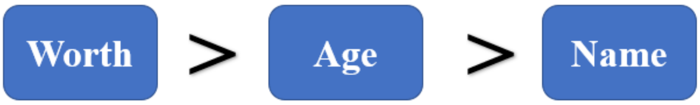
The World's Richest

Author’s Name：Zhang Youchao

Date: 2020-12-30

Chapter 1: Introduction

Analysis: This is a sorting problem. The key to the problem is how to sort according to conditions while ensuring that the complexity of sorting is low enough.



At the same time, this is a sort based on comparison, so a comparison function needs to be implemented to highlight the priority of different attributes.

In order to avoid directly sorting large structures, my idea is to create an array of storage addresses and sort the array.

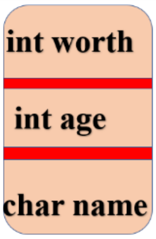
In order to achieve the fastest sorting speed, I plan to use a quick sort algorithm.

The basic idea of quick sort is: divide the data to be sorted into two independent parts through a sort, all the data in one part is smaller than all the data in the other part, and then perform the two parts of data separately according to this method For quick sorting, the entire sorting process can be performed recursively so that the entire data becomes an ordered sequence.

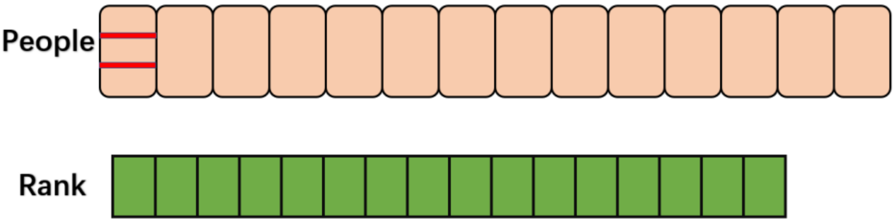
Chapter 2: Algorithm Specification

1.The data structure

```
/*the struct to store a person*/
struct person
{
    int age;
    name Name;
    int worth;
};
```



```
struct person Person[Maxnum]; /*to store people*/
rank Rank[Maxnum];   //to make a array to store the address of people
```



In order to avoid directly sorting large structures, my idea is to create an array of storage addresses and sort the array.

2. Pseudo code of quick sort algorithm.

Algorithm1: Quicksort

Input : array,left,right

Output: None

```
function Quicksort (array, left, right)
quicksort (array){
  if (array.length > 1){
    choose a pivot;
    while (there are items left in array){
      if (item < pivot)
        put item into subarray1;
      else
        put item into subarray2;
    }
    quicksort(subarray1);
    quicksort(subarray2);
  }
}
```

3.A comparison function needs to be implemented to highlight the priority of different attributes.

```
/*compare by worth -> age->Name*/
int cmp(rank a, rank b)
{
    if (a->worth != b->worth)
        return a->worth > b->worth;/*bigger net worth*/
    else if (a->age != b->age)
        return a->age < b->age;    /*small age is bigger*/
    return strcmp(a->Name, b->Name) < 0;/*small name is bigger*/
}
```

4.The main loop

```
main:
    Read in data
    Sort by corresponding priority
    Traverse the sorted array
    Output in order
```

Chapter 3: Testing Results

Result table:

Test case	a brief description of the purpose	Expected result	Actual behavior	Current status
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 4 30 35 4 5 95 1 45 50	To test whether the program can detect the input given by the project	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	Expected result	Pass
1 1 Zoe_Bill 35 2333 4 15 45	To test whether the program can pass simple input	Case #:1 Zoe_Bill 35 2333	Expected result	Pass
12 1 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 12 0 100	To test whether the program can sort all right	Case #:1 Anny_Cin 95 999999 Michael 5 300000 Alice 18 88888 Cindy 76 76000 Billy 24 5888 Bob_Volk 24 5888 Dobby 24 5888 Rosemary 40 5888 Joe_Mike 32 3222 Zoe_Bill 35 2333 Nobody 5 0 Williams 30 -22	Expected result	Pass

36 1 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 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 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 36 0 100	To test whether the program can sort all right	Case #:1 Anny_Cin 95 999999 Anny_Cin 95 999999 Anny_Cin 95 999999 Michael 5 300000 Michael 5 300000 Michael 5 300000 Alice 18 88888 Alice 18 88888 Alice 18 88888 Cindy 76 76000 Cindy 76 76000 Cindy 76 76000 Billy 24 5888 Billy 24 5888 Billy 24 5888 Bob_Volk 24 5888 Bob_Volk 24 5888 Bob_Volk 24 5888 Dobby 24 5888 Dobby 24 5888 Dobby 24 5888 Rosemary 40 5888 Rosemary 40 5888 Rosemary 40 5888 Joe_Mike 32 3222 Joe_Mike 32 3222 Joe_Mike 32 3222 Zoe_Bill 35 2333 Zoe_Bill 35 2333 Zoe_Bill 35 2333 Nobody 5 0 Nobody 5 0 Nobody 5 0 Williams 30 -22 Williams 30 -22 Williams 30 -22	Expected result	Pass

12 1 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 12 100 200	To test whether the program can detect wrong range	Case #:1 None	Expected result	Pass

Chapter 4: Analysis and Comments

Analysis of the time and space complexities of the algorithms. Comments on further possible improvements.

Quick sort-time and space complexity:

Quicksort divides the array to be sorted into two parts each time. In an ideal situation, dividing the array to be sorted into two parts of equal length each time requires log(n) divisions.

In the worst case, that is, when the array is already ordered or roughly ordered, each division can only reduce one element, and quick sort will unfortunately degenerate into bubble sort, so the lower bound of the time complexity of quick sort is O(nlogn), the worst case is O(n^2). In practical applications, the average time complexity of quicksort is O(nlogn).

Quick sort only needs a constant level of space during the operation of the sequence. The space complexity is S(1).

But you need to pay attention to the space required to spend at least log(n) at most n on the recursive stack.

Further possible improvements

The realization of quick sort needs to consume the space of the recursive stack, and in most cases, the recursive solution is completed by using the system recursive stack. When the number of elements is large, frequent access to the system stack will affect the efficiency of sorting.

A common method is to set a threshold. In each recursive solution, if the total number of elements is less than this threshold, the quick sort is abandoned and a simple sorting process is called to complete the sorting of the subsequence. This method reduces the frequent access to the recursive stack of the system and saves time consumption.

Appendix: Source Code (in C)

At least 30% of the lines must be commented. Otherwise th e code will NOT be evaluated.

```
#include <stdio.h>
#include <string.h>
#define Maxnum 1000000

typedef char name[9];
typedef struct person *rank;
struct person /*the struct to store a person*/
{
    int age;
    name Name;
    int worth;
};
rank Rank[Maxnum];            //to make a array to store the address of people
struct person Person[Maxnum]; /*to store people*/

void Swap(rank *a, rank *b)/*Swap*/
{
    rank temp = *a;
    *a = *b;
    *b = temp;
}
int cmp(rank a, rank b)/*compare by worth -> age->Name*/
{
    if (a->worth != b->worth)
        return a->worth > b->worth;/*bigger net worth*/
    else if (a->age != b->age)
        return a->age < b->age;    /*small age is bigger*/
    return strcmp(a->Name, b->Name) < 0;/*small name is bigger*/
}

/*Quick sort*/
void Q_sort(rank A[], int left, int right)
{
    int i = left;
    int j = right - 1;
    int pivot = right; /* select pivot */
    if (left < right)  /*if left >= right , then  end */
    {
        for(;;)
        {
            while (i <= right - 1 && !cmp(A[i], A[pivot]))
                i++; /* scan from left */
            while (j >= left && cmp(A[j], A[pivot]))
                j--; /* scan from right */
            if (i < j)
                Swap(&A[i], &A[j]); /* adjust partition */
            else break;
        }
        Swap(&A[i], &A[right]);  /* restore pivot */
        Q_sort(A, left, i - 1);  /* recursively sort left part */
        Q_sort(A, i + 1, right); /* recursively sort right part */
    }
}
int main()
{
    int numPerson = 0; //number of people
    int numCase = 0;   //number of cases
    scanf("%d %d", &numPerson, &numCase);
    for (int i = 0; i < numPerson; ++i) //to read data & combine rank to person
    {
        scanf("%s %d %d", Person[i].Name, &Person[i].age, &Person[i].worth);
        Rank[i] = Person + i; //store the address
    }
    Q_sort(Rank, 0, numPerson - 1); //sort the Rank by net worth
    /*cases*/
    for (int i = 1; i <= numCase; ++i)
    {
        int num, lower, upper, cnt = 0;
        scanf("%d %d %d", &num, &lower, &upper);
        printf("Case #:%d\n", i);
        for (int k = numPerson - 1; k >= 0 && cnt < num; --
k) /*from biggest to smallest*/
        {
            if (Rank[k]->age >= lower && Rank[k]->age <= upper)
            {
                ++cnt;
                printf("%s %d %d\n", Rank[k]->Name, Rank[k]->age, Rank[k]->worth);
            }
        }
        if (!cnt) printf("None\n");
    }
    system("pause");
    return 0;
}
```

Declaration

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