The World's Richest

Name: 吴欣倍

Date: 2020-12-31

Chapter 1: Introduction

Problem Description

1. Input Specification

The First Line: Number of Richer(N <= 10^5) and Queries(K <= 1000)

Following N Lines: Each line contains specific information of richer, including Name(less than 9 chars) Age(0 < age <= 200) and Net-worth(-10^6 <= net worth <= 10^6)

Following K Lines: Each line contains a specific query, including max number of outputs, min-age and max-age

Note: max number of outputs defines the upper of outputs rather than the exact number of result.

2. Goal

Sort richer given according to their net-worth, age and name. **The priority of sorting key is net-worth(high->low), age(young->old), name(alphabetic).** Then output given number richest richer whose age is in range of min-age and max-age.

3. Output Specification

Each Query requires a case declaration and every line contains complete information of a satisfied richer.

4. Priority of Sorting Keys

1. Highest Priority: Non-increasing order of the net worths.
2. Second Priority: Non-decreasing order of the ages.
3. Third Priority: Non-decreasing alphabetical order of the names.

Note: It's assured that there's no two persons within same net-worth, age and name.

Problem Analysis

We have to sort richer by their net worth, age and name within certain priority and filtrate satisfied richer according to specific query. To make problem easier, we can divide the problem into several part.

1. **Part 1: Information Read in**

   Read in given information and queries by loops.

2. **Part 2: Richer Sorting**

   We have to sort richer by certain priorities. To save time and space, **Quick Sort** is preferred.

   *Note: It's an array sorting processor. And **time complexity** is also a vital point.*

3. **Part 3: Query Filtrate**

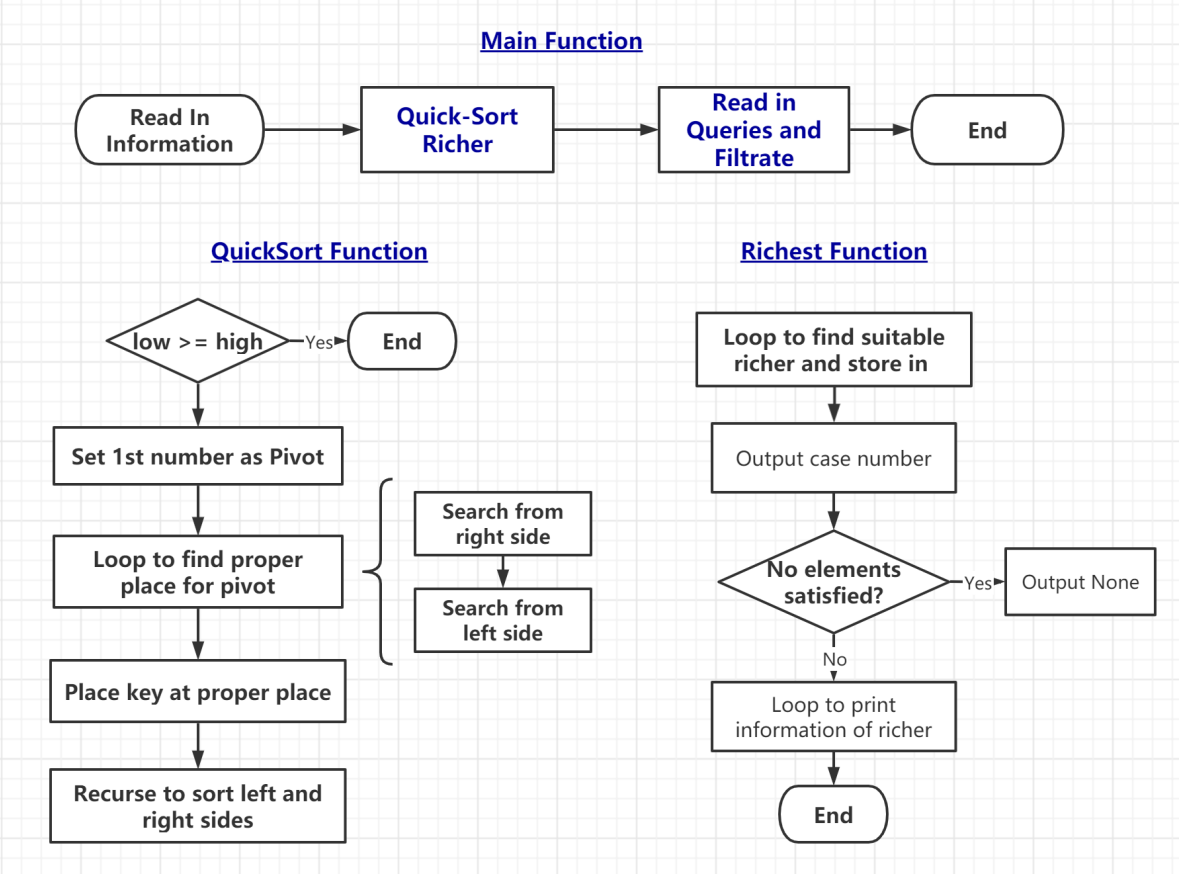
   Output given number of richer within satisfied age.

Chapter 2: Algorithm Specification

1. Main Algorithm: Quick-Sort

To save time and space, **Quick Sort algorithm** is selected. It's the most time-saving algorithm in practice.

2. Overall of Flow Chart



3. Pseudo-code for Core Part Function

1. **Data Structure**

```
#define maxn 100001 // max number of richer
#define maxm 101    // max number of richer required to output
#define maxc 9      // max chars of one's name
/*
 * struct to store information of richer
 * richers[maxn]: store all richer
 * tmpSort[maxm]: richer within suitable age
 */
struct Richer {
    char name[maxc];// used to store richer's name
    int  age;       // used to store richer's age
    int  netWorth;  // used to store richer's net worth
} richers[maxn], tmpSort[maxm];
```

2. **Pseudo-code for Quick-Sort**

```
Get low(i) edge and high(j) edge;
Detect if sort finished:
    if Yes, end sorting;
    if No, continue;
```

```
set fisrt number in array as pivot;
loop to find proper place for pivot:
    Search for the 1st number larger than key from right side;
    if i < j (sorting continue):
        set array[i] = this number;
    Search for the 1st number larger than key from left side;
    if i < j (sorting continue):
        set array[j] = this number
place key at proper place;
Recurse to sort left side;
Recurse to sort right side;
```

3. **Pseudo-code for Query-Filtrate**

```
loop to visit richers:
    if richer's age is in the range:
        push it in array;
        mark number of elements in the array;
Output result:
    print "Case #No.:\n";
    if no element in the array:
        print "None\n";
    else loop to output result:
        print information of richer line by line;
end;
```

Chapter 3: Testing Results

1. **Given Sample**

   sample

```
12 4    // N K
Zoe_Bill 35 2333    // information of richer
Bob_Volk 24 5888    // Name Age Net-worth
Anny_Cin 95 999999
Williams 30 -22
Cindy 76 76000
Alice 18 88888
Joe_Mike 32 3222
Michael 5 300000
Rosemary 40 5888
Dobby 24 5888
Billy 24 5888
Nobody 5 0
4 15 45 // output Number  min-age  max-age
4 30 35
4 5 95
1 45 50
```

   Expected Output

```
Case #1:
Alice 18 88888
```

```
Billy 24 5888
Bob_Volk 24 5888
Dobby 24 5888
Case #2:
Joe_Mike 32 3222
Zoe_Bill 35 2333
Williams 30 -22
Case #3:
Anny_Cin 95 999999
Michael 5 300000
Alice 18 88888
Cindy 76 76000
Case #4:
None
```

Actual Output

```
Case #1:
Alice 18 88888
Billy 24 5888
Bob_Volk 24 5888
Dobby 24 5888
Case #2:
Joe_Mike 32 3222
Zoe_Bill 35 2333
Williams 30 -22
Case #3:
Anny_Cin 95 999999
Michael 5 300000
Alice 18 88888
Cindy 76 76000
Case #4:
None
```

```
12 4
Zoe_Bill 35 2333
Bob_Volk 24 5888
Anny_Cin 95 999999
Williams 30 -22
Cindy 76 76000
Alice 18 88888
Joe_Mike 32 3222
Michael 5 300000
Rosemary 40 5888
Dobby 24 5888
Billy 24 5888
Nobody 5 0
4 15 45
Case #1:
Alice 18 88888
Billy 24 5888
Bob_Volk 24 5888
Dobby 24 5888
4 30 35
Case #2:
Joe_Mike 32 3222
Zoe_Bill 35 2333
Williams 30 -22
4 5 95
Case #3:
Anny_Cin 95 999999
Michael 5 300000
Alice 18 88888
Cindy 76 76000
1 45 50
Case #4:
None
--------------------------------
Process exited with return value 0
```

Result: Pass

2. **Min Input**

   Sample

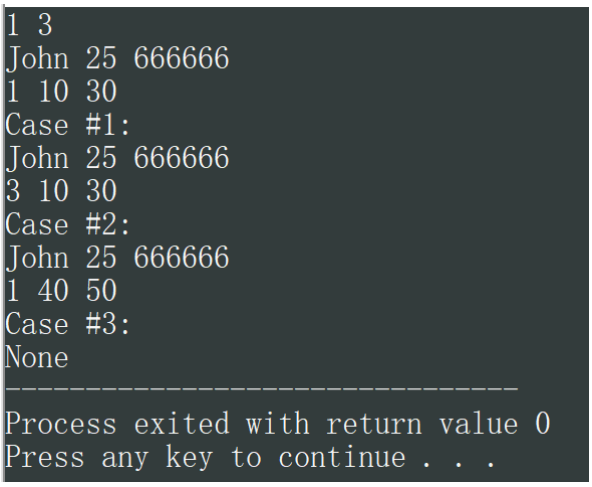
   ```
   1 3 // N K
   John 25 666666  // information of richer
   1 10 30 // output Number  min-age  max-age
   3 10 30
   1 40 50
   ```

   Expected Output

   ```
   Case #1:
   John 25 666666
   Case #2:
   John 25 666666
   Case #3:
   None
   ```

   Actual Result

```
Case #1:
John 25 666666
Case #2:
John 25 666666
Case #3:
None
```



Result: Pass

3. **Min Query**

   Sample

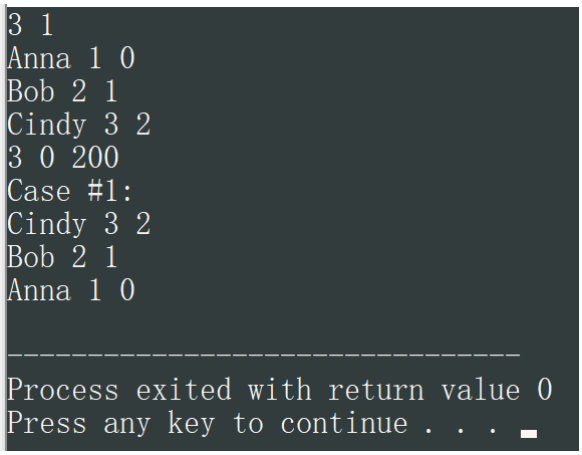
```
3 1 // N K
Anna 1 0    // information of richer
Bob 2 1     // Name Age Net-worth
Cindy 3 2   // output Number  min-age  max-age
3 0 200
```

   Expected Output

```
Case #1:
Cindy 3 2
Bob 2 1
Anna 1 0
```

   Actual Result

```
Case #1:
Cindy 3 2
Bob 2 1
Anna 1 0
```



Result: Pass

4. **Same Wealth(sort by age)**

   Sample

```
5 3 // N K
Anna 15 2000    // information of richer
Bob 28 2000     // Name Age Net-worth
Cindy 20 2000
David 10 2000
Elio 5 2000
5 5 28          // output Number  min-age  max-age
5 5 20
3 5 20
```

   Expected Output

```
Case #1:
Elio 5 2000
David 10 2000
Anna 15 2000
Cindy 20 2000
Bob 28 2000
Case #2:
Elio 5 2000
David 10 2000
Anna 15 2000
Cindy 20 2000
Case #3:
Elio 5 2000
David 10 2000
Anna 15 2000
```

   Actual Result

```
Case #1:
Elio 5 2000
David 10 2000
Anna 15 2000
Cindy 20 2000
Bob 28 2000
Case #2:
Elio 5 2000
```

```
David 10 2000
Anna 15 2000
Cindy 20 2000
Case #3:
Elio 5 2000
David 10 2000
Anna 15 2000
```

```
5 3
Anna 15 2000
Bob 28 2000
Cindy 20 2000
David 10 2000
Elio 5 2000
5 5 28
Case #1:
Elio 5 2000
David 10 2000
Anna 15 2000
Cindy 20 2000
Bob 28 2000
5 5 20
Case #2:
Elio 5 2000
David 10 2000
Anna 15 2000
Cindy 20 2000
3 5 20
Case #3:
Elio 5 2000
David 10 2000
Anna 15 2000

--------------------------------
Process exited with return value 0
Press any key to continue . . .
```

Result: Pass

5. **Same Wealth and Age(sort by name)**

   Sample

```
5 1
Alice 20 2000
Bob 20 2000
Cindy 20 2000
Candy 20 2000
Carol 20 2000
5 20 20
```

   Expected Output

```
Case #1:
Alice 20 2000
Bob 20 2000
Candy 20 2000
Carol 20 2000
Cindy 20 2000
```

Actual Result

```
Case #1:
Alice 20 2000
Bob 20 2000
Candy 20 2000
Carol 20 2000
Cindy 20 2000
```

```
5 1
Alice 20 2000
Bob 20 2000
Cindy 20 2000
Candy 20 2000
Carol 20 2000
5 20 20
Case #1:
Alice 20 2000
Bob 20 2000
Candy 20 2000
Carol 20 2000
Cindy 20 2000

--------------------------------
Process exited with return value 0
Press any key to continue . . .
```

Result: Pass

6. **Negative Net-worth**

Sample

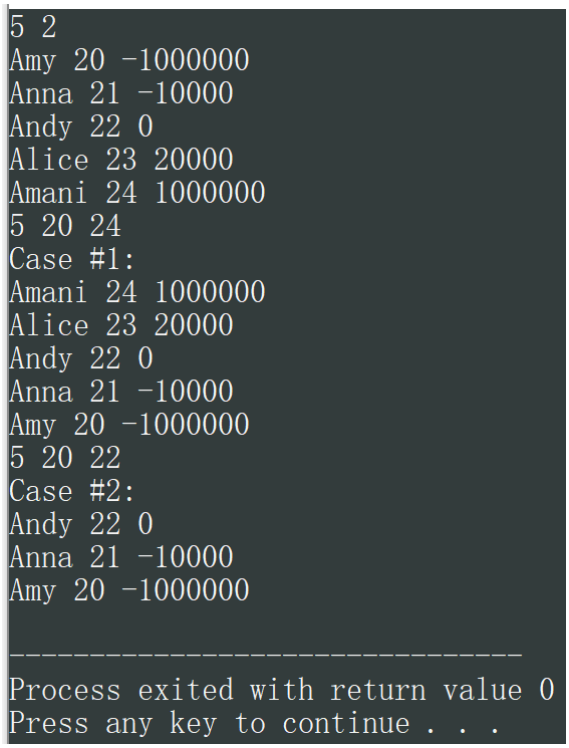
```
5 2
Amy 20 -1000000
Anna 21 -10000
Andy 22 0
Alice 23 20000
Amani 24 1000000
5 20 24
5 20 22
```

Expected Output

```
Case #1:
Amani 24 1000000
Alice 23 20000
Andy 22 0
Anna 21 -10000
Amy 20 -1000000
Case #2:
Andy 22 0
Anna 21 -10000
Amy 20 -1000000
```

Actual Result

```
Case #1:
Amani 24 1000000
Alice 23 20000
Andy 22 0
Anna 21 -10000
Amy 20 -1000000
Case #2:
Andy 22 0
Anna 21 -10000
Amy 20 -1000000
```



Result: Pass

7. **Big Data**

   Big-data Sample can be tested on PAT (Advanced Level) Practice 1055.

Final Result

The program works successfully for all testing samples.

*Note: All testing cases except big-data case are provided in file "testing cases.txt”

Chapter 4: Analysis and Comments

Time Complexity

Firstly, assume the number of richer as **N** , the number of queries is **K**, the number of outputs in specif query is **n**.

Secondly, compute each part’s time complexity.

1. Main Function

   It takes one-layer loop to read in information of all richer and the other one-layer loop to read in queries. Assume quick-sort function takes Tqs(N) and Filtrating Richest takes Tr(N, n);

T(N, K) = O(N) + O(K*Tr(N, n)) + Tqs(N);

2. Quick Sort

   Though the Compare function is a little complex, it does not enlarge the time complexity of Quick Sort. Time complexity for normal quick sort algorithm is O(nlogn).

   Tqs(N) = O(NlogN);

3. Filtrate Richest

   It takes a one-layer loop to filtrate richer within satisfied age and a one-layer loop to output information if need.

   Tr(N, n) = O(N) + O(n) = O(N)

Thirdly, Compute the total time complexity.

T(N, K) = O(N) + Tqs(N) + O(K*Tr(N, n)) = O(N) + O(NlogN) + O(K *O(N)) = max{O(NlogN), O(K *N)}

Result: T(N, K) = max{O(NlogN), O(K *N)};

Space Complexity

Firstly, assume the number of richer as **N** , the number of queries is **K**, the number of outputs in specif query is **n**.

Secondly, compute each part's space complexity.

1. Main Data Structure

```
#define maxn 100001
#define maxm 101
#define maxc 9
struct Richer {
    char name[maxc];
    int  age;
    int  netWorth;
}richers[maxn], tmpSort[maxm];
```

   Thus, Sd(N) = O((maxn + maxm)*(O(Richer)));

2. Quick Sort

   Quick Sort Algorithm takes O(NlogN) space complexity.

   Sqs(N) = O(NlogN);

Thus, S(N, maxm, maxn) = Sd(N) + Sqs(N) = O((maxn + maxm)*(O(Richer))) + O(NlogN) = O(NlogN)

Result: S(N) = O(NlogN)

Appendix: Source Code in C

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define maxn 100001
#define maxm 101
#define maxc 9
```

```c
struct Richer {
    char name[maxc];
    int  age;
    int  netWorth;
}richers[maxn], tmpSort[maxm];
int N, K;

void Richest( int num, int ageMin, int ageMax, int caseNum );
int  Compare( struct Richer richer, int i );
void QuickSort( struct Richer* array, int low, int high );


/*
 * Part 1: Main Function
 * Read in Input -> Sort -> Answer Queries
 */
int main() {
    // Read in N, K
    int i;
    scanf("%d %d", &N, &K );
    // Read in Information of Richers
    for ( i = 0; i < N; i++ )
        scanf("%s %d %d", richers[i].name, &richers[i].age,
&richers[i].netWorth);

    // Sort Richers
    QuickSort( richers, 0, N-1 );

    // Read in queries and Output
    for ( i = 0; i < K; i++ ) {
        // Read in specific Query
        int num, ageMin, ageMax;
        scanf("%d %d %d", &num, &ageMin, &ageMax );
        // Filtrate Richest
        Richest( num, ageMin, ageMax, i+1 );
    }

    return 0;
}


/*
 * Part 2: Quick Sort
 * Sort richer by Quick Sort Algorithm
 */
void QuickSort( struct Richer* array, int low, int high ) {
    // Detect if sort finished
    if ( low >= high )
        return ;
    int i = low, j = high;
    struct Richer key = array[low]; // set as pivot
    while ( i < j ) {
        // search for the 1st number larger than key from right side
        while ( i < j && (Compare(key, j)))
            j--;
        if ( i < j )
            array[i++] = array[j];
        // search for the 1st number larger than key from left side
```

```
        while ( i < j && !Compare(key, i) )
            i++;
        if ( i < j )
            array[j--] = array[i];
    }
    array[i] = key; // place key at proper place
    QuickSort(array, low, i-1); // Sort numbers by pivot's left side
    QuickSort(array, i+1, high);// Sort numbers by pivot's right side
}

int Compare( struct Richer richer, int i ) {
    // 1st, compare by net worth(wealthier)
    if ( richer.netWorth != richers[i].netWorth )
        return richer.netWorth > richers[i].netWorth;
    // 2nd, compare by age(younger)
    if ( richer.age != richers[i].age )
        return richer.age < richers[i].age;
    // 3rd, compare by name(alphabetic)
    if ( strcmp( richers[i].name, richer.name) > 0 )
        return 1;
    return 0;
}


/*
 * Part 3: Filtrate Richest
 * Select richer within suitable age -> Output
 */
void Richest( int num, int ageMin, int ageMax, int caseNum ) {
    int i, j = 0;
    // visit richers and select suitable ones
    for ( i = 0; i < N && j < num; i++ )
        if ( richers[i].age >= ageMin && richers[i].age <= ageMax ) {
            tmpSort[j] = richers[i];
            j++;
        }
    // Output
    printf("Case #%d:\n", caseNum);
    // if no one in the range of age
    if ( !j ) {
        printf("None");
        return ;
    }
    // else, print information line by line
    for ( i = 0; i < num && i < j; i++ ) {
        printf("%s %d %d\n", tmpSort[i].name, tmpSort[i].age,
tmpSort[i].netWorth);
    }

    return ;
}
```

Declaration

I hereby declare that all the work done in this project titled "The World's Richest" is of my independent effort.