VG101 Lab Worksheet

Lab 9

Instructor: Dr. Yifei ZHU
TA: Hangrui CAO
TA: Haoxuan SHAN
TA: Qinhang WU
TA: Muchen XU

Code Quality Requirement

- Ensure proper indentation
- Make naming meaningful
- Include necessary comments
- Split the code over functions
- Test the code as much as you can

Basic Syntax Exercise

File IO and Program Argument

Write a program to read from the input file which contains many lines of texts, and then output all the lines in the reversed order (but keep the line contents unchanged) to the output file.

Your program should receive two strings in the program argument as the input and output filename.

You should run your program like `./xxx in.txt out.txt` or `xxx.exe in.txt out.txt`.

The sample input and answer are listed below.

in.txt:

```
>_<
qwq
0v0
```

out.txt:

```
0v0
qwq
>_<
```

Sort (JOJ available)

You need to have command of at least one sorting method.

Selection Sort

O(n^2)

In i-th round, find the i-th smallest number. Loop through the element from i+1 to end, swap a_i and a_j if $a_i > a_j$.

```
for( int i = 1 ; i <= n ; ++ i )
    for( int j = i + 1 ; j <= n ; ++ j )
        if( a[i] > a[j] )
            swap( &a[i] , &a[j] );
```

Insertion Sort

O(n^2)

Maintain an ordered array. Each time add one number into the set. Find the proper position and insert.

```
int input[SIZE];
int output[SIZE],cur_length = 0;
for( int i = 1 ; i <= n ; ++ i ) {
    int j = 1;
    for( ; j < i ; ++ j )
        if( input[i] < output[j] )
            break;
    for( int k = i-1 ; k >= j ; -- k )
        output[k+1] = output[k];
    output[j] = input[i];
}
```

Bubble Sort (suggested)

O(n^2)

In i^{th} round, find the $(n-i)^{th}$ smallest number. Loop through the element from 1 to $(n-i)$, compare a_i and a_{i+1} and swap if necessary.

```
for( int i = 1 ; i <= n ; ++ i )
    for( int j = 1 ; j <= n-i ; ++ j )
        if( a[i] > a[i+1] )
            swap( &a[i] , &a[+1] ); // define it yourself
```

Quick Sort

O($n \log n$)

The main idea for this sort algorithm is dividing an interval into two parts where any element in the left part is smaller or equal than any element in the right part. Each round, we set a judge value and finish this division. Then we recursively check the left part and the right part. Continuously do this procedure, we will sort the sequence.

Try to understand the program below through reading the comments and simulate the procedure on a piece of paper.

```
void quickSort(int left, int right, int a[]) // sort the interval from left to right
{
    if(left >= right) // there's only one element in the interval, no need to sort.
        return;

    int i, j, mid;
    i = left, j = right;
    mid = a[left];  // set left element as a judge (can choose other positions)

    // Expect Result: Find out the number i that all elements
    // on the leftside of i are smaller/equal than mid while all elements
    // on the rightside of i are larger/equal than mid.

    while (i < j) // Condition 1: this loop end when i == j
    {
        while (a[j] >= mid && i < j) // Condition 2:
            j--;       // from rightside find a number smaller than mid
        while (a[i] <= mid && i < j)  // Condition 3:
            i++;       // from leftside find a number larger than mid
        swap( &a[i] , &a[j] ); // define it by yourself
    }

    // Put mid on the i position and finished our expect result.
    // Note that, carefully examine Condition 2 and 3 as well as their order,
    // you will find when i == j, a[i] must smaller than mid. Hence, it can
    // be put into the leftside.
    a[left] = a[i];
    a[i] = mid;


    quickSort(left, i - 1, a); // recursively sort the leftside
    quickSort(i + 1, right, a); // recursively sort the rightside
}
```

Indeed, you can use the function `void qsort( void *ptr, size_t count, size_t size, int (*comp)(const void *, const void *) );` in C library `stdlib.h` (which will make your life easier). Refer to https://en.cppreference.com/w/c/algorithm/qsortz for detailed explanation.

Note: for your quickSort, the expect time complexity is $O(n\log(n))$, however, it can reach $O(n^2)$ in the worst case.

Merge Sort

O($nlog(n)$)

As we have mentioned this method both in lab and lecture, we hope to introduce a solution given on Merge Sort to the Reverse Pair question. Consider a pair in a sequence, if the larger one has the position before the smaller one, this pair is called a reverse pair.

An application of merge sort is to count the number of reverse pairs in a sequence. Then, to change the merge sort into the solution to this question, you need to change the merge part. For examples, I'm currently merging i^{th} number in left and j^{th} number in right, if $a_i < b_j$, we add a_i to the output sequence, and a_i will donate j pairs in this merge process. The rest procedure will

be the same as the merge sort.

```
3 5 6 1 2 7
has (3,1) (3,2) (5,1) (5,2) (6,1) (6,2) 6 in total.
```

Stability / Instability

Consider you have n students with n scores and numbered the id as 1~n. The score may be the same. You hope that when the score is the same, the smaller id will in a front position. To realize this, you need to pay attention to what the swap condition should be, e.g. `<` or `<=`. Note that Selection Sort, Quick Sort is not stable. The bubble sort, Merge Sort is stable.

Algorithmic Exercise

Fast Power Algorithm (JOJ available)

Description

The *Fast Power Algorithm* is an efficient algorithm to calculate $a^b \mod p$ (" $\mod p$" means `% p`, the **modulo arithmetic** in C/C++) within $O(\log b)$ time complexity.

The main idea is to consider the relationship between $a^{\lfloor \frac{b}{2} \rfloor} \mod p$ and $a^b \mod p$, discuss the situation when b is odd or even. And then solve the problem recursively.

Start with the code:

```
typedef long long LL;
LL quickpow(LL a,LL b,LL p)
{
    //TODO: solve the problem recursively
    //      by identifying the boundary stage
    //      and the recusion stage.
}
```

Hint: In mathematics it holds that

```
(a+b)%p==(a%p+b%p)%p;
(a*b)%p==((a%p)*(b%p))%p
```

Requirement

Given a, b, p, find $a^b \mod p$

Input Format

Three integers a, b, and p.

Output Format

A string in the format `a^b mod p=s`, where $s = a^b \mod p$.

Sample #1

input:

```
2 10 9
```

output:

```
2^10 mod 9=7
```

Specification

$0 \leq b, p < 2^{31}$, and $1 \leq p < 2^{31}$

Eratosthenes Sieve

Description

Given a sequence of integers within a certain range, you need to find out all the prime numbers without changing their order. Recall that a prime number is a natural number greater than 1 divisible only by itself and 1.

Requirement

To check if an integer is a prime, intuitively we can achieve that by dividing integers smaller than itself. However, there is a more efficient algorithm called **Eratosthenes Sieve**, and you are required to implement it for this exercise. The algorithm works as follows:

1. For all integers within given range, mark them as prime
2. For 0 and 1, mark them as not prime
3. Find the next smallest integer `i` that is marked as prime
4. For all integers that can be divided by `i`, mark them as not prime
5. Repeat Steps 3-4 until no more `i` can be found in Step 3

After running the above algorithm, you have generated a lookup table. To check if a given integer is a prime, simply check that lookup table and see if it is marked as prime or not.

Input Format

The first line: an integer n `(1 <= n <= 20)` that stands for the length of the sequence.

The second line: n integers within range `[0,1000000]`.

Output Format

All the primes in original order.

Sample #1

input:

```
6
2 4 5 233 2333 23333 6662333
```

output:

```
2 5 233 2333 23333 6662333
```

Takeaway

In this exercise, we use more memory to store the look-up table, but thanks to that we can solve the problem in less time. Time-space Tradeoff is very common when you try to solve problems and hope this exercise gives you a basic sense.

> Reference: VE280-SU2020-Lab2, created by Yiqing Fan