Lecture 4
Sorting

Euiseong Seo
(euiseong@skku.edu)

Sorting

- **One of the basic programming techniques**
- **Tons of existing approaches**
  - Time complexity
  - Space complexity
  - Other properties
- **Applicable to many problems**

Applications of Sorting

- Uniqueness Testing
- Deleting Duplications
- Median/Selection
- Frequency Counting
- Reconstructing the Original Order
- Set Intersection/Union
- Finding a Target Pair
- Efficient Searching

Basic Sorting Algorithms

- Insertion sort
- Merge sort
- Heap sort
- Quick sort
- Counting Sort
- Bucket Sort

Multicriteria Sorting

- **How can we break ties in sorting using multiple criteria?**
- **To use a complicated comparison function that combines all the keys to break ties**
- **To use repeated passes through a stable sorting function**
  - In reverse order of priorities of the keys

Sorting Library Functions in C

```
#include <stdlib.h>

void qsort(void *base, size_t nel, size_t width,
          int (*compare) (const void *, const void *));

int intcompare(int *i, int *j)
{
    if (*i > *j) return (1);
    if (*i < *j) return (-1);
    return (0);
}

qsort((char *) a, cnt, sizeof(int), intcompare);

bsearch(key, (char *) a, cnt, sizeof(int), intcompare);
```

Sorting and Searching in C++

```
void sort(RandomAccessIterator bg, RandomAccessIterator end)
void sort(RandomAccessIterator bg, RandomAccessIterator end,
          BinaryPredicate op)

void stable_sort(RandomAccessIterator bg, RandomAccessIterator end)
void stable_sort(RandomAccessIterator bg, RandomAccessIterator end,
          BinaryPredicate op)
```

Sorting and Searching in Java

```
static void sort(Object[] a)
static void sort(Object[] a, Comparator c)

binarySearch(Object[] a, Object key)
binarySearch(Object[] a, Object key, Comparator c)
```

Problem: Rating the Field

Pretty Polly has no shortage of gentlemen suitors who come a' courting. Indeed, her biggest problem is keeping track of who the best ones are. She is smart enough to realize that a program which ranks the men from most to least desirable would simplify her life. She is also persuasive enough to have talked you into writing the program.

Polly really likes to dance, and has determined the optimal partner height is 180 centimeters tall. Her first criteria is finding someone who is as close as possible to this height; whether they are a little taller or shorter doesn't matter. Among all candidates of the same height, she wants someone as close as possible to 75 kilograms without going over. If all equal-height candidates are over this limit, she will take the lightest of the bunch. If two or more people are identical by all these characteristics, sort them by last name, then by first name if it is necessary to break the tie.

Problem: Rating the Field

Polly is only interested in seeing the candidates ranked by name, so the input file:

```
George Bush          195     110
Harry Truman         180     75
Bill Clinton         180     75
John Kennedy         180     65
Ronald Reagan        165     110
Richard Nixon        170     70
Jimmy Carter         180     77
```

yields the following output:

```
Clinton, Bill
Truman, Harry
Kennedy, John
Carter, Jimmy
Nixon, Richard
Bush, George
Reagan, Ronald
```

Problem: Rating the Field

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

#define NAMELENGTH      30              /* maximum length of name */
#define NSUITORS        100             /* maximum number of suitors */

#define BESTHEIGHT      180             /* best height in centimeters */
#define BESTWEIGHT      75              /* best weight in kilograms */

typedef struct {
        char first[NAMELENGTH];         /* suitor's first name */
        char last[NAMELENGTH];          /* suitor's last name */
        int height;                     /* suitor's height */
        int weight;                     /* suitor's weight */
} suitor;

suitor suitors[NSUITORS];               /* database of suitors */
int nsuitors;                           /* number of suitors */
```

Problem: Rating the Field

```
read_suitors()
{
        char first[NAMELENGTH], last[NAMELENGTH];
        int height, weight;

        nsuitors = 0;

        while (scanf("%s %s %d %d\n",suitors[nsuitors].first,
               suitors[nsuitors].last, &height, &weight) != EOF) {
                suitors[nsuitors].height = abs(height - BESTHEIGHT);
                if (weight > BESTWEIGHT)
                        suitors[nsuitors].weight = weight - BESTWEIGHT;
                else
                        suitors[nsuitors].weight = - weight;

                nsuitors ++;
        }
}
```

Problem: Rating the Field

```
int suitor_compare(suitor *a, suitor *b)
{
        int result;                     /* result of comparison */

        if (a->height < b->height) return(-1);
        if (a->height > b->height) return(1);

        if (a->weight < b->weight) return(-1);
        if (a->weight > b->weight) return(1);

        if ((result=strcmp(a->last,b->last)) != 0) return result;

        return(strcmp(a->first,b->first));
}
```

Problem: Rating the Field

```
main()
{
        int i;                                  /* counter */
        int suitor_compare();

        read_suitors();

        qsort(suitors, nsuitors, sizeof(suitor), suitor_compare);

        for (i=0; i<nsuitors; i++)
                printf("%s, %s\n",suitors[i].last, suitors[i].first);
 }
```

Problem: Vito's Family

The famous gangster Vito Deadstone is moving to New York. He has a very big family there, all of them living on Lamafia Avenue. Since he will visit all his relatives very often, he wants to find a house close to them.

Indeed, Vito wants to minimize the total distance to all of his relatives and has blackmailed you to write a program that solves his problem.

Input

The input consists of several test cases. The first line contains the number of test cases.

For each test case you will be given the integer number of relatives r $(0 < r < 500)$ and the street numbers (also integers) $s_1, s_2, \ldots, s_i, \ldots, s_r$ where they live $(0 < s_i < 30,000)$. Note that several relatives might live at the same street number.

Output

For each test case, your program must write the minimal sum of distances from the optimal Vito's house to each one of his relatives. The distance between two street numbers s_i and s_j is $d_{ij} = |s_i - s_j|$.

Problem: Vito's Family

Sample Input

```
2
2 2 4
3 2 4 6
```

Sample Output

```
2
4
```

Problem: Bridge

A group of n people wish to cross a bridge at night. At most two people may cross at any time, and each group must have a flashlight. Only one flashlight is available among the n people, so some sort of shuttle arrangement must be arranged in order to return the flashlight so that more people may cross.

Each person has a different crossing speed; the speed of a group is determined by the speed of the slower member. Your job is to determine a strategy that gets all n people across the bridge in the minimum time.

Problem: Bridge

Input

The input begins with a single positive integer on a line by itself indicating the number of test cases, followed by a blank line. There is also a blank line between each two consecutive inputs.

The first line of each case contains n, followed by n lines giving the crossing times for each of the people. There are not more than 1,000 people and nobody takes more than 100 seconds to cross the bridge.

Output

For each test case, the first line of output must report the total number of seconds required for all n people to cross the bridge. Subsequent lines give a strategy for achieving this time. Each line contains either one or two integers, indicating which person or people form the next group to cross. Each person is indicated by the crossing time specified in the input. Although many people may have the same crossing time, this ambiguity is of no consequence.

Note that the crossings alternate directions, as it is necessary to return the flashlight so that more may cross. If more than one strategy yields the minimal time, any one will do.

The output of two consecutive cases must be separated by a blank line.

Problem: Bridge

Sample Input

```
1

4
1
2
5
10
```

Sample Output

```
17
1 2
1
5 10
2
1 2
```