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

class Main {                          // standard Java class name in UVa
OJ
  public static void main(String[] args) {
    BigInteger fac = BigInteger.ONE;
    for (int i = 2; i <= 25; i++)
      fac = fac.multiply(BigInteger.valueOf(i));   // it is in the
library!
    System.out.println(fac);
} }
```

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

int N;         // using global variables in contests can be a good
strategy
char x[110];  // make it a habit to set array size a bit larger than
needed

int main() {
  scanf("%d\n", &N);
  while (N--) {                  // we simply loop from N, N-1, N-2, ...,
0
    scanf("0.%[0-9]...\n", &x);   // `&' is optional when x is a char
array
                         // note: if you are surprised with the trick
above,
                      // please check scanf details in
www.cppreference.com
    printf("the digits are 0.%s\n", x);
} } // return 0;
```

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

int main() {
  int arr[5] = {7,7,7};   // initial size (5) and initial value
{7,7,7,0,0}
  vector<int> v(5, 5);    // initial size (5) and initial value
{5,5,5,5,5}

  printf("arr[2] = %d and v[2] = %d\n", arr[2], v[2]);           // 7 and
5

  for (int i = 0; i < 5; i++) {
    arr[i] = i;
    v[i] = i;
  }

  printf("arr[2] = %d and v[2] = %d\n", arr[2], v[2]);           // 2 and
2

  // arr[5] = 5;     // static array will generate index out of bound
error
  // uncomment the line above to see the error

  v.push_back(5);                          // but vector will resize
itself
  printf("v[5] = %d\n", v[5]);                                        //
5

  return 0;
}
```

```
#include <algorithm>
#include <cstdio>
#include <string>
#include <vector>
using namespace std;

typedef struct {
  int id;
  int solved;
  int penalty;
} team;

bool icpc_cmp(team a, team b) {
  if (a.solved != b.solved) // can use this primary field to decide
sorted order
    return a.solved > b.solved;   // ICPC rule: sort by number of problem
solved
  else if (a.penalty != b.penalty)       // a.solved == b.solved, but we
can use
                                       // secondary field to decide
sorted order
    return a.penalty < b.penalty;       // ICPC rule: sort by descending
penalty
  else                        // a.solved == b.solved AND a.penalty ==
b.penalty
    return a.id < b.id;                      // sort based on increasing
team ID
}

int main() {
  int *pos, arr[] = {10, 7, 2, 15, 4};
  vector<int> v(arr, arr + 5);          // another way to initialize
vector
  vector<int>::iterator j;

  // sort descending with vector
  sort(v.rbegin(), v.rend());        // example of using 'reverse
iterator'
  for (vector<int>::iterator it = v.begin(); it != v.end(); it++)
    printf("%d ", *it);                     // access the value of
iterator
  printf("\n");
  printf("==================\n");

  // sort descending with integer array
  sort(arr, arr + 5);                                          //
ascending
  reverse(arr, arr + 5);                                    // then
reverse
  for (int i = 0; i < 5; i++)
    printf("%d ", arr[i]);
  printf("\n");
  printf("==================\n");

  random_shuffle(v.begin(), v.end());          // shuffle the content
again
  for (vector<int>::iterator it = v.begin(); it != v.end(); it++)
    printf("%d ", *it);
  printf("\n");
  printf("==================\n");
  partial_sort(v.begin(), v.begin() + 2, v.end());     // partial_sort
demo
  for (vector<int>::iterator it = v.begin(); it != v.end(); it++)
    printf("%d ", *it);
  printf("\n");
  printf("==================\n");

  // sort ascending
  sort(arr, arr + 5);                                 // arr should be sorted
now
  for (int i = 0; i < 5; i++)                                  // 2, 4, 7, 10,
15
    printf("%d ", arr[i]);
  printf("\n");
  sort(v.begin(), v.end());                 // sorting a vector, same
output
  for (vector<int>::iterator it = v.begin(); it != v.end(); it++)
    printf("%d ", *it);
  printf("\n");
  printf("==================\n");

  // multi-field sorting example, suppose we have 4 ICPC teams
  team nus[4] = { {1, 1, 10},
                  {2, 3, 60},
                  {3, 1, 20},
                  {4, 3, 60} };

  // without sorting, they will be ranked like this:
  for (int i = 0; i < 4; i++)
    printf("id: %d, solved: %d, penalty: %d\n",
           nus[i].id, nus[i].solved, nus[i].penalty);

  sort(nus, nus + 4, icpc_cmp);          // sort using a comparison
function
  printf("==================\n");
  // after sorting using ICPC rule, they will be ranked like this:
  for (int i = 0; i < 4; i++)
    printf("id: %d, solved: %d, penalty: %d\n",
           nus[i].id, nus[i].solved, nus[i].penalty);
  printf("==================\n");

  // there is a trick for multi-field sorting if the sort order is
"standard"
  // use "chained" pair class in C++ and put the highest priority in
front
  typedef pair < int, pair < string, string > > state;
  state a = make_pair(10, make_pair("steven", "grace"));
  state b = make_pair(7, make_pair("steven", "halim"));
  state c = make_pair(7, make_pair("steven", "felix"));
  state d = make_pair(9, make_pair("a", "b"));
  vector<state> test;
  test.push_back(a);
  test.push_back(b);
  test.push_back(c);
  test.push_back(d);
  for (int i = 0; i < 4; i++)
    printf("value: %d, name1 = %s, name2 = %s\n", test[i].first,
    ((string)test[i].second.first).c_str(),
((string)test[i].second.second).c_str());
  printf("==================\n");
  sort(test.begin(), test.end());   // no need to use a comparison
function
  // sorted ascending based on value, then based on name1,
  // then based on name2, in that order!
  for (int i = 0; i < 4; i++)
    printf("value: %d, name1 = %s, name2 = %s\n", test[i].first,
      ((string)test[i].second.first).c_str(),
((string)test[i].second.second).c_str());
  printf("==================\n");

  // binary search using lower bound
  pos = lower_bound(arr, arr + 5, 7);                              //
found
  printf("%d\n", *pos);
  j = lower_bound(v.begin(), v.end(), 7);
  printf("%d\n", *j);

  pos = lower_bound(arr, arr + 5, 77);                         // not
found
  if (pos - arr == 5) // arr is of size 5 ->
                      //    arr[0], arr[1], arr[2], arr[3], arr[4]
                      // if lower_bound cannot find the required value,
                      //   it will set return arr index +1 of arr size,
i.e.
                      //   the 'non existent' arr[5]
                      // thus, testing whether pos - arr == 5 blocks
                      //   can detect this "not found" issue
    printf("77 not found\n");
  j = lower_bound(v.begin(), v.end(), 77);
  if (j == v.end()) // with vector, lower_bound will do the same:
                    //   return vector index +1 of vector size
                    // but this is exactly the position of vector.end()
                    //   so we can test "not found" this way
    printf("77 not found\n");
  printf("==================\n");

  // useful if you want to generate permutations of set
  next_permutation(arr, arr + 5); // 2, 4, 7, 10, 15 -> 2, 4, 7, 15, 10
  next_permutation(arr, arr + 5); // 2, 4, 7, 15, 10 -> 2, 4, 10, 7, 15
  for (int i = 0; i < 5; i++)
    printf("%d ", arr[i]);
  printf("\n");

  next_permutation(v.begin(), v.end());
  next_permutation(v.begin(), v.end());
  for (vector<int>::iterator it = v.begin(); it != v.end(); it++)
    printf("%d ", *it);
  printf("\n");
  printf("==================\n");

  // sometimes these two useful simple macros are used
  printf("min(10, 7) = %d\n", min(10, 7));
  printf("max(10, 7) = %d\n", max(10, 7));

  return 0;
}
```

```
// note: for example usage of bitset, see ch5_06_primes.cpp

#include <cmath>
#include <cstdio>
#include <stack>
using namespace std;

#define isOn(S, j) (S & (1 << j))
#define setBit(S, j) (S |= (1 << j))
#define clearBit(S, j) (S &= ~(1 << j))
#define toggleBit(S, j) (S ^= (1 << j))
#define lowBit(S) (S & (-S))
#define setAll(S, n) (S = (1 << n) - 1)

#define modulo(S, N) ((S) & (N - 1))   // returns S % N, where N is a
power of 2
#define isPowerOfTwo(S) (!(S & (S - 1)))
#define nearestPowerOfTwo(S) ((int)pow(2.0, (int)((log((double)S) /
log(2.0)) + 0.5)))
#define turnOffLastBit(S) ((S) & (S - 1))
#define turnOnLastZero(S) ((S) | (S + 1))
#define turnOffLastConsecutiveBits(S) ((S) & (S + 1))
#define turnOnLastConsecutiveZeroes(S) ((S) | (S - 1))

void printSet(int vS) {                         // in binary
representation
  printf("S = %2d = ", vS);
  stack<int> st;
  while (vS)
    st.push(vS % 2), vS /= 2;
  while (!st.empty())                         // to reverse the print
order
    printf("%d", st.top()), st.pop();
  printf("\n");
}

int main() {
  int S, T;

  printf("1. Representation (all indexing are 0-based and counted from
right)\n");
  S = 34; printSet(S);
  printf("\n");

  printf("2. Multiply S by 2, then divide S by 4 (2x2), then by 2\n");
  S = 34; printSet(S);
  S = S << 1; printSet(S);
  S = S >> 2; printSet(S);
  S = S >> 1; printSet(S);
  printf("\n");

  printf("3. Set/turn on the 3-th item of the set\n");
  S = 34; printSet(S);
  setBit(S, 3); printSet(S);
  printf("\n");

  printf("4. Check if the 3-th and then 2-nd item of the set is on?\n");
  S = 42; printSet(S);
  T = isOn(S, 3); printf("T = %d, %s\n", T, T ? "ON" : "OFF");
  T = isOn(S, 2); printf("T = %d, %s\n", T, T ? "ON" : "OFF");
  printf("\n");

  printf("5. Clear/turn off the 1-st item of the set\n");
  S = 42; printSet(S);
  clearBit(S, 1); printSet(S);
  printf("\n");

  printf("6. Toggle the 2-nd item and then 3-rd item of the set\n");
  S = 40; printSet(S);
  toggleBit(S, 2); printSet(S);
  toggleBit(S, 3); printSet(S);
  printf("\n");

  printf("7. Check the first bit from right that is on\n");
  S = 40; printSet(S);
  T = lowBit(S); printf("T = %d (this is always a power of 2)\n", T);
  S = 52; printSet(S);
  T = lowBit(S); printf("T = %d (this is always a power of 2)\n", T);
  printf("\n");

  printf("8. Turn on all bits in a set of size n = 6\n");
  setAll(S, 6); printSet(S);
  printf("\n");

  printf("9. Other tricks (not shown in the book)\n");
  printf("8 %c 4 = %d\n", '%', modulo(8, 4));
  printf("7 %c 4 = %d\n", '%', modulo(7, 4));
  printf("6 %c 4 = %d\n", '%', modulo(6, 4));
  printf("5 %c 4 = %d\n", '%', modulo(5, 4));
  printf("is %d power of two? %d\n", 9, isPowerOfTwo(9));
  printf("is %d power of two? %d\n", 8, isPowerOfTwo(8));
  printf("is %d power of two? %d\n", 7, isPowerOfTwo(7));
  for (int i = 0; i <= 16; i++)
    printf("Nearest power of two of %d is %d\n", i,
nearestPowerOfTwo(i));
  printf("S = %d, turn off last bit in S, S = %d\n", 40,
turnOffLastBit(40));
  printf("S = %d, turn on last zero in S, S = %d\n", 41,
turnOnLastZero(41));
  printf("S = %d, turn off last consectuve bits in S, S = %d\n", 39,
turnOffLastConsecutiveBits(39));
  printf("S = %d, turn on last consecutive zeroes in S, S = %d\n", 36,
turnOnLastConsecutiveZeroes(36));

  return 0;
}
```

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

int main() {
  stack<char> s;
  queue<char> q;
  deque<char> d;

  printf("%d\n", s.empty());                   // currently s is empty,
true (1)
  printf("==================\n");
  s.push('a');
  s.push('b');
  s.push('c');
  // stack is LIFO, thus the content of s is currently like this:
  // c <- top
  // b
  // a
  printf("%c\n", s.top());                                   // output
'c'
  s.pop();                                                  // pop
topmost
  printf("%c\n", s.top());                                   // output
'b'
  printf("%d\n", s.empty());        // currently s is not empty, false
(0)
  printf("==================\n");

  printf("%d\n", q.empty());             // currently q is empty, true
(1)
  printf("==================\n");
  while (!s.empty()) {                   // stack s still has 2 more
items
    q.push(s.top());                          // enqueue 'b', and then
'a'
    s.pop();
  }
  q.push('z');                                        // add one more
item
  printf("%c\n", q.front());                                  // prints
'b'
  printf("%c\n", q.back());                                   // prints
'z'

  // output 'b', 'a', then 'z' (until queue is empty), according to the
insertion order above
  printf("==================\n");
  while (!q.empty()) {
    printf("%c\n", q.front());                      // take the front
first
    q.pop();                             // before popping (dequeue-ing)
it
  }

  printf("==================\n");
  d.push_back('a');
  d.push_back('b');
  d.push_back('c');
  printf("%c - %c\n", d.front(), d.back());               // prints 'a -
c'
  d.push_front('d');
  printf("%c - %c\n", d.front(), d.back());               // prints 'd -
c'
  d.pop_back();
  printf("%c - %c\n", d.front(), d.back());               // prints 'd -
b'
  d.pop_front();
  printf("%c - %c\n", d.front(), d.back());               // prints 'a -
b'

  return 0;
}
```

```
#include <cstdio>
#include <map>
#include <set>
#include <string>
using namespace std;

int main() {
  char name[20];
  int value;
  // note: there are many clever usages of this set/map
  // that you can learn by looking at top coder's codes
  // note, we don't have to use .clear() if we have just initialized the
set/map
  set<int> used_values; // used_values.clear();
  map<string, int> mapper; // mapper.clear();

  // suppose we enter these 7 name-score pairs below
  /*
  john 78
  billy 69
  andy 80
  steven 77
  felix 82
  grace 75
  martin 81
  */
  mapper["john"] = 78;   used_values.insert(78);
  mapper["billy"] = 69;  used_values.insert(69);
  mapper["andy"] = 80;   used_values.insert(80);
  mapper["steven"] = 77; used_values.insert(77);
  mapper["felix"] = 82;  used_values.insert(82);
  mapper["grace"] = 75;  used_values.insert(75);
  mapper["martin"] = 81; used_values.insert(81);

  // then the internal content of mapper MAY be something like this:
  // re-read balanced BST concept if you do not understand this diagram
  // the keys are names (string)!
  //                        (grace,75)
  //           (billy,69)               (martin,81)
  //     (andy,80)   (felix,82)    (john,78)  (steven,77)

  // iterating through the content of mapper will give a sorted output
  // based on keys (names)
  for (map<string, int>::iterator it = mapper.begin(); it !=
mapper.end(); it++)
    printf("%s %d\n", ((string)it->first).c_str(), it->second);

  // map can also be used like this
  printf("steven's score is %d, grace's score is %d\n",
    mapper["steven"], mapper["grace"]);
  printf("==================\n");

  // interesting usage of lower_bound and upper_bound
  // display data between ["f".."m") ('felix' is included, martin' is
excluded)
  for (map<string, int>::iterator it = mapper.lower_bound("f"); it !=
mapper.upper_bound("m"); it++)
    printf("%s %d\n", ((string)it->first).c_str(), it->second);

  // the internal content of used_values MAY be something like this
  // the keys are values (integers)!
  //                 (78)
  //         (75)            (81)
  //     (69)    (77)    (80)    (82)

  // O(log n) search, found
  printf("%d\n", *used_values.find(77));
  // returns [69, 75] (these two are before 77 in the inorder traversal
of this BST)
  for (set<int>::iterator it = used_values.begin(); it !=
used_values.lower_bound(77); it++)
    printf("%d,", *it);
  printf("\n");
  // returns [77, 78, 80, 81, 82] (these five are equal or after 77 in
the inorder traversal of this BST)
  for (set<int>::iterator it = used_values.lower_bound(77); it !=
used_values.end(); it++)
    printf("%d,", *it);
  printf("\n");
  // O(log n) search, not found
  if (used_values.find(79) == used_values.end())
    printf("79 not found\n");

  return 0;
}
```

```
#include <cstdio>
#include <iostream>
#include <string>
#include <queue>
using namespace std;

int main() {
  int money;
  char name[20];
  priority_queue< pair<int, string> > pq;             // introducing
'pair'
  pair<int, string> result;

  // suppose we enter these 7 money-name pairs below
  /*
  100 john
  10 billy
  20 andy
  100 steven
  70 felix
  2000 grace
  70 martin
  */
  pq.push(make_pair(100, "john"));         // inserting a pair in O(log
n)
  pq.push(make_pair(10, "billy"));
  pq.push(make_pair(20, "andy"));
  pq.push(make_pair(100, "steven"));
  pq.push(make_pair(70, "felix"));
  pq.push(make_pair(2000, "grace"));
  pq.push(make_pair(70, "martin"));
  // priority queue will arrange items in 'heap' based
  // on the first key in pair, which is money (integer), largest first
  // if first keys tie, use second key, which is name, largest first

  // the internal content of pq heap MAY be something like this:
  // re-read (max) heap concept if you do not understand this diagram
  // the primary keys are money (integer), secondary keys are names
(string)!
  //                        (2000,grace)
  //           (100,steven)               (70,martin)
  //     (100,john)   (10,billy)     (20,andy)  (70,felix)

  // let's print out the top 3 person with most money
  result = pq.top();                // O(1) to access the top / max
element
  pq.pop();          // O(log n) to delete the top and repair the
structure
  printf("%s has %d $\n", ((string)result.second).c_str(), result.first);
  result = pq.top(); pq.pop();
  printf("%s has %d $\n", ((string)result.second).c_str(), result.first);
  result = pq.top(); pq.pop();
  printf("%s has %d $\n", ((string)result.second).c_str(), result.first);

  return 0;
}
```

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

typedef pair<int, int> ii;
typedef vector<ii> vii;

int main() {
  int V, E, total_neighbors, id, weight, a, b;
  int AdjMat[100][100];
  vector<vii> AdjList;
  priority_queue< pair<int, ii> > EdgeList;   // one way to store Edge
List

  // Try this input for Adjacency Matrix/List/EdgeList
  // Adj Matrix
  //   for each line: |V| entries, 0 or the weight
  // Adj List
  //   for each line: num neighbors, list of neighbors + weight pairs
  // Edge List
  //   for each line: a-b of edge(a,b) and weight
  /*
  6
    0  10    0    0 100    0
   10    0    7    0    8    0
    0    7    0    9    0    0
    0    0    9    0   20    5
  100    8    0   20    0    0
    0    0    0    5    0    0
  6
  2 2 10 5 100
  3 1 10 3 7 5 8
  2 2 7 4 9
  3 3 9 5 20 6 5
  3 1 100 2 8 4 20
  1 4 5
  7
  1 2 10
  1 5 100
  2 3 7
  2 5 8
  3 4 9
  4 5 20
  4 6 5
  */
  freopen("in_07.txt", "r", stdin);

  scanf("%d", &V);                         // we must know this size
first!
                     // remember that if V is > 100, try NOT to use
AdjMat!
  for (int i = 0; i < V; i++)
    for (int j = 0; j < V; j++)
      scanf("%d", &AdjMat[i][j]);

  printf("Neighbors of vertex 0:\n");
  for (int j = 0; j < V; j++)                                    //
O(|V|)
```

```
    if (AdjMat[0][j])
      printf("Edge 0-%d (weight = %d)\n", j, AdjMat[0][j]);

  scanf("%d", &V);
  AdjList.assign(V, vii()); // quick way to initialize AdjList with V
entries of vii
  for (int i = 0; i < V; i++) {
    scanf("%d", &total_neighbors);
    for (int j = 0; j < total_neighbors; j++) {
      scanf("%d %d", &id, &weight);
      AdjList[i].push_back(ii(id - 1, weight));    // some index
adjustment
    }
  }

  printf("Neighbors of vertex 0:\n");
  for (vii::iterator j = AdjList[0].begin(); j != AdjList[0].end(); j++)
    // AdjList[0] contains the required information
    // O(k), where k is the number of neighbors
    printf("Edge 0-%d (weight = %d)\n", j->first, j->second);

  scanf("%d", &E);
  for (int i = 0; i < E; i++) {
    scanf("%d %d %d", &a, &b, &weight);
    EdgeList.push(make_pair(-weight, ii(a, b))); // trick to reverse sort
order
  }

  // edges sorted by weight (smallest->largest)
  for (int i = 0; i < E; i++) {
    pair<int, ii> edge = EdgeList.top(); EdgeList.pop();
    // negate the weight again
    printf("weight: %d (%d-%d)\n", -edge.first, edge.second.first,
edge.second.second);
  }

  return 0;
}
```

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

typedef vector<int> vi;

// Union-Find Disjoint Sets Library written in OOP manner, using both
path compression and union by rank heuristics
class UnionFind {                                              // OOP
style
private:
  vi p, rank, setSize;                       // remember: vi is
vector<int>
  int numSets;
public:
  UnionFind(int N) {
    setSize.assign(N, 1); numSets = N; rank.assign(N, 0);
    p.assign(N, 0); for (int i = 0; i < N; i++) p[i] = i; }
  int findSet(int i) { return (p[i] == i) ? i : (p[i] = findSet(p[i])); }
  bool isSameSet(int i, int j) { return findSet(i) == findSet(j); }
  void unionSet(int i, int j) {
    if (!isSameSet(i, j)) { numSets--;
    int x = findSet(i), y = findSet(j);
    // rank is used to keep the tree short
    if (rank[x] > rank[y]) { p[y] = x; setSize[x] += setSize[y]; }
    else                   { p[x] = y; setSize[y] += setSize[x];
                             if (rank[x] == rank[y]) rank[y]++; } } }
  int numDisjointSets() { return numSets; }
  int sizeOfSet(int i) { return setSize[findSet(i)]; }
};

int main() {
  printf("Assume that there are 5 disjoint sets initially\n");
  UnionFind UF(5); // create 5 disjoint sets
  printf("%d\n", UF.numDisjointSets()); // 5
  UF.unionSet(0, 1);
  printf("%d\n", UF.numDisjointSets()); // 4
  UF.unionSet(2, 3);
  printf("%d\n", UF.numDisjointSets()); // 3
  UF.unionSet(4, 3);
  printf("%d\n", UF.numDisjointSets()); // 2
  printf("isSameSet(0, 3) = %d\n", UF.isSameSet(0, 3)); // will return 0
(false)
  printf("isSameSet(4, 3) = %d\n", UF.isSameSet(4, 3)); // will return 1
(true)
  for (int i = 0; i < 5; i++) // findSet will return 1 for {0, 1} and 3
for {2, 3, 4}
    printf("findSet(%d) = %d, sizeOfSet(%d) = %d\n", i, UF.findSet(i), i,
UF.sizeOfSet(i));
  UF.unionSet(0, 3);
  printf("%d\n", UF.numDisjointSets()); // 1
  for (int i = 0; i < 5; i++) // findSet will return 3 for {0, 1, 2, 3,
4}
    printf("findSet(%d) = %d, sizeOfSet(%d) = %d\n", i, UF.findSet(i), i,
UF.sizeOfSet(i));
  return 0;
}
```

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

typedef vector<int> vi;

class SegmentTree {         // the segment tree is stored like a heap
array
private: vi st, A;            // recall that vi is: typedef vector<int>
vi;
  int n;
  int left (int p) { return p << 1; }     // same as binary heap
operations
  int right(int p) { return (p << 1) + 1; }

  void build(int p, int L, int R) {                           // O(n log
n)
    if (L == R)                            // as L == R, either one is
fine
      st[p] = L;                                         // store the
index
    else {                                // recursively compute the
values
      build(left(p) , L              , (L + R) / 2);
      build(right(p), (L + R) / 2 + 1, R          );
      int p1 = st[left(p)], p2 = st[right(p)];
      st[p] = (A[p1] <= A[p2]) ? p1 : p2;
  } }

  int rmq(int p, int L, int R, int i, int j) {                  // O(log
n)
    if (i >  R || j <  L) return -1; // current segment outside query
range
    if (L >= i && R <= j) return st[p];               // inside query
range

     // compute the min position in the left and right part of the
interval
    int p1 = rmq(left(p) , L              , (L+R) / 2, i, j);
    int p2 = rmq(right(p), (L+R) / 2 + 1, R          , i, j);

    if (p1 == -1) return p2;   // if we try to access segment outside
query
    if (p2 == -1) return p1;                               // same as
above
    return (A[p1] <= A[p2]) ? p1 : p2; }          // as as in build
routine

  int update_point(int p, int L, int R, int idx, int new_value) {
    // this update code is still preliminary, i == j
    // must be able to update range in the future!
    int i = idx, j = idx;

    // if the current interval does not intersect
    // the update interval, return this st node value!
    if (i > R || j < L)
      return st[p];

    // if the current interval is included in the update range,
    // update that st[node]
    if (L == i && R == j) {
      A[i] = new_value; // update the underlying array
      return st[p] = L; // this index
    }

    // compute the minimum pition in the
    // left and right part of the interval
    int p1, p2;
    p1 = update_point(left(p) , L              , (L + R) / 2, idx,
new_value);
    p2 = update_point(right(p), (L + R) / 2 + 1, R          , idx,
new_value);

    // return the pition where the overall minimum is
    return st[p] = (A[p1] <= A[p2]) ? p1 : p2;
  }

public:
  SegmentTree(const vi &_A) {
    A = _A; n = (int)A.size();              // copy content for local
usage
    st.assign(4 * n, 0);            // create large enough vector of
zeroes
    build(1, 0, n - 1);                                  // recursive
build
  }

  int rmq(int i, int j) { return rmq(1, 0, n - 1, i, j); }   //
overloading

  int update_point(int idx, int new_value) {
    return update_point(1, 0, n - 1, idx, new_value); }
};

int main() {
  int arr[] = { 18, 17, 13, 19, 15, 11, 20 };         // the original
array
  vi A(arr, arr + 7);                      // copy the contents to a
vector
  SegmentTree st(A);

  printf("             idx    0, 1, 2, 3, 4,  5, 6\n");
  printf("             A is {18,17,13,19,15, 11,20}\n");
  printf("RMQ(1, 3) = %d\n", st.rmq(1, 3));             // answer = index
2
  printf("RMQ(4, 6) = %d\n", st.rmq(4, 6));             // answer = index
5
  printf("RMQ(3, 4) = %d\n", st.rmq(3, 4));             // answer = index
4
  printf("RMQ(0, 0) = %d\n", st.rmq(0, 0));             // answer = index
0
  printf("RMQ(0, 1) = %d\n", st.rmq(0, 1));             // answer = index
1
  printf("RMQ(0, 6) = %d\n", st.rmq(0, 6));             // answer = index
5

  printf("             idx    0, 1, 2, 3, 4,  5, 6\n");
  printf("Now, modify A into {18,17,13,19,15,100,20}\n");
  st.update_point(5, 100);                    // update A[5] from 11 to
100
  printf("These values do not change\n");
  printf("RMQ(1, 3) = %d\n", st.rmq(1, 3));                           //
2
  printf("RMQ(3, 4) = %d\n", st.rmq(3, 4));                           //
4
  printf("RMQ(0, 0) = %d\n", st.rmq(0, 0));                           //
0
  printf("RMQ(0, 1) = %d\n", st.rmq(0, 1));                           //
1
  printf("These values change\n");
  printf("RMQ(0, 6) = %d\n", st.rmq(0, 6));                         // 5-
>2
  printf("RMQ(4, 6) = %d\n", st.rmq(4, 6));                         // 5-
>4
  printf("RMQ(4, 5) = %d\n", st.rmq(4, 5));                         // 5-
>4

  return 0;
}
```

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

typedef vector<int> vi;
#define LSOne(S) (S & (-S))

class FenwickTree {
private:
  vi ft;

public:
  FenwickTree() {}
  // initialization: n + 1 zeroes, ignore index 0
  FenwickTree(int n) { ft.assign(n + 1, 0); }

  int rsq(int b) {                                     // returns RSQ(1,
b)
    int sum = 0; for (; b; b -= LSOne(b)) sum += ft[b];
    return sum; }

  int rsq(int a, int b) {                              // returns RSQ(a,
b)
    return rsq(b) - (a == 1 ? 0 : rsq(a - 1)); }

  // adjusts value of the k-th element by v (v can be +ve/inc or -ve/dec)
  void adjust(int k, int v) {                    // note: n = ft.size() -
1
    for (; k < (int)ft.size(); k += LSOne(k)) ft[k] += v; }
};

int main() {              // idx   0 1 2 3 4 5 6 7  8 9 10, no index 0!
  FenwickTree ft(10);     // ft = {-,0,0,0,0,0,0,0, 0,0,0}
  ft.adjust(2, 1);        // ft = {-,0,1,0,1,0,0,0, 1,0,0}, idx 2,4,8 =>
+1
  ft.adjust(4, 1);        // ft = {-,0,1,0,2,0,0,0, 2,0,0}, idx 4,8 => +1
  ft.adjust(5, 2);        // ft = {-,0,1,0,2,2,2,0, 4,0,0}, idx 5,6,8 =>
+2
  ft.adjust(6, 3);        // ft = {-,0,1,0,2,2,5,0, 7,0,0}, idx 6,8 => +3
  ft.adjust(7, 2);        // ft = {-,0,1,0,2,2,5,2, 9,0,0}, idx 7,8 => +2
  ft.adjust(8, 1);        // ft = {-,0,1,0,2,2,5,2,10,0,0}, idx 8 => +1
  ft.adjust(9, 1);        // ft = {-,0,1,0,2,2,5,2,10,1,1}, idx 9,10 =>
+1
  printf("%d\n", ft.rsq(1, 1));  // 0 => ft[1] = 0
  printf("%d\n", ft.rsq(1, 2));  // 1 => ft[2] = 1
  printf("%d\n", ft.rsq(1, 6));  // 7 => ft[6] + ft[4] = 5 + 2 = 7
  printf("%d\n", ft.rsq(1, 10)); // 11 => ft[10] + ft[8] = 1 + 10 = 11
  printf("%d\n", ft.rsq(3, 6));  // 6 => rsq(1, 6) - rsq(1, 2) = 7 - 1

  ft.adjust(5, 2); // update demo
  printf("%d\n", ft.rsq(1, 10)); // now 13
} // return 0;
```

```
/* 8 Queens Chess Problem */
#include <cstdlib>                       // we use the int version of
'abs'
#include <cstdio>
#include <cstring>
using namespace std;

int row[8], TC, a, b, lineCounter;            // ok to use global
variables

bool place(int r, int c) {
  for (int prev = 0; prev < c; prev++)    // check previously placed
queens
    if (row[prev] == r || (abs(row[prev] - r) == abs(prev - c)))
      return false;        // share same row or same diagonal ->
infeasible
  return true; }

void backtrack(int c) {
  if (c == 8 && row[b] == a) {         // candidate sol, (a, b) has 1
queen
    printf("%2d      %d", ++lineCounter, row[0] + 1);
    for (int j = 1; j < 8; j++) printf(" %d", row[j] + 1);
    printf("\n"); }
  for (int r = 0; r < 8; r++)                       // try all possible
row
    if (place(r, c)) {          // if can place a queen at this col and
row
      row[c] = r; backtrack(c + 1);      // put this queen here and
recurse
}   }

int main() {
  scanf("%d", &TC);
  while (TC--) {
    scanf("%d %d", &a, &b); a--; b--;         // switch to 0-based
indexing
    memset(row, 0, sizeof row); lineCounter = 0;
    printf("SOLN       COLUMN\n");
    printf(" #      1 2 3 4 5 6 7 8\n\n");
    backtrack(0);           // generate all possible 8! candidate
solutions
    if (TC) printf("\n");
} } // return 0;
```

```
/* UVa 11450 - Wedding Shopping - Top Down */
// this code is similar to recursive backtracking code
// parts of the code specific to top-down DP are commented with: `TOP-
DOWN'
// if these lines are commented, this top-down DP will become
backtracking!
#include <algorithm>
#include <cstdio>
#include <cstring>
using namespace std;

int M, C, price[25][25];                 // price[g (<= 20)][model (<=
20)]
int memo[210][25];    // TOP-DOWN: dp table memo[money (<= 200)][g (<=
20)]
int shop(int money, int g) {
  if (money < 0) return -1000000000;     // fail, return a large -ve
number
  if (g == C) return M - money;        // we have bought last garment,
done
  if (memo[money][g] != -1) return memo[money][g]; // TOP-DOWN:
memoization
  int ans = -1;   // start with a -ve number as all prices are non
negative
  for (int model = 1; model <= price[g][0]; model++)      // try all
models
    ans = max(ans, shop(money - price[g][model], g + 1));
  return memo[money][g] = ans; // TOP-DOWN: assign ans to table + return
it
}

/*

int shop(int money, int g) {
  if (money < 0) return -1000000000;
  if (g == C) return M - money;
  int &ans = memo[money][g];                 // remember the memory
address
  if (ans != -1) return ans;
  for (int model = 1; model <= price[g][0]; model++)
    ans = max(ans, shop(money - price[g][model], g + 1));
  return ans;                // ans (or memo[money][g]) is directly
updated
}

void print_shop(int money, int g) { // this function does not return
anything
  if (money < 0 || g == C) return;                      // similar base
cases
  for (int model = 1; model <= price[g][0]; model++)   // which model is
opt?
    if (shop(money - price[g][model], g + 1) == memo[money][g]) { // this
one
      printf("%d - ", price[g][model]);
      print_shop(money - price[g][model], g + 1); // recurse to this one
only
      break;
}   }

*/
```

```
int main() {            // easy to code if you are already familiar with
it
  int i, j, TC, score;

  scanf("%d", &TC);
  while (TC--) {
    scanf("%d %d", &M, &C);
    for (i = 0; i < C; i++) {
      scanf("%d", &price[i][0]);                  // store K in
price[i][0]
      for (j = 1; j <= price[i][0]; j++) scanf("%d", &price[i][j]);
    }
    memset(memo, -1, sizeof memo);    // TOP-DOWN: initialize DP memo
table
    score = shop(M, 0);                            // start the top-down
DP
    if (score < 0) printf("no solution\n");
    else           printf("%d\n", score);
} } // return 0;
```

```
/* UVa 11450 - Wedding Shopping - Bottom Up */
#include <cstdio>
#include <cstring>
using namespace std;

int main() {
  int i, j, k, TC, M, C;
  int price[25][25];                      // price[g (<= 20)][model (<=
20)]
  bool reachable[25][210];    // reachable table[g (<= 20)][money (<=
200)]
  scanf("%d", &TC);
  while (TC--) {
    scanf("%d %d", &M, &C);
    for (i = 0; i < C; i++) {
      scanf("%d", &price[i][0]);               // we store K in
price[i][0]
      for (j = 1; j <= price[i][0]; j++) scanf("%d", &price[i][j]);
    }

    memset(reachable, false, sizeof reachable);         // clear
everything
    for (i = 1; i <= price[0][0]; i++)       // initial values (base
cases)
      if (M - price[0][i] >= 0)      // to prevent array index out of
bound
        reachable[0][M - price[0][i]] = true;  // using first garment g =
0

    for (i = 1; i < C; i++)                    // for each remaining
garment
      for (j = 0; j < M; j++) if (reachable[i - 1][j]) // a reachable
state
        for (k = 1; k <= price[i][0]; k++) if (j - price[i][k] >= 0)
          reachable[i][j - price[i][k]] = true;   // also a reachable
state

    for (j = 0; j <= M && !reachable[C - 1][j]; j++); // the answer in
here

    if (j == M + 1) printf("no solution\n");          // last row has on
bit
    else            printf("%d\n", M - j);
} } // return 0;


/*
// same as above, but using space saving trick
#include <cstdio>
#include <cstring>
using namespace std;

int main() {
  int i, j, k, TC, M, C, cur;
  int price[25][25];
  bool reachable[2][210]; // reachable table[ONLY TWO ROWS][money (<=
200)]
  scanf("%d", &TC);
  while (TC--) {
    scanf("%d %d", &M, &C);
    for (i = 0; i < C; i++) {
      scanf("%d", &price[i][0]);
      for (j = 1; j <= price[i][0]; j++) scanf("%d", &price[i][j]);
    }

    memset(reachable, false, sizeof reachable);
    for (i = 1; i <= price[0][0]; i++)
      if (M - price[0][i] >= 0)
        reachable[0][M - price[0][i]] = true;

    cur = 1;                                      // we start with this
row
    for (i = 1; i < C; i++) {
      memset(reachable[cur], false, sizeof reachable[cur]);    // reset
row
      for (j = 0; j < M; j++) if (reachable[!cur][j])        // notice
!cur
        for (k = 1; k <= price[i][0]; k++) if (j - price[i][k] >= 0)
          reachable[cur][j - price[i][k]] = true;
      cur = !cur;                                      // flip the two
rows
    }

    for (j = 0; j <= M && !reachable[!cur][j]; j++);         // notice
!cur

    if (j == M + 1) printf("no solution\n");         // last row has on
bit
    else            printf("%d\n", M - j);
} } // return 0;

*/
```

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

int main() {
  int n = 9, A[] = { 4, -5, 4, -3, 4, 4, -4, 4, -5 };   // a sample array
A
  int running_sum = 0, ans = 0;
  for (int i = 0; i < n; i++)                                       //
O(n)
    if (running_sum + A[i] >= 0) {  // the overall running sum is still
+ve
      running_sum += A[i];
      ans = max(ans, running_sum);          // keep the largest RSQ
overall
    }
    else        // the overall running sum is -ve, we greedily restart
here
      running_sum = 0;      // because starting from 0 is better for
future
                           // iterations than starting from -ve running
sum
  printf("Max 1D Range Sum = %d\n", ans);                     // should be
9
} // return 0;
```

```
// Maximum Sum

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

int n, A[101][101], maxSubRect, subRect;

int main() {     // O(n^3) 1D DP + greedy (Kadane's) solution, 0.008 s in
UVa
  scanf("%d", &n);                    // the dimension of input square
matrix
  for (int i = 0; i < n; i++) for (int j = 0; j < n; j++) {
    scanf("%d", &A[i][j]);
    if (j > 0) A[i][j] += A[i][j - 1];      // only add columns of this
row i
  }

  maxSubRect = -127*100*100;    // the lowest possible value for this
problem
  for (int l = 0; l < n; l++) for (int r = l; r < n; r++) {
    subRect = 0;
    for (int row = 0; row < n; row++) {
      // Max 1D Range Sum on columns of this row i
      if (l > 0) subRect += A[row][r] - A[row][l - 1];
      else       subRect += A[row][r];

      // Kadane's algorithm on rows
      if (subRect < 0) subRect = 0;     // greedy, restart if running sum
< 0
      maxSubRect = max(maxSubRect, subRect);
  } }

  printf("%d\n", maxSubRect);
  return 0;
}



/*
#include <algorithm>
#include <cstdio>
using namespace std;

int n, A[101][101], maxSubRect, subRect;

int main() {                            // O(n^4) DP solution, ~0.076s in
UVa
  scanf("%d", &n);                    // the dimension of input square
matrix
  for (int i = 0; i < n; i++) for (int j = 0; j < n; j++) {
    scanf("%d", &A[i][j]);
    if (i > 0) A[i][j] += A[i - 1][j];           // if possible, add from
top
    if (j > 0) A[i][j] += A[i][j - 1];          // if possible, add from
left
    if (i > 0 && j > 0) A[i][j] -= A[i - 1][j - 1];     // avoid double
count
  }                                          // inclusion-exclusion
principle
```

```
  maxSubRect = -127*100*100;    // the lowest possible value for this
problem
  for (int i = 0; i < n; i++) for (int j = 0; j < n; j++) // start
coordinate
    for (int k = i; k < n; k++) for (int l = j; l < n; l++) {    // end
coord
      subRect = A[k][l];      // sum of all items from (0, 0) to (k, l):
O(1)
      if (i > 0) subRect -= A[i - 1][l];                              //
O(1)
      if (j > 0) subRect -= A[k][j - 1];                              //
O(1)
      if (i > 0 && j > 0) subRect += A[i - 1][j - 1];                 //
O(1)
      maxSubRect = max(maxSubRect, subRect); }          // the answer is
here

  printf("%d\n", maxSubRect);
  return 0;
}

*/



/*

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

int n, A[101][101], maxSubRect, subRect;

int main() {                // O(n^6) brute force solution, TLE (> 3s) in
UVa
  scanf("%d", &n);
  for (int i = 0; i < n; i++) for (int j = 0; j < n; j++)
    scanf("%d", &A[i][j]);

  maxSubRect = -127*100*100;    // the lowest possible value for this
problem
  for (int i = 0; i < n; i++) for (int j = 0; j < n; j++) // start
coordinate
    for (int k = i; k < n; k++) for (int l = j; l < n; l++) {    // end
coord
      subRect = 0;                           // sum items in this sub-
rectangle
      for (int a = i; a <= k; a++) for (int b = j; b <= l; b++)
        subRect += A[a][b];
      maxSubRect = max(maxSubRect, subRect); }          // the answer is
here

  printf("%d\n", maxSubRect);
  return 0;
}

*/
```

```
#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;
}
```

```
/* SuperSale */

// 0-1 Knapsack DP (Top-Down) - faster as not all states are visited

#include <algorithm>
#include <cstdio>
#include <cstring>
using namespace std;

#define MAX_N 1010
#define MAX_W 40

int i, T, G, ans, N, MW, V[MAX_N], W[MAX_N], memo[MAX_N][MAX_W];

int value(int id, int w) {
  if (id == N || w == 0) return 0;
  if (memo[id][w] != -1) return memo[id][w];
  if (W[id] > w)         return memo[id][w] = value(id + 1, w);
  return memo[id][w] = max(value(id + 1, w), V[id] + value(id + 1, w -
W[id]));
}

int main() {
  scanf("%d", &T);
  while (T--) {
    memset(memo, -1, sizeof memo);

    scanf("%d", &N);
    for (i = 0; i < N; i++)
      scanf("%d %d", &V[i], &W[i]);

    ans = 0;
    scanf("%d", &G);
    while (G--) {
      scanf("%d", &MW);
      ans += value(0, MW);
    }

    printf("%d\n", ans);
  }

  return 0;
}

/*

// 0-1 Knapsack DP (Bottom-Up)

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

#define MAX_N 1010
#define MAX_W 40

int i, w, T, N, G, MW, V[MAX_N], W[MAX_N], C[MAX_N][MAX_W], ans;

int main() {
  scanf("%d", &T);
  while (T--) {
    scanf("%d", &N);
    for (i = 1; i<= N; i++)
      scanf("%d %d", &V[i], &W[i]);

    ans = 0;
    scanf("%d", &G);
    while (G--) {
      scanf("%d", &MW);

      for (i = 0; i <= N;  i++) C[i][0] = 0;
      for (w = 0; w <= MW; w++) C[0][w] = 0;

      for (i = 1; i <= N; i++)
        for (w = 1; w <= MW; w++) {
          if (W[i] > w) C[i][w] = C[i - 1][w];
          else          C[i][w] = max(C[i - 1][w], V[i] + C[i - 1][w -
W[i]]);
        }

      ans += C[N][MW];
    }

    printf("%d\n", ans);
  }

  return 0;
}

*/
```

```
/* Coin Change */

// O(NV) DP solution

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

int N = 5, V, coinValue[5] = {1, 5, 10, 25, 50}, memo[6][7500];
// N and coinValue are fixed for this problem, max V is 7489

int ways(int type, int value) {
  if (value == 0)              return 1;
  if (value < 0 || type == N)  return 0;
  if (memo[type][value] != -1) return memo[type][value];
  return memo[type][value] = ways(type + 1, value) + ways(type, value -
coinValue[type]);
}

int main() {
  memset(memo, -1, sizeof memo); // we only need to initialize this once
  while (scanf("%d", &V) != EOF)
    printf("%d\n", ways(0, V));

  return 0;
}
```