

# Algorithm Design and Analysis - Assignment 3

2020K8009907032

唐嘉良

2022 年 11 月 20 日

## 目录

<b>1</b>	<b>Problem One</b>	<b>2</b>
1.1	Algorithm Description . . . . .	2
1.2	the correctness of the algorithm . . . . .	4
<b>2</b>	<b>Problem Two</b>	<b>5</b>
2.1	Algorithm Description . . . . .	5
2.2	the correctness of the algorithm . . . . .	6
<b>3</b>	<b>Problem Three</b>	<b>7</b>
3.1	Algorithm Description . . . . .	7
3.2	the correctness of the algorithm . . . . .	8
<b>4</b>	<b>Problem Four</b>	<b>9</b>
4.1	Algorithm Description . . . . .	9
4.2	the correctness of the algorithm . . . . .	10

# 1 Problem One

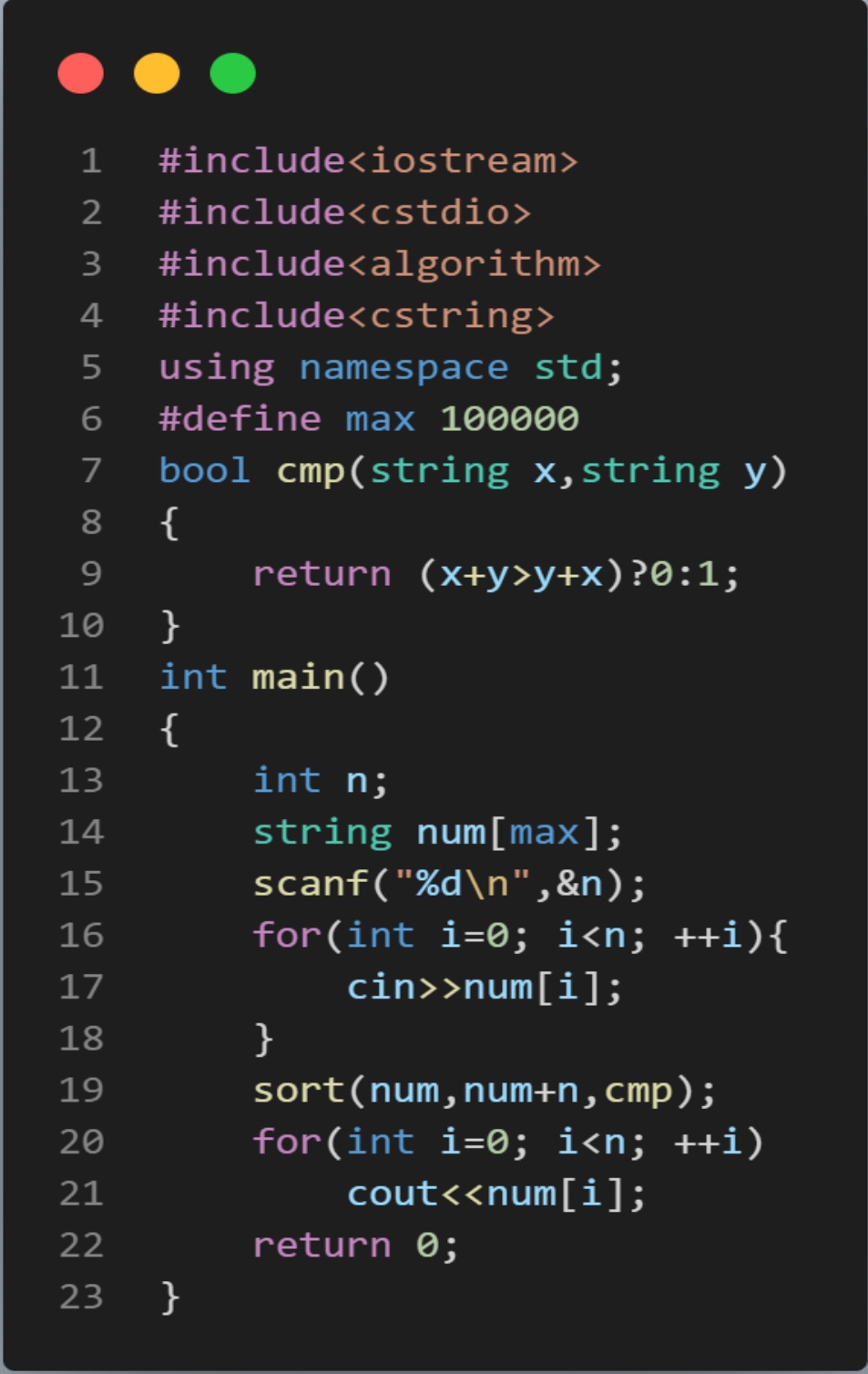
nums is an array consisting of non-negative integers, please concatenate them in certain permutation such that they can form a smallest new integer number.

Input: nums = [1, 4, 13, 2, 25]

Output: 1132254

## 1.1 Algorithm Description

考虑最高位最小的数应该排在前面，而如果有多个最高位一样最小的数，则考虑次高位比较，次高位最小的数应该排在最前面，以此类推。所以，一对数处于正确的相对位置上，当且仅当这两个数交换位置后得到的数不会变小，这等价于两数拼接所得数小于两数倒拼接所得数。所以，我们唯一需要做的就是按照字符串拼接的大小关系对输入进行排序。C++语言代码如下：



```
1  #include<iostream>
2  #include<cstdio>
3  #include<algorithm>
4  #include<cstring>
5  using namespace std;
6  #define max 100000
7  bool cmp(string x,string y)
8  {
9      return (x+y>y+x)?0:1;
10 }
11 int main()
12 {
13     int n;
14     string num[max];
15     scanf("%d\n",&n);
16     for(int i=0; i<n; ++i){
17         cin>>num[i];
18     }
19     sort(num,num+n,cmp);
20     for(int i=0; i<n; ++i)
21         cout<<num[i];
22     return 0;
23 }
```

图 1: C++ code for problem one

## 1.2 the correctness of the algorithm

我们使用调整法证明正确性。对于完全乱序拼起来的数，固定其它数，如果某对前后出现的字符串 $x$ 和 $y$ 满足 $x+y$ 小于 $y+x$ （即在我们的排序规则下是逆序的数对），那么交换两数位置，由于两数总位数不变，且我们判断两数谁更小是字典序比较的，所以交换后整体数不会变大，任意乱序拼接数一定可以通过有限步这样的逆序对交换成为有序，在此过程中拼接数不会变大。所以有序拼接一定是最小的数，另一方面，该代码已经通过算法研讨课OJ的全部测试点，正确性得证。

## 2 Problem Two

N is a non-negative integer, remove k digits from it to obtain a new number. Find the biggest possible output number.

Input: N= 147128, k = 3

Output: 728

### 2.1 Algorithm Description

先考虑最简单的情况：如果N是递减的，则只需要移除最后三位，递增反之，假设N=147128，k=3，那么需要移除两个1和4。从第一位开始，如果目前数字是递减的，那么这些数字暂时不应该移除，当遇到第一个比前面最后一个数字更大的数的时候，需要移除前面的小的数，直到前面没有比该数更小的数或者k用光了，以此类推，这需要我们使用单调栈这一数据结构。伪代码如下：

---

**Algorithm 1** k Digits Removal

---

```
1: function REMOVE DIGITS( $N, k$ )
2:   char* s = itos(N);
3:   for  $i = 0$  to  $len(s) - 1$  do
4:     if stackempty then
5:       push();
6:     else
7:       if  $s[i] \leq top(stack)$  then
8:         push();
9:       end if
10:      while  $s[i] > top(stack)$  do
11:        pop_and_delete(s);
12:         $k--$ ;
13:        if  $k \leq 0$  || stackempty then
14:          break;
15:        end if
16:      end while
17:      if  $k \leq 0$  then
18:        break;
19:      end if
20:    end if
21:  end for
22:  if  $k > 0$  then
23:    delete(s,k); //删除s的最后k的元素
24:  end if
25:  output = stoi(s);
26:  return output;
27: end function
```

---

## 2.2 the correctness of the algorithm

首先 $k$ 是固定的，所以移除之后的总位数一定，那么首先如果是递增和递减结构，易见应该移除前 $k$ 位和后 $k$ 位。每次递减列遇到折点的时候，在 $k$ 允许的情况下，但凡不删除这个折点附近应该删除的某个元素，都会导致最后得到的结果变小，此外，多出来的 $k$ 但凡不是删除新数组最后 $k$ 个数，也会导致最后得到的结果变小（因为此时 $s$ 已经是非递减的了）； $k$ 不允许的情况下，显然是优先删除高位的小数，这与我们的伪代码一致。算法正确。

### 3 Problem Three


You are given an integer array `nums`. You are initially positioned at the array's first index, and each element in the array represents your maximum jump length at that position. Return "yes" if you can reach the last index, or "no" otherwise.

Input: `nums = [2,3,1,1,4]`

Output: "yes"

#### 3.1 Algorithm Description

我们遍历整个数组，每次都更新能够跳到的最远位置，看看最远位置是否能超过数组长度，如果可以，那么自然返回True，否则必然在遍历到某一个Index的时候该Index超过了最大距离，则不成功。

A screenshot of a code editor with a dark background and light-colored text. At the top left of the editor, there are three colored circles: red, yellow, and green. The code is written in C++ and is as follows:

```
1  bool Jump(int* nums){
2      int reach = 0;
3      for (int i=0;i<len(nums);i++){
4          if (i > reach) return False;
5          reach = max(reach, i + nums[i]);
6      }
7      return True;
8  }
```

图 2: C++ code for problem three

### 3.2 the correctness of the algorithm

正确性是显然的，因为遍历数组的过程中根据当前位置和当前位置的最远距离，不断地更新能到达的最远位置（根据当前index和此处max jump distance相加，与旧的最大位置比较得到），如果这个过程中遍历到了到达不了的位置，那么肯定跳不到最后的index，如果最远位置大于等于最大index，那么显然能跳到。



## 4 Problem Four

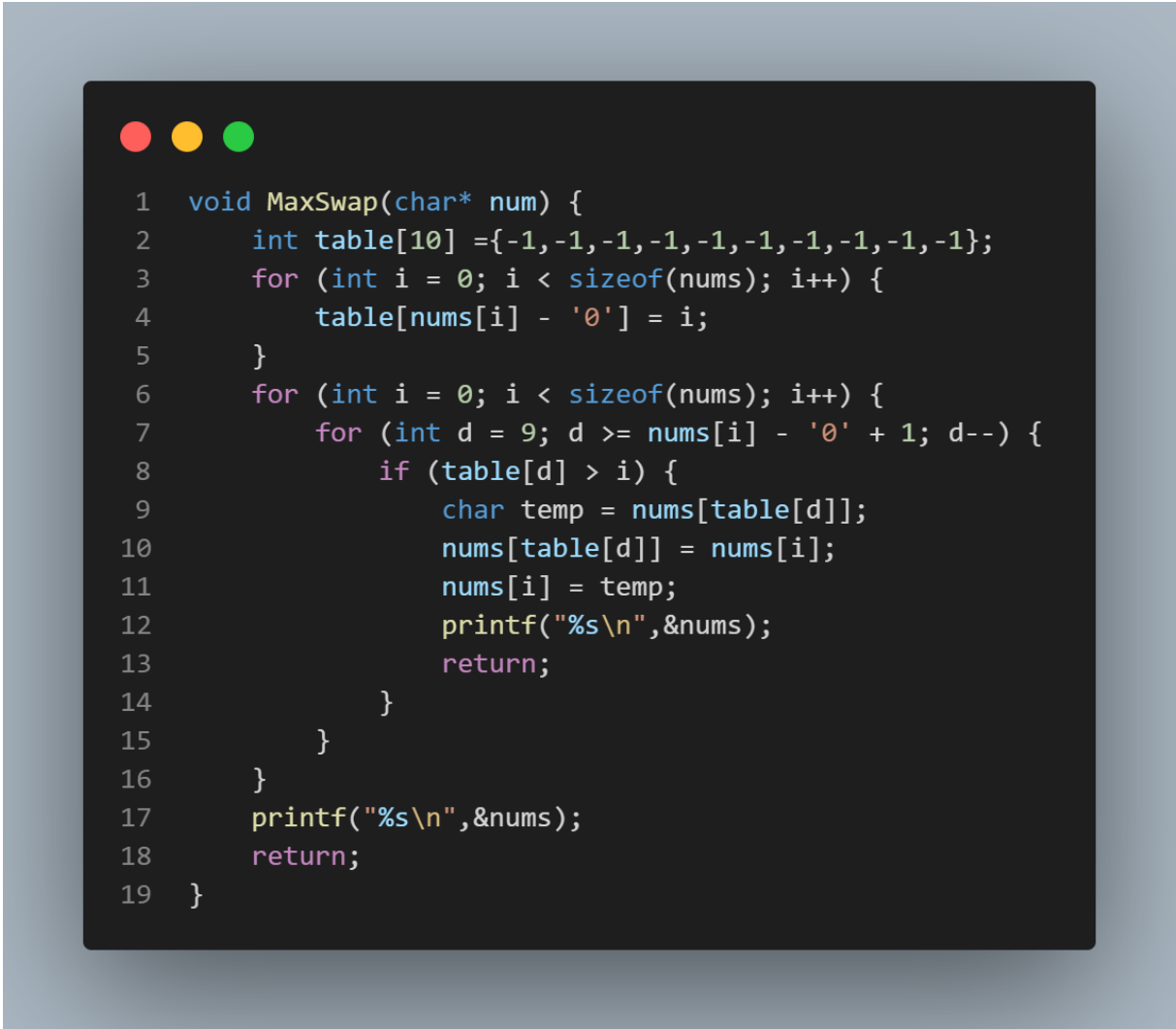
There is a number  $k$  and you can swap two digits at most once. Please design a greedy algorithm to find the maximum value you can get.

Input:  $k = 39748$

Output: 93748

### 4.1 Algorithm Description

我们采用一个表来存储0-9在数组中最后出现的index，贪心地从高位开始遍历不是9的数，然后我们希望将其换为一个更大的数，并且把它换到尽可能低位，于是贪心地从数值的大到小找比它大的数的最后出现位置，如果在它的更低位，那么两个位置进行交换，得到结果。

The image shows a C++ code snippet for solving the problem. It is presented in a dark-themed code editor with three colored window control buttons (red, yellow, green) at the top left. The code is as follows:

```
1 void MaxSwap(char* num) {
2     int table[10] = {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1};
3     for (int i = 0; i < sizeof(nums); i++) {
4         table[nums[i] - '0'] = i;
5     }
6     for (int i = 0; i < sizeof(nums); i++) {
7         for (int d = 9; d >= nums[i] - '0' + 1; d--) {
8             if (table[d] > i) {
9                 char temp = nums[table[d]];
10                nums[table[d]] = nums[i];
11                nums[i] = temp;
12                printf("%s\n", &nums);
13                return;
14            }
15        }
16    }
17    printf("%s\n", &nums);
18    return;
19 }
```

图 3: C++ code for problem four

## 4.2 the correctness of the algorithm

从最高位往低位看，显然首先遇到的所有9都不需要交换，因为这些9被交换只可能让所得数不增。从高位往低位搜索是因为在可能的情况下交换高位数字让最后结果能够更大，所以找不是9的数，再哈希出比它大的数（从9开始哈希，保证能交换的情况下这一位尽可能高）的最后出现位置，如果在低位，则直接交换即可。假设交换前高位数为 $x$ ，低位数为 $y$ ，那么根据以上算法，如果将 $x$ 换成更低位的数，或者将 $y$ 换为更小的数，或者将更高位的相同数作为 $y$ 参与交换，都会使得结果变小，所以该算法正确，所得数就是一次交换能达到的最大数。