Data Stucture Part 1

1.1 Objectives

To learn how to choose an appropriate data structure for a given problem.

1.2 Requirements

The tasks should be performed at the Lab (Makmal Pengajaran 1, Block B) machines in the faculty

Important:

Select **one leader** for each team of two students. Each team can use two computers but the computers must be side by side.

1.3 Tasks

Prior to the Lab

a. Start reading chapter 2 in the text book.

a) Type the algorithm collection codes below and submit the results in I-Folio.

```
#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");


  // 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;
}
```

b) Log in to the PC Square

Log in to the PC Square using the given username and passwords.

c) Problem Rice Sack (ACM/ICPC 2013)

Submit the solution for the following problem via PC Square

G	RICE SACK	
	Input	Standard Input
	Output	Standard Output
	Time Limit	1 second

Problem Description

Several sacks of rice need to be transported to five Orphanage Houses. The heaviest sack will go to Orphanage House Al-Ameen because it has the most number of orphanges. The lightest will be sent to Orphanage House Mutiara due to the small number of children staying there.
Given a row of rice sacks, decide which sack goes to Al-Ameen?

Input

The first line is an integer that represent the number of case. The following lines have 5 integers indicating the weights of 5 rice sacks, each separated by a blank. No sack will have a weight of more than 100 unit.

Output

For each test case, the output contains a line in the format Case #x: followed by a sequence of integers, where x is the case number (starting from 1) and an integer that indicates the weight of a rice sack that will go to Al-Ameen.

Sample Input Output

Sample Input	Sample Output
4 1 6 10 5 20 5 10 25 3 1 30 15 5 1 8 7 4 20 50 5	Case #1: 20 Case #2: 25 Case #3: 30 Case #4: 50

d) Based on the algorithm collection above, write a solution for UVAContest scoreboard https://uva.onlinejudge.org/external/102/10258.pdf. Submit the solution via PC Square.

UVa Online Judge

10258 Contest Scoreboard

Think the contest score boards are wrong? Here's your chance to come up with the right rankings.

Contestants are ranked first by the number of problems solved (the more the better), then by decreasing amounts of penalty time. If two or more contestants are tied in both problems solved and penalty time, they are displayed in order of increasing team numbers.

A problem is considered solved by a contestant if any of the submissions for that problem was judged correct. Penalty time is computed as the number of minutes it took for the first correct submission for a problem to be received plus 20 minutes for each incorrect submission received prior to the correct solution. Unsolved problems incur no time penalties.

Input

The input begins with a single positive integer on a line by itself indicating the number of the cases following, each of them as described below. This line is followed by a blank line, and there is also a blank line between two consecutive inputs.

Input consists of a snapshot of the judging queue, containing entries from some or all of contestants 1 through 100 solving problems 1 through 9. Each line of input will consist of three numbers and a letter in the format

contestant problem time L

where L can be 'C', 'I', 'R', 'U' or 'E'. These stand for Correct, Incorrect, clarification Request, Unjudged and Erroneous submission. The last three cases do not affect scoring.

Lines of input are in the order in which submissions were received.

Output

For each test case, the output must follow the description below. The outputs of two consecutive cases will be separated by a blank line.

Output will consist of a scoreboard sorted as previously described. Each line of output will contain a contestant number, the number of problems solved by the contestant and the time penalty accumulated by the contestant. Since not all of contestants 1-100 are actually participating, display only the contestants that have made a submission.

Sample Input

```
1

1 2 10 I
3 1 11 C
1 2 19 R
1 2 21 C
1 1 25 C
```

Sample Output

```
1 2 66
3 1 11
```

Evaluation

Items	
Submission	**5**
Achievement	**10**
Total	**15**