2023 年度全国中学生程式设计竞赛 - 第五次工作坊

National Secondary School Programming Contest - Workshop 5 (Session1)

Speaker: Dr Kh'ng Xin Yi

Date: 6th Aug 2023

Problem Solving I

Session 1 (9:00AM – 10:30AM)

SAMPLE 1

Reordering Array

You are required to reorder the elements of an array with the following instructions:

(1) All the element(s) with the smallest value will be listed at the beginning of the array.

(2) All the element(s) with the largest value will be listed at the end of the array.

(3) The remaining elements will be listed in the relative order as they appear in the original array.

The example below shows the original array content and the reordered array content.

Original Array:

[6, 10, 8, 0, 3, 10, 0, 10, 1]

Reordered Array:

[0, 0, 6, 8, 3, 1, 10, 10, 10]

SAMPLE 1

Write a programme to

Input, in sequence

***X*,** the size of the array where $1 \leq X \leq 50$; and subsequently,

***X* integers** that represent the elements of the array.

Output, the order of the elements in the reordered array, following the 3 instructions given. Each two consecutive elements in the array will be separated by a space.

Input (输入)	Output (输出)
9 6 10 8 0 3 10 0 10 1	0 0 6 8 3 1 10 10 10
15 -7 9 -6 5 10 -7 0 7 12 6 8 1 -6 -7 -7	-7 -7 -7 -7 9 -6 5 10 0 7 6 8 1 -6 12
7 -2 6 6 -2 6 -2 6	-2 -2 -2 6 6 6 6
5 9 9 9 9 9	9 9 9 9 9
10 1 2 3 4 5 6 7 8 9 10	1 2 3 4 5 6 7 8 9 10

SOLUTION OVERVIEW

Not maximum value, push to front

max = 3

2 3 1 — **a[0] ≠ 3**
a[0] swaps to position 0

2 3 1 — **a[1] = 3**
a[1] remains

2 3 1 — **a[2] ≠ 3**
a[2] swaps to position 1

2 1 3

max = 3

1 3 3 2 — **a[0] ≠ 3**
a[0] swaps to position 0

1 3 3 2 — **a[1] = 3**
a[1] remains

1 3 3 2 — **a[2] = 3**
a[2] remains

1 3 3 2 — **a[3] ≠ 3**
a[3] swaps to position 1

1 2 3 3

SOLUTION OVERVIEW

Not maximum value, push to front

max = 3

2 3 1 **a[0] ≠ 3**
a[0] swaps to position 0

2 3 1 **a[1] = 3**
a[1] remains

2 3 1 **a[2] ≠ 3**
a[2] swaps to position 1

2 1 3

01 Find the maximum.

02 Traverse each value from the beginning.

03 If value = maximum, the current value remains.
If value ≠ maximum, swap it with the value at position where the next occurrence of the non-maximum value would be placed. Then, position + 1.

SOLUTION OVERVIEW

Not minimum value, push to back

min = 1

2 1 3 **a[2] ≠ 1**
a[2] swaps to position 2

2 1 3 **a[1] = 1**
a[1] remains

2 1 3 **a[0] ≠ 1**
a[0] swaps to position 1

1 2 3

04 Find the minimum.

05 Traverse each value from the end.

06 If value = minimum, the current value remains.
If value ≠ minimum, swap it with the value at position where the next occurrence of the non- minimum value would be placed.
Then, position - 1.

Suggested Solution-Sample 1

C++

```cpp
#include <iostream>
#include <algorithm>
using namespace std;

void swap(int& t1, int& t2) {
    int tmp = t1;
    t1 = t2;
    t2 = tmp;
}

int main() {
    int X, max, min, index, index2;

    cin >> X;
    int* arr = new int[X];
    for (int i = 0; i < X; i++)
        cin >> arr[i];

    max = *max_element(arr, arr + X);
    min = *min_element(arr, arr + X);
```

t1 = 1, t2 = 3
tmp = t1 = 1
t1 = t2 = 3
t2 = tmp = 1

01

Array Initialization
Get array size, X.
Create array with size X.
Input array value.

02

Maximum & Minimum
Get the maximum and minimum value.

Suggested Solution-Sample 1

C++

```
    index = 0;
    index2 = X - 1;

    for (int i = 0; i < X; i++) {
        if (arr[i] != max) {
            swap(arr[i], arr[index]);
            index++;
        }
    }

    for (int i = X - 1; i >= 0; i--) {
        if (arr[i] != min) {
            swap(arr[i], arr[index2]);
            index2--;
        }
    }

    for (int i = 0; i < X; i++)
        cout << arr[i] << " ";

    return 0;
}
```

03

Maximum Swap
Go through each element to see if it's maximum. If value ≠ maximum, swap it with the value at position where the next occurrence of the non-maximum value would be placed. Position + 1.

04

Minimum Swap
Go through each element to see if it's minimum. If value ≠ minimum, swap it with the value at position where the next occurrence of the non-minimum value would be placed. Position - 1.

05

Result Printing
Print reordered array.

SAMPLE 2

Sum of 24

Given an array of N integers, where $2 \leq N \leq 8$, it is known that there is one and only one combination of these integers that can be summed up to a total of 24.

For example, if the given array is [1,12,32,11], then the only combination will be [1,11,12].

Find such a combination from a given array and display the numbers in the combination in ascending order.

SAMPLE 2

Write a programme to

Input, in sequence

the first integer is *N*, indicating the number of integers in the array; and subsequently the *N* integers in the array.

Output, in sequence the combination of the integers from the given array that are summed up to 24. **Note that** you need to sort the output integers in ascending order.

Input (输入)	Output (输出)
6 14 6 12 7 8 6	6 6 12
8 6 11 14 8 12 5 14 15	5 8 11
8 1 10 7 15 7 15 15 3	7 7 10
8 15 13 6 4 8 13 2 8	2 6 8 8
8 15 10 8 13 13 9 13 10	9 15

SOLUTION OVERVIEW

Array = **[2 22 1]**

$2^3 = 8$ combinations

Combination of numbers	Sum
X X X	0
2 X X	2
X 22 X	22
X X 1	1
2 22 X	24
X 22 1	23
2 X 1	3
2 22 1	25

SOLUTION OVERVIEW

Array = **[2 22 1]**
2^3 = 8 combinations

Combination of numbers	Sum	Reversed binary representation
X X X	0	0 0 0
2 X X	2	1 0 0
X 22 X	22	0 1 0
X X 1	1	0 0 1
2 22 X	24	1 1 0
X 22 1	23	0 1 1
2 X 1	3	1 0 1
2 22 1	25	1 1 1

1. Efficient Element Representation: Binary digits indicate inclusion (1) or exclusion (0) of value.

2. Memory Efficiency: Storing all subsets directly consumes extensive memory, growing exponentially with input size. Binary approach uses fixed memory, much more efficient. {{2},{22},{2,22}}

3. Single Loop Computation: Binary approach computes subsets in a single loop, avoiding nesting or recursion for faster calculations.

SOLUTION OVERVIEW

Array = **[2 22 1]**

$2^3 = 8$ combinations

Combination of numbers	Sum	Reversed binary representation
X X X	0	0 0 0
2 X X	2	1 0 0
X 22 X	22	0 1 0
X X 1	1	0 0 1
2 22 X	24	1 1 0
X 22 1	23	0 1 1
2 X 1	3	1 0 1
2 22 1	25	1 1 1

01 Generate all possible combinations of input numbers using binary representation and reverse it.

02 Calculate sum of all possible combinations using binary representation of numbers.
'1' bits determine included numbers.

03 Check if any combination sums up to 24. Print its numbers in ascending order.

Suggested Solution-Sample 2

C++

```cpp
#include <iostream>
#include <bitset>
#include <vector>
#include <algorithm>
using namespace std;

int main(void) {
    int number_of_inputs, count = 0, total;

    cin >> number_of_inputs;
    int* data = new int[number_of_inputs];
    for (int i = 0; i < number_of_inputs; i++)
        cin >> data[i];

    for (int i = 0; i < pow(2, number_of_inputs); i++) {
        string binary = std::bitset<32>(i).to_string();
        reverse(binary.begin(), binary.end());
```

01

Array Initialization
Get array size
Create array
Input array value

02

Binary Representation
Generate all possible combinations of input numbers using binary representation and reverse it.

Suggested Solution-Sample 2

C++

```
        total = 0;
        vector<int> answer;
        for (int j = 0; j < number_of_inputs; j++) {
            if (binary[j] == '1') {
                total += data[j];
                answer.push_back(data[j]);
            }
        }

        if (total == 24) {
            sort(answer.begin(), answer.end());
            for (int k = 0; k < answer.size(); k++) {
                cout << answer[k] << endl;
            }
            count++;
        }
    }

    return 0;
}
```

03

Sum of Combinations
Calculate sum of all possible combinations using binary representation of numbers.
'1' bits determine included numbers.

04

Result Printing
Check if any combination sums up to 24.
Print its numbers in ascending order.

SAMPLE 3

Taking the Train

Dena takes the train to work every day from the Main Street Station to the City Center Station. There are two trains she can take, the red train or the blue train. Both trains start operating at 8 a.m. A red train arrives every X minutes, and a blue train arrives every Y minutes, where X and Y are both positive integers. Dena always arrives at the platform after Z minutes past 8 a.m. but before 9 a.m., and then she gets on the first train arriving at the station. However, if both trains arrive simultaneously, Dena gets on the train with relatively lower frequency. Write a programme to find out which train Dena is taking more often.

SAMPLE 3

Taking the Train

For example:

Assume that $X = 2$, $Y = 3$ and $Z = 6$.

If she arrives between 08:06 and 08:08, she takes the red train that arrives at 08:08.

If she arrives between 08:08 and 08:09, she takes the blue train that arrives at 08:09.

If she arrives between 08:09 and 08:10, she takes the red train that arrives at 08:10.

If she arrives between 08:10 and 08:12, she waits for both trains to arrive at 08:12 and takes the blue train because it is less frequent than the red train.

Since the arrival times of both trains are periodic with a period of 6 minutes, the rest of the calculation can be omitted in this example.

From the above, in conclusion, Dena is taking both trains equally often.

SAMPLE 3

Write a programme to

Input, in sequence

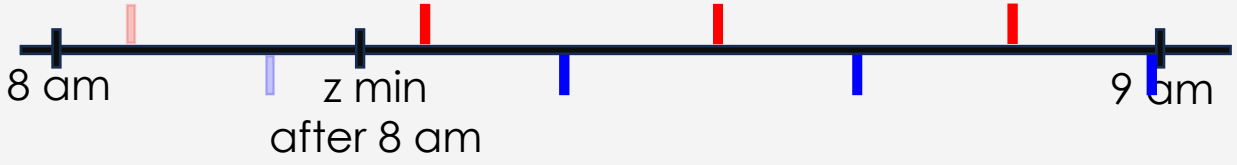
two positive integers, X, Y, where $X \neq Y$, $1 \leq X, Y \leq 100$, and one positive real number, Z, where $0 < Z < 60$.

Output, "Red" if Dena takes the red train more frequently. If she takes the blue train more frequently, then output "Blue". If she takes both trains equally often, then output "Equal".

Input (输入)	Output (输出)
2 3 6	Equal
33 38 36	Red
70 68 51	Blue
27 23 45	Red
58 59 57.5	Red

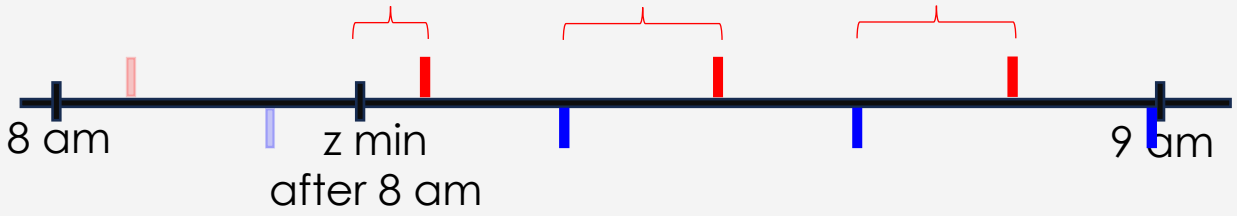
SOLUTION OVERVIEW

total duration of red train usage VS
total duration of blue train usage



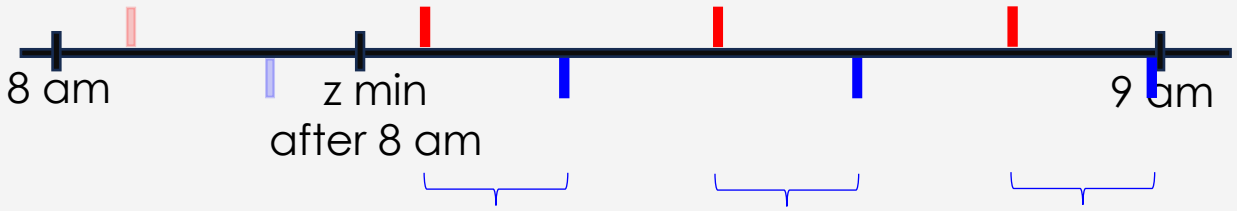
Scenario 1:
(red train arrives)

total red train usage += red train arrival time – last train arrival time



Scenario 2:
(blue train arrives)

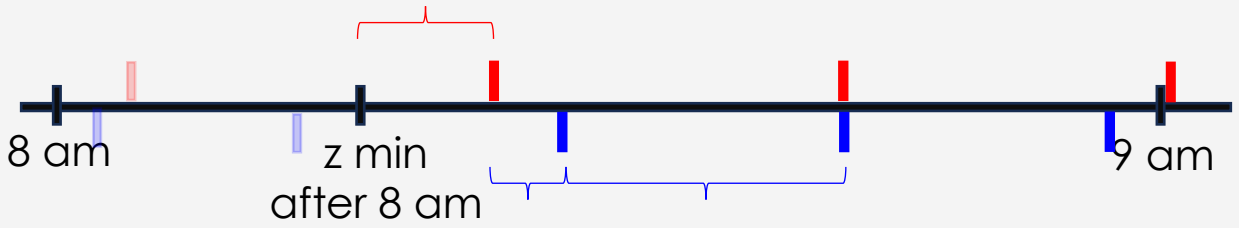
total blue train usage += blue train arrival time – last train arrival time



SOLUTION OVERVIEW

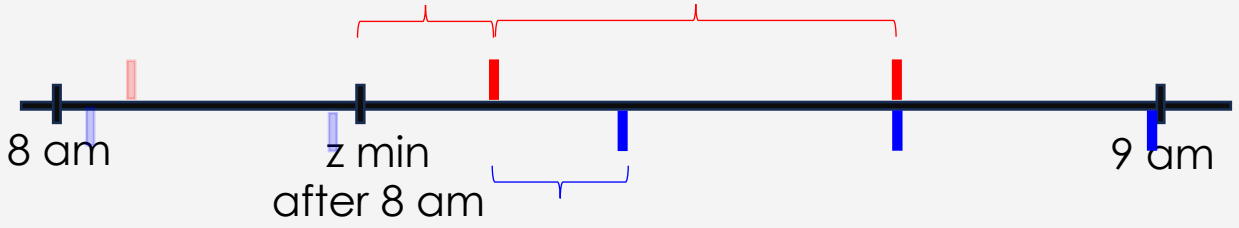
Scenario 3: (red and blue train arrive)

total red train usage ≠ total blue train usage



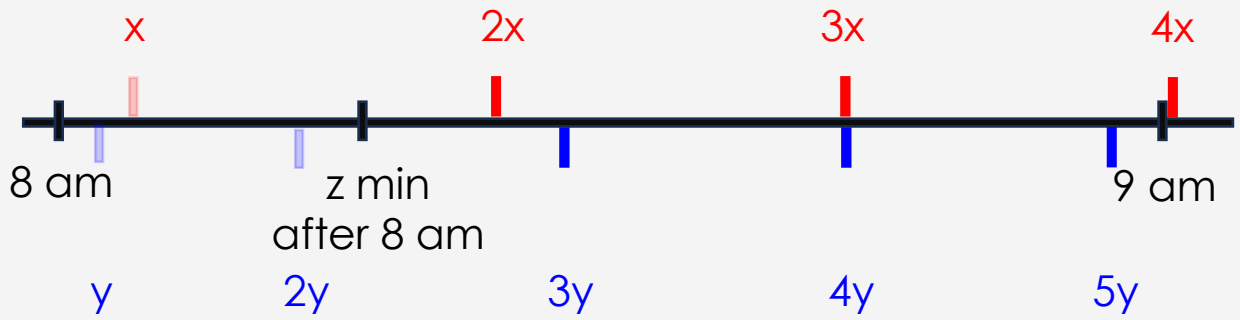
>> Choose the train with less frequency of use (e.g. blue)

total red train usage = total blue train usage



>> Choose the train with lower count of train services available (e.g. red)

SOLUTION OVERVIEW



01 Check the periodic time of only red train arriving at the station >> update total red train usage

02 Check the periodic time of only blue train arriving at the station >> update total blue train usage

03 Find LCM to determine the periodic time of both trains arriving at the station
>> choose train with lower usage frequency if total red train usage ≠ total blue train usage
>> choose train with fewer available services if total red train usage = total blue train usage)

Suggested Solution-Sample 3

C++

```cpp
#include <iostream>
using namespace std;

long int gcd(long int a, long int b){
    if (b == 0)
        return a;
    else
        return gcd(b, a % b);
}
int main(){
    long int x, y;
    double z, last, sum1 = 0, sum2 = 0;
    long int lcm, t = 0, threshold = 60, t1 = 1, t2 = 1;

    cin >> x >> y >> z;
    if (x > y)
        lcm = x * y / gcd(x, y);
    else
        lcm = x * y / gcd(y, x);
    last = z;
```

$$\operatorname{lcm}(a,b) = \frac{|a \cdot b|}{\gcd(a,b)}$$

01

Variable Initialization
Get the arrival time of red train (x), blue train (y) and Dena (z)

02

Lowest Common Multiple
Find the LCM of x and y to determine the periodic time of both trains arriving at the station.

Suggested Solution-Sample 3

C++

```
while (t < 60){
    t = min(t1 * x, t2 * y);
    if (t % x == 0) t1++;
    if (t % y == 0) t2++;
    if (t < z) continue;
    if (t % lcm == 0){
        if (t >= threshold) t = threshold;
        if (sum1 == sum2){
            if(t1 > t2)
                sum2 = sum2 + (t - last) ;
            else if (t1 < t2)
                sum1 = sum1 + (t - last) ;
            else if (t1 = t2){
                sum1 = sum1 + (t - last) / 2;
                sum2 = sum2 + (t - last) / 2;
            }
        }
        else if (sum1 > sum2)
            sum2 = sum2 + (t - last);
        else
            sum1 = sum1 + (t - last);
```

03

Arrival of Red & Blue Train

During periodic time, both red and blue trains arrive.

For train with lower frequency of use, update its frequency (train arrival time - passenger arrival time).

For train with same frequency of use, yet lower count of train services available, update its frequency.

For trains with same frequency of use, yet equal count of train services available, update both frequency (average).

Suggested Solution-Sample 3

C++

```
        else if (t % x == 0){
            if (t >= threshold) t = threshold;
            sum1 = sum1 + (t - last);
        }
        else if (t % y == 0){
            if (t >= threshold) t = threshold;
            sum2 = sum2 + (t - last);
        }
        last = t;
    }

    if (sum1 == sum2)
        cout << "Equal" << endl;
    else if (sum1 > sum2)
        cout << "Red" << endl;
    else
        cout << "Blue" << endl;
    return 0;
}
```

04

Arrival of Red / Blue Train

Before periodic time, either red or blue train arrives. For arriving train, update its frequency of use (train arrival time - passenger arrival time).

05

Result Printing

"Red" if Dena prefers the red train
"Blue" if she prefers the blue train
"Equal" if she takes both trains equally often.

Website:
http://nsspc2023.utar.edu.my/login

Student account:

1) username: demo01, password: 1234567890
2) username: demo02, password: 1234567890
3) username: Demo03, password: 1234567890
4) username: demo04, password: 1234567890

THANKS