```
##########Disjoint sets

#define REP(i, a, b) \ // all codes involving REP uses this macro
for (int i = int(a); i <= int(b); i++)
vector<int> pset(1000); // 1000 is just an initial number, it is user-adjustable.
void initSet(int _size) { pset.resize(_size); REP (i, 0, _size - 1) pset[i] = i; }
int findSet(int i) { return (pset[i] == i) ? i : (pset[i] = findSet(pset[i])); }
void unionSet(int i, int j) { pset[findSet(i)] = findSet(j); }
bool isSameSet(int i, int j) { return findSet(i) == findSet(j);}

########### Shortcuts

// Shortcuts for "common" data types in contests
typedef long long ll; // comments that are mixed in with code
typedef pair<int, int> ii; // are aligned to the right like this
typedef vector<ii> vii;
typedef vector<int> vi;
#define INF 1000000000 // 1 billion, safer than 2B for Floyd Warshall’s
// Common memset settings
memset(memo, -1, sizeof memo); // initialize DP memoization table with -1
memset(arr, 0, sizeof arr); // to clear array of integers
// We have abandoned the use of "REP" and "TRvii" since the second edition
// in order to reduce the confusion encountered by new programmers
//The following shortcuts are frequently used in both our C/C++ and Java code:
ans = a ? b : c; // to simplify: if (a) ans = b; else ans = c;
ans += val; // to simplify: ans = ans + val; and its variants
index = (index + 1) % n; // index++; if (index >= n) index = 0;
index = (index + n - 1) % n; // index--; if (index < 0) index = n - 1;
int ans = (int)((double)d + 0.5); // for rounding to nearest integer
ans = min(ans, new_computation); // min/max shortcut
//alternative form but not used in this book: ans <?= new_computation;
some code use short circuit && (AND) and || (OR)

######### Libraries
#include <bits/stdc++.h>
std::sync_with_studio(false);
$ g++ -std=c++11 myC.cc -o myC
$ ./myC <a.in >a.out
######### I/O

C/C++ Source Code                            | Sample Input | Sample Output
---------------------------------------------------------------------------
int TC, a, b;                                | 3            | 3
scanf("%d", &TC); // number of test cases    | 1 2          | 12
while (TC--) { // shortcut to repeat until 0 | 5 7          | 9
scanf("%d %d", &a, &b); // compute answer    | 6 3          |---------------
printf("%d\n", a + b); // on the fly         |--------------|
}                                            |              |
---------------------------------------------------------------------------
int a, b;                                    | 1 2          | 3
// stop when both integers are 0             | 5 7          | 12
while (scanf("%d %d", &a, &b), (a || b))     | 6 3          | 9
printf("%d\n", a + b);                       | 0 0          |---------------
---------------------------------------------------------------------------
int a, b;                                    | 1 2          | 3
// scanf returns the number of items read    | 5 7          | 12
while (scanf("%d %d", &a, &b) == 2)          | 6 3          | 9
// or you can check for EOF, i.e.            |--------------|---------------
while (scanf("%d %d", &a, &b) != EOF)        |              |
    printf("%d\n", a + b);                   |              |
---------------------------------------------------------------------------
int a, b, c = 1;                             | 1 2          | Case 1: 3
while (scanf("%d %d", &a, &b) != EOF)        | 5 7          |
// notice the two ‘\n’                       | 6 3          | Case 2: 12
printf("Case %d: %d\n\n", c++, a + b);       |--------------|
                                             |              | Case 3: 9
                                             |              |
                                             |              |---------------
---------------------------------------------------------------------------
int a, b, c = 1;                             | 1 2          | Case 1: 3
while (scanf("%d %d", &a, &b) != EOF) {      | 5 7          |
if (c > 1) printf("\n"); // 2nd/more cases   | 6 3          | Case 2: 12
    printf("Case %d: %d\n", c++, a + b);     |--------------|
}                                            |              | Case 3: 9
                                             |              |--------------
---------------------------------------------------------------------------
int k, ans, v;                               | 1 1          | 1
while (scanf("%d", &k) != EOF) {             | 2 3 4        | 7
  ans = 0;                                   | 3 8 1 1      | 10
  while (k--) { scanf("%d", &v); ans += v; } | 4 7 2 9 3    | 21
  printf("%d\n", ans);                       | 5 1 1 1 1 1  | 5
}                                            |--------------|--------------

####### DFS
typedef pair<int, int> ii; // In this chapter, we will frequently use these
typedef vector<ii> vii; // three data type shortcuts. They may look cryptic
typedef vector<int> vi; // but they are useful in competitive programming
vi dfs_num; // global variable, initially all values are set to UNVISITED
void dfs(int u) { // DFS for normal usage: as graph traversal algorithm
    dfs_num[u] = VISITED; // important: we mark this vertex as visited
    for (int j = 0; j < (int)AdjList[u].size(); j++) { // default DS: AdjList
        ii v = AdjList[u][j]; // v is a (neighbor, weight) pair
        if (dfs_num[v.first] == UNVISITED) // important check to avoid cycle
            dfs(v.first); // recursively visits unvisited neighbors of vertex u
} } // for simple graph traversal, we ignore the weight stored at v.second

####### BFS

// inside int main()---no recursion
vi d(V, INF); d[s] = 0; // distance from source s to s is 0
queue<int> q; q.push(s); // start from source
while (!q.empty()) {
    int u = q.front(); q.pop(); // queue: layer by layer!
    for (int j = 0; j < (int)AdjList[u].size(); j++) {
        ii v = AdjList[u][j]; // for each neighbor of u
        if (d[v.first] == INF) { // if v.first is unvisited + reachable
            d[v.first] = d[u] + 1; // make d[v.first] != INF to flag it
            q.push(v.first); // enqueue v.first for the next iteration
} } }

####### Blood Fill
int dr[] = {1,1,0,-1,-1,-1, 0, 1}; // trick to explore an implicit 2D grid
int dc[] = {0,1,1, 1, 0,-1,-1,-1}; // S,SE,E,NE,N,NW,W,SW neighbors
int floodfill(int r, int c, char c1, char c2) { // returns the size of CC
    if (r < 0 || r >= R || c < 0 || c >= C) return 0; // outside grid
    if (grid[r][c] != c1) return 0; // does not have color c1
    int ans = 1; // adds 1 to ans because vertex (r, c) has c1 as its color
    grid[r][c] = c2; // now recolors vertex (r, c) to c2 to avoid cycling!
    for (int d = 0; d < 8; d++)
        ans += floodfill(r + dr[d], c + dc[d], c1, c2);
    return ans; // the code is neat due to dr[] and dc[]
}
####### Kruskall

// inside int main()
vector< pair<int, ii> > EdgeList; // (weight, two vertices) of the edge
for (int i = 0; i < E; i++) {
    scanf("%d %d %d", &u, &v, &w); // read the triple: (u, v, w)
    EdgeList.push_back(make_pair(w, ii(u, v))); } // (w, u, v)
sort(EdgeList.begin(), EdgeList.end()); // sort by edge weight O(E log E)
// note: pair object has built-in comparison function
int mst_cost = 0;
UnionFind UF(V); // all V are disjoint sets initially
for (int i = 0; i < E; i++) { // for each edge, O(E)
    pair<int, ii> front = EdgeList[i];
    if (!UF.isSameSet(front.second.first, front.second.second)) { // check
        mst_cost += front.first; // add the weight of e to MST
        UF.unionSet(front.second.first, front.second.second); // link them
} } // note: the runtime cost of UFDS is very light
// note: the number of disjoint sets must eventually be 1 for a valid MST
printf("MST cost = %d (Kruskal’s)\n", mst_cost);
```

```
####### Djikstra
vector<int> dist(V, INF); dist[s] = 0; // INF = 2.10^9 not MAX_INT to avoid overflow
priority_queue<ii, vector<ii>, greater<ii> > pq; pq.push(ii(0, s)); // sort by distance
while (!pq.empty()) { // main loop
    ii top = pq.top(); pq.pop(); // greedy: pick shortest unvisited vertex
    int d = top.first, u = top.second;
    if (d == dist[u]) // This check is important! We want to process vertex u only once but
    we can
        // actually enqueue u several times in priority_queue... Fortunately, other
        occurrences of u
        // in priority_queue will have greater distances and can be ignored (the overhead is
        small) :)
        TRvii (AdjList[u], it) { // all outgoing edges from u
            int v = it->first, weight_u_v = it->second;
            if (dist[u] + weight_u_v < dist[v]) { // if can relax
            dist[v] = dist[u] + weight_u_v; // relax
            pq.push(ii(dist[v], v)); // enqueue this neighbor
}   } }                          // regardless whether it is already in pq or not

####### LIS
#include <algorithm>
#include <cstdio>
#include <stack>
using namespace std;

#define MAX_N 100000

void print_array(const char *s, int a[], int n) {
  for (int i = 0; i < n; ++i) {
    if (i) printf(", ");
    else printf("%s: [", s);
    printf("%d", a[i]);
  }
  printf("]\n");
}

void reconstruct_print(int end, int a[], int p[]) {
  int x = end;
  stack<int> s;
  for (; p[x] >= 0; x = p[x]) s.push(a[x]);
  printf("[%d", a[x]);
  for (; !s.empty(); s.pop()) printf(", %d", s.top());
  printf("]\n");
}

int main() {
  int n = 11, A[] = {-7, 10, 9, 2, 3, 8, 8, 1, 2, 3, 4};
  int L[MAX_N], L_id[MAX_N], P[MAX_N];

  int lis = 0, lis_end = 0;
  for (int i = 0; i < n; ++i) {
    int pos = lower_bound(L, L + lis, A[i]) - L;
    L[pos] = A[i];
    L_id[pos] = i;
    P[i] = pos ? L_id[pos - 1] : -1;
    if (pos + 1 > lis) {
      lis = pos + 1;
      lis_end = i;
    }

    printf("Considering element A[%d] = %d\n", i, A[i]);
    printf("LIS ending at A[%d] is of length %d: ", i, pos + 1);
    reconstruct_print(i, A, P);
    print_array("L is now", L, lis);
    printf("\n");
  }

  printf("Final LIS is of length %d: ", lis);
  reconstruct_print(lis_end, A, P);
  return 0;
}
```

```
####### Algorithm

Non-modifying sequence operations:

all_of    Test condition on all elements in range (function template )
any_of    Test if any element in range fulfills condition (function template )
none_of    Test if no elements fulfill condition (function template )
for_each    Apply function to range (function template )
find    Find value in range (function template )
find_if    Find element in range (function template )
find_if_not    Find element in range (negative condition) (function template )
find_end    Find last subsequence in range (function template )
find_first_of    Find element from set in range (function template )
adjacent_find    Find equal adjacent elements in range (function template )
count    Count appearances of value in range (function template )
count_if    Return number of elements in range satisfying condition (function template )
mismatch    Return first position where two ranges differ (function template )
equal    Test whether the elements in two ranges are equal (function template )
is_permutation    Test whether range is permutation of another (function template )
search    Search range for subsequence (function template )
search_n    Search range for elements (function template )

Modifying sequence operations:

copy    Copy range of elements (function template )
copy_n    Copy elements (function template )
copy_if    Copy certain elements of range (function template )
copy_backward    Copy range of elements backward (function template )
move    Move range of elements (function template )
move_backward    Move range of elements backward (function template )
swap    Exchange values of two objects (function template )
swap_ranges    Exchange values of two ranges (function template )
iter_swap    Exchange values of objects pointed to by two iterators (function template )
transform    Transform range (function template )
replace    Replace value in range (function template )
replace_if    Replace values in range (function template )
replace_copy    Copy range replacing value (function template )
replace_copy_if    Copy range replacing value (function template )
fill    Fill range with value (function template )
fill_n    Fill sequence with value (function template )
generate    Generate values for range with function (function template )
generate_n    Generate values for sequence with function (function template )
remove    Remove value from range (function template )
remove_if    Remove elements from range (function template )
remove_copy    Copy range removing value (function template )
remove_copy_if    Copy range removing values (function template )
unique    Remove consecutive duplicates in range (function template )
unique_copy    Copy range removing duplicates (function template )
reverse    Reverse range (function template )
reverse_copy    Copy range reversed (function template )
rotate    Rotate left the elements in range (function template )
rotate_copy    Copy range rotated left (function template )
random_shuffle    Randomly rearrange elements in range (function template )
shuffle    Randomly rearrange elements in range using generator (function template )

Partitions:

is_partitioned    Test whether range is partitioned (function template )
partition    Partition range in two (function template )
stable_partition    Partition range in two - stable ordering (function template )
partition_copy    Partition range into two (function template )
partition_point    Get partition point (function template )

Sorting:

sort    Sort elements in range (function template )
stable_sort    Sort elements preserving order of equivalents (function template )
partial_sort    Partially sort elements in range (function template )
partial_sort_copy    Copy and partially sort range (function template )
is_sorted    Check whether range is sorted (function template )
```

```
is_sorted_until    Find first unsorted element in range (function template )
nth_element    Sort element in range (function template )

Binary search (operating on partitioned/sorted ranges):

lower_bound    Return iterator to lower bound (function template )
upper_bound    Return iterator to upper bound (function template )
equal_range    Get subrange of equal elements (function template )
binary_search    Test if value exists in sorted sequence (function template )

Merge (operating on sorted ranges):

merge    Merge sorted ranges (function template )
inplace_merge    Merge consecutive sorted ranges (function template )
includes    Test whether sorted range includes another sorted range (function template )
set_union    Union of two sorted ranges (function template )
set_intersection    Intersection of two sorted ranges (function template )
set_difference    Difference of two sorted ranges (function template )
set_symmetric_difference    Symmetric difference of two sorted ranges (function template )

Heap:

push_heap    Push element into heap range (function template )
pop_heap    Pop element from heap range (function template )
make_heap    Make heap from range (function template )
sort_heap    Sort elements of heap (function template )
is_heap    Test if range is heap (function template )
is_heap_until    Find first element not in heap order (function template )

Min/max:

min    Return the smallest (function template )
max    Return the largest (function template )
minmax    Return smallest and largest elements (function template )
min_element    Return smallest element in range (function template )
max_element    Return largest element in range (function template )
minmax_element    Return smallest and largest elements in range (function template )

Other:

lexicographical_compare    Lexicographical less-than comparison (function template )
next_permutation    Transform range to next permutation (function template )
prev_permutation    Transform range to previous permutation (function template )
```