair { iterator(key, value) , bool }
// if the key already exists, it doesn't overwrite the value
void update_count(Point &p)
{
    auto ret = ptcounts.emplace(p, 1);
    // auto ret = ptcounts.insert(make_pair(p, 1)); //
    if (!ret.second)
        ret.first->second++;
}

// -------------------------
// generating ids with map
int get_id(string &name)
{
    static int id = 0;
    static map<string, int> name2id;
    auto it = name2id.find(name);
    if (it == name2id.end())
        return name2id[name] = id++;
    return it->second;
}

/* ======================== */
/* BITSET UTILITY FUNCTIONS */
/* ======================== */
bitset<4> foo; // 0000
foo.size();    // 4
foo.set();     // 1111
foo.set(1, 0); // 1011
foo.test(1);   // false
foo.set(1);    // 1111
foo.test(1);   // true

/* =============== */
/* RANDOM INTEGERS */
/* =============== */
#include <cstdlib>
#include <ctime>
srand(time(NULL));
int x = rand() % 100; // 0-99
int randBetween(int a, int b)
{ // a-b
    return a + (rand() % (1 + b - a));
}

/* ======= */
/* CLIMITS */
/* ======= */
#include <climits>
INT_MIN
INT_MAX
UINT_MAX
LONG_MIN
LONG_MAX
ULONG_MAX
LLONG_MIN
LLONG_MAX
ULLONG_MAX

/* ============== */
/* Bitwise Tricks */
/* ============== */

// amount of one-bits in number
int __builtin_popcount(int x);
int __builtin_popcountl(long x);
int __builtin_popcountll(long long x);

// amount of leading zeros in number
int __builtin_clz(int x);
int __builtin_clzl(long x);
int __builtin_clzll(ll x);

// binary length of non-negative number
int bitlen(int x) { return sizeof(x) * 8 - __builtin_clz(x); }
int bitlen(ll x) { return sizeof(x) * 8 - __builtin_clzll(x); }
```

```
// index of most significant bit
int log2(int x) { return sizeof(x) * 8 - __builtin_clz(x) - 1; }
int log2(ll x) { return sizeof(x) * 8 - __builtin_clzll(x) - 1; }

// reverse the bits of an integer
int reverse_bits(int x)
{
    int v = 0;
    while (x)
        v <<= 1, v |= x & 1, x >>= 1;
    return v;
}

// get string binary representation of an integer
string bitstring(int x)
{
    int len = sizeof(x) * 8 - __builtin_clz(x);
    if (len == 0)
        return "0";

    char buff[len + 1];
    buff[len] = '\0';
    for (int i = len - 1; i >= 0; --i, x >>= 1)
        buff[i] = (char)('0' + (x & 1));
    return string(buff);
}

/* ================== */
/* Hexadecimal Tricks */
/* ================== */

// get string hex representation of an integer
string to_hex(int num)
{
    static char buff[100];
    static const char *hexdigits = "0123456789abcdef";
    buff[99] = '\0';
    int i = 98;
    do
    {
        buff[i--] = hexdigits[num & 0xf];
        num >>= 4;
    } while (num);
    return string(buff + i + 1);
}

// ['0'-'9' 'a'-'f'] ->  [0 - 15]
int char_to_digit(char c)
{
    if ('0' <= c && c <= '9')
        return c - '0';
    return 10 + c - 'a';
}

/* ============ */
```

```
/* Other Tricks */
/* ============ */
// swap stuff
int x = 1, y = 2;
swap(x, y);

/* =========== */
/*    TIPS     */
/* =========== */
// 1) do not use .emplace(x, y) if your struct doesn't have an explicit
     constructor
//    instead you can use .push({x, y})
// 2) be careful while mixing scanf() with getline(), scanf will not
     consume \n unless
//    you explicitly tell it to do so (e.g scanf("%d\n", &x)) )
```

3 General Algorithms

3.1 Search

3.2 Brute Force

4 Data Structures

5 Dynamic Programming

6 Graphs

7 Mathematics

7.1 Modular Arithmetic

7.2 Primality Checks

7.3 Others

8 Geometry

8.1 Vectors/Points

```
#include "../../headers/headers.h"

const double PI = acos(-1);

struct vector2D
{
    double x, y;

```

```
    vector2D &operator+=(const vector2D &o)
    {
        this->x += o.x;
        this->y += o.y;
        return *this;
    }

    vector2D &operator-=(const vector2D &o)
    {
        this->x -= o.x;
        this->y -= o.y;
        return *this;
    }

    vector2D operator+(const vector2D &o)
    {
        return {x + o.x, y + o.y};
    }

    vector2D operator-(const vector2D &o)
    {
        return {x - o.x, y - o.y};
    }

    vector2D operator*(const double &o)
    {
        return {x * o, y * o};
    }

    bool operator==(const vector2D &o)
    {
        return x == o.x and y == o.y;
    }

    double norm2() { return x * x + y * y; }
    double norm() { return sqrt(norm2()); }
    double dot(const vector2D &o) { return x * o.x + y * o.y; }
    double cross(const vector2D &o) { return x * o.y - y * o.x; }
    double angle()
    {
        double angle = atan2(y, x);
        if (angle < 0)
            angle += 2 * PI;
        return angle;
    }

    vector2D Unit()
    {
        return {x / norm(), y / norm()};
    }
};

/* ======================================================== */
/* Cross Product -> orientation of vector2D with respect to ray */
/* ======================================================== */
// cross product (b - a) x (c - a)
ll cross(vector2D &a, vector2D &b, vector2D &c)
{
    ll dx0 = b.x - a.x, dy0 = b.y - a.y;
    ll dx1 = c.x - a.x, dy1 = c.y - a.y;
    return dx0 * dy1 - dx1 * dy0;
    // return (b - a).cross(c - a); // alternatively, using struct
        function
}

// calculates the cross product (b - a) x (c - a)
// and returns orientation:
// LEFT (1):      c is to the left of  ray (a -> b)
// RIGHT (-1):    c is to the right of ray (a -> b)
// COLLINEAR (0): c is collinear to    ray (a -> b)
// inspired by: https://www.geeksforgeeks.org/orientation-3-ordered-
    points/
int orientation(vector2D &a, vector2D &b, vector2D &c)
{
    ll tmp = cross(a, b, c);
    return tmp < 0 ? -1 : tmp == 0 ? 0 : 1; // sign
}

/* ======================================================== */
/* Check if a segment is below another segment (wrt a ray) */
/* ======================================================== */
// i.e: check if a segment is intersected by the ray first
// Assumptions:
// 1) for each segment:
//  p1 should be LEFT (or COLLINEAR) and p2 should be RIGHT (or
    COLLINEAR) wrt ray
// 2) segments do not intersect each other
// 3) segments are not collinear to the ray
// 4) the ray intersects all segments
struct Segment
{
    vector2D p1, p2;
};
#define MAXN (int)1e6   //Example
Segment segments[MAXN]; // array of line segments
bool is_si_below_sj(int i, int j)
{ // custom comparator based on cross product
    Segment &si = segments[i];
    Segment &sj = segments[j];
    return (si.p1.x >= sj.p1.x) ? cross(si.p1, sj.p2, sj.p1) > 0 : cross
        (sj.p1, si.p1, si.p2) > 0;
}
// this can be used to keep a set of segments ordered by order of
    intersection
// by the ray, for example, active segments during a SWEEP LINE
set<int, bool (*)(int, int)> active_segments(is_si_below_sj); // ordered
    set

/* ====================== */
/* Rectangle Intersection */
```

```
/* ======================= */
bool do_rectangles_intersect(vector2D &dl1, vector2D &ur1, vector2D &dl2
    , vector2D &ur2)
{
    return max(dl1.x, dl2.x) <= min(ur1.x, ur2.x) && max(dl1.y, dl2.y)
        <= min(ur1.y, ur2.y);
}

/* ========================= */
/* Line Segment Intersection */
/* ========================= */
// returns whether segments p1q1 and p2q2 intersect, inspired by:
// https://www.geeksforgeeks.org/check-if-two-given-line-segments-
    intersect/
bool do_segments_intersect(vector2D &p1, vector2D &q1, vector2D &p2,
    vector2D &q2)
{
    int o11 = orientation(p1, q1, p2);
    int o12 = orientation(p1, q1, q2);
    int o21 = orientation(p2, q2, p1);
    int o22 = orientation(p2, q2, q1);
    if (o11 != o12 and o21 != o22) // general case -> non-collinear
        intersection
        return true;
    if (o11 == o12 and o11 == 0)
    { // particular case -> segments are collinear
        vector2D dl1 = {min(p1.x, q1.x), min(p1.y, q1.y)};
        vector2D ur1 = {max(p1.x, q1.x), max(p1.y, q1.y)};
        vector2D dl2 = {min(p2.x, q2.x), min(p2.y, q2.y)};
        vector2D ur2 = {max(p2.x, q2.x), max(p2.y, q2.y)};
        return do_rectangles_intersect(dl1, ur1, dl2, ur2);
    }
    return false;
}

/* ================== */
/* Circle Intersection */
/* ================== */
struct Circle
{
    double x, y, r;
};
bool is_fully_outside(double r1, double r2, double d_sqr)
{
    double tmp = r1 + r2;
    return d_sqr > tmp * tmp;
}
bool is_fully_inside(double r1, double r2, double d_sqr)
{
    if (r1 > r2)
        return false;
    double tmp = r2 - r1;
    return d_sqr < tmp * tmp;
}
bool do_circles_intersect(Circle &c1, Circle &c2)
{
    double dx = c1.x - c2.x;
    double dy = c1.y - c2.y;
    double d_sqr = dx * dx + dy * dy;
    if (is_fully_inside(c1.r, c2.r, d_sqr))
        return false;
    if (is_fully_inside(c2.r, c1.r, d_sqr))
        return false;
    if (is_fully_outside(c1.r, c2.r, d_sqr))
        return false;
    return true;
}

/* ==================== */
/* vector2D - Line distance */
/* ==================== */
// get distance between p and projection of p on line <- a - b ->
double point_line_dist(vector2D &p, vector2D &a, vector2D &b)
{
    vector2D d = b - a;
    double t = d.dot(p - a) / d.norm2();
    return (a + d * t - p).norm();
}

/* ======================= */
/* vector2D - Segment distance */
/* ======================= */
// get distance between p and truncated projection of p on segment a ->
    b
double point_segment_dist(vector2D &p, vector2D &a, vector2D &b)
{
    if (a == b)
        return (p - a).norm(); // segment is a single vector2D
    vector2D d = b - a;        // direction
    double t = d.dot(p - a) / d.norm2();
    if (t <= 0)
        return (p - a).norm(); // truncate left
    if (t >= 1)
        return (p - b).norm(); // truncate right
    return (a + d * t - p).norm();
}

/* ================================== */
/* Straight Line Hashing (integer coords) */
/* ================================== */
// task: given 2 points p1, p2 with integer coordinates, output a unique
// representation {a,b,c} such that a*x + b*y + c = 0 is the equation
// of the straight line defined by p1, p2. This representation must be
// unique for each straight line, no matter which p1 and p2 are sampled.
struct Line
{
    int a, b, c;
};
int gcd(int a, int b)
{ // greatest common divisor
```

```
    a = abs(a);
    b = abs(b);
    while (b)
    {
        int c = a;
        a = b;
        b = c % b;
    }
    return a;
}
Line getLine(vector2D p1, vector2D p2)
{
    int a = p1.y - p2.y;
    int b = p2.x - p1.x;
    int c = p1.x * (p2.y - p1.y) - p1.y * (p2.x - p1.x);
    int sgn = (a < 0 || (a == 0 && b < 0)) ? -1 : 1;
    int f = gcd(a, gcd(b, c)) * sgn;
    a /= f;
    b /= f;
    c /= f;
    return {a, b, c};
}
```

8.2 Calculate Areas

8.2.1 Integration via Simpson's Method

```
#include "../../headers/headers.h"

//O(Evaluate f)=g(f)
//Numerical Integration of f in interval [a,b]
double simpsons_rule(function<double(double)> f, double a, double b)
{
    double c = (a + b) / 2;
    double h3 = abs(b - a) / 6;
    return h3 * (f(a) + 4 * f(c) + f(b));
}

//O(n g(f))
//Integrate f between a and b, using intervals of length (b-a)/n
double simpsons_rule(function<double(double)> f, double a, double b, int
      n)
{
    //n sets the precision for the result
    double ans = 0;
    double step = 0, h = (b - a) / n;
    rep(i, n)
    {
        ans += simpsons_rule(f, step, step + h);
        step += h;
    }
    return ans;
}
```

8.2.2 Green's Theorem

```
#include "../../headers/headers.h"

// O(1)
// Circle Arc
double arc(double theta, double phi)
{
}

// O(1)
// Line
double line(double x1, double y1, double x2, double y2)
{
}
```

9 Strings

9.1 Trie

```
#include "../../headers/headers.h"

/* Implementation from: https://pastebin.com/fyqsH65k */
struct TrieNode
{
    int leaf; // number of words that end on a TrieNode (allows for
         duplicate words)
    int height; // height of a TrieNode, root starts at height = 1, can
         be changed with the default value of constructor
    // number of words that pass through this node,
    // ask root node for this count to find the number of entries on the
          whole Trie
    // all nodes have 1 as they count the words than end on themselves (
         ie leaf nodes count themselves)
    int count;
    TrieNode *parent; // pointer to parent TrieNode, used on erasing
         entries
    map<char, TrieNode *> child;
    TrieNode(TrieNode *parent = NULL, int height = 1):
        parent(parent),
        leaf(0),
        height(height),
        count(0), // change to -1 if leaf nodes are to have count 0
             insead of 1
        child()
    {}
};

/**
 * Complexity: O(|key| * log(k))
 */
TrieNode *trie_find(TrieNode *root, const string &str)
{
    TrieNode *pNode = root;
    for (string::const_iterator key = str.begin(); key != str.end(); key
         ++)
```

```
    {
        if (pNode->child.find(*key) == pNode->child.end())
            return NULL;
        pNode = pNode->child[*key];
    }
    return (pNode->leaf) ? pNode : NULL; // returns only whole word
    // return pNode; // allows to search for a suffix
}

/**
 * Complexity: O(|key| * log(k))
 */
void trie_insert(TrieNode *root, const string &str)
{
    TrieNode *pNode = root;
    root -> count += 1;
    for (string::const_iterator key = str.begin(); key != str.end(); key
         ++)
    {
        if (pNode->child.find(*key) == pNode->child.end())
            pNode->child[*key] = new TrieNode(pNode, pNode->height + 1);
        pNode = pNode->child[*key];
        pNode -> count += 1;
    }
    pNode->leaf += 1;
}

/**
 * Complexity: O(|key| * log(k))
 */
void trie_erase(TrieNode *root, const string &str)
{
    TrieNode *pNode = root;
    string::const_iterator key = str.begin();
    for (; key != str.end(); key++)
    {
        if (pNode->child.find(*key) == pNode->child.end())
            return;
        pNode = pNode->child[*key];
    }
    pNode->leaf -= 1;
    pNode->count -= 1;
    while (pNode->parent != NULL)
    {
        if (pNode->child.size() > 0 || pNode->leaf)
            break;
        pNode = pNode->parent, key--;
        pNode->child.erase(*key);
        pNode->count -= 1;
    }
}
```

9.2 kmp

```
#include "../../headers/headers.h"

vi prefix(string &S)
{
    vector<int> p(S.size());
    p[0] = 0;
    for (int i = 1; i < S.size(); ++i)
    {
        p[i] = p[i - 1];
        while (p[i] > 0 && S[p[i]] != S[i])
            p[i] = p[p[i] - 1];
        if (S[p[i]] == S[i])
            p[i]++;
    }
    return p;
}

vi KMP(string &P, string &S)
{
    vector<int> pi = prefix(P);
    vi matches;
    int n = S.length(), m = P.length();
    int j = 0, ans = 0;
    for (int i = 0; i < n; ++i)
    {
        while (j > 0 && S[i] != P[j])
            j = pi[j - 1];
        if (S[i] == P[j])
            ++j;

        if (j == P.length())
        {
            /* This is where KMP found a match
             * we can calculate its position on S by using i - m + 1
             * or we can simply count it
             */
            ans += 1; // count the number of matches
            matches.eb(i - m + 1); // store the position of those
                matches
            // return; we can return on the first match if needed
            // this must stay the same
            j = pi[j - 1];
        }
    }
    return matches; // can be modified to return number of matches or
        location
}
```