武汉大学

WUHAN UNIVERSITY

武汉大学计算机学院

本科生课程设计报告

XXXXXXXXXXXX 实验

专 业 名 称 :XXXXXXXXXXXXX

课 程 名 称 :XXXXXXXXXXXXX

指 导 教 师 :张三　副教授

学 生 学 号 :2023XXXXXXXX

学 生 姓 名 :李四

二〇二五 年 十 月

摘要

“就是——”她走近两步，放低了声音，极秘密似的切切的说，“一个人死了之后，究竟有没有魂灵的?”

我很悚然，一见她的眼钉着我的，背上也就遭了芒刺一般，比在学校里遇到不及豫防的临时考，教师又偏是站在身旁的时候，惶急得多了。对于魂灵的有无，我自己是向来毫不介意的；但在此刻，怎样回答她好呢？我在极短期的踌躇中，想，这里的人照例相信鬼，然而她，却疑惑了，——或者不如说希望：希望其有，又希望其无……，人何必增添末路的人的苦恼，一为她起见，不如说有罢。

“也许有罢，——我想。”我于是吞吞吐吐的说。

“那么，也就有地狱了?”

“啊！地狱?”我很吃惊，只得支吾著，“地狱？——论理，就该也有。——然而也未必，……谁来管这等事……。”

“那么，死掉的一家的人，都能见面的?”

“唉唉，见面不见面呢？……”这时我已知道自己也还是完全一个愚人，甚么踌躇，甚么计划，都挡不住三句问，我即刻胆怯起来了，便想全翻过先前的话来，“那是，……实在，我说不清……。其实，究竟有没有魂灵，我也说不清。”

我乘她不再紧接的问，迈开步便走，匆匆的逃回四叔的家中，心里很觉得不安逸。自己想，我这答话怕于她有些危险。她大约因为在别人的祝福时候，感到自身的寂寞了，然而会不会含有别的甚么意思的呢？——或者是有了甚么豫感了？倘有别的意思，又因此发生别的事，则我的答话委实该负若干的责任……。但随后也就自笑，觉得偶尔的事，本没有甚么深意义，而我偏要细细推敲，正无怪教育家要说是生著神经病；而况明明说过“说不清”，已经推翻了答话的全局，即使发生甚么事，于我也毫无关系了。

“说不清”是一句极有用的话。不更事的勇敢的少年，往往敢于给人解决疑问，选定医生，万一结果不佳，大抵反成了怨府，然而一用这说不清来作结束，便事事逍遥自在了。我在这时，更感到这一句话的必要，即使和讨饭的女人说话，也是万不可省的。

但是我总觉得不安，过了一夜，也仍然时时记忆起来，仿佛怀着甚么不祥的豫

关键词： 劳仑衣；第二天；于是大家；旧历的年底毕

目录

分治法 4
1.1 数组中的第 K 个最大元素 4
1.2 寻找峰值 5
1.3 逆序对数量 6

分治法

定义：分治法

分治法（Divide and Conquer）是一种算法设计范式，它将一个复杂的问题分解成较小的子问题，递归地解决这些子问题，然后将它们的解合并以获得原始问题的解。分治法通常包括三个步骤：

1. **分解（Divide）**：将原始问题划分为若干个规模较小的子问题，这些子问题通常与原始问题具有相同的结构。
2. **解决（Conquer）**：递归地解决这些子问题。当子问题的规模足够小（通常是基本情况）时，直接求解。
3. **合并（Combine）**：将子问题的解合并成原始问题的解。

分治法在许多经典算法中得到了广泛应用，如快速排序、归并排序、二分搜索等。它通常能够显著提高算法的效率，特别是在处理大规模数据时。

1.1 数组中的第 K 个最大元素

问题描述：给定整数数组 nums 和整数 k，请返回数组中第 k 个最大的元素。

算法思想：随机给一个基准值 piviot, 容易得到大于 piviot, 等于 piviot , 小于 piviot 三部分的数量，从而将目标的“数组中的第 K 个最大元素”所可能出现的范围缩小到三者之一并继续分支，如果在等于 piviot 的区间内则直接返回

伪代码：

输入：数组 num, 整数 k。

输出：第 k 个最大元素的值。

```C++
int k_largest(vector<int> num, int k){
  return quick_sort(num, k);
}

int quick_sort(vector<int>num, int k){
  int idx = rand() % num.size();
  int x = num[idx];
  vector<int> smalls, equals, bigs;
  for (int ele in num){
    if (ele < x){
      smalls.push_back(ele);
    }else if(ele == x){
      equals.push_back(ele);
    }else{
      bigs.push_back(ele);
    }
  }
```

```
18    if (k < bigs.size()){
19      return quick_sort(bigs, k);
20    }else if(k < bigs.size() + equals.size()){
21      return equals[0];
22    }else{
23      return quick_sort(smalls, k);
24    }
25  }
```

推导：时间复杂度分析

对于长度为 N 的数组执行哨兵划分操作的时间复杂度为 $O(N)$。而向下分支子数组的平均长度是 $\frac{N}{2}$, 则平均的总时间是 $N + \frac{N}{2} + ... + 1 = 2N - 1 = O(N)$

1.2 寻找峰值

问题描述:峰值元素是指其值严格大于左右相邻值的元素。给你一个整数数组 nums，找到峰值元素并返回其索引。数组可能包含多个峰值，在这种情况下，返回 **任何一个峰值** 所在位置即可。假设 nums[−1] = nums[n] = -∞

算法思想:在边界不为峰值点时，能确定中间必有峰值点,从而在更小的区间分治，直到真正找到峰值元素。

结论：阿巴阿巴

我认为这个算法的核心思想是利用快速排序中的划分思想，通过随机选择一个基准值，将数组划分为三部分，从而有效地缩小搜索范围，最终找到第 k 个最大元素。这种方法在平均情况下具有线性时间复杂度，适用于大规模数据的处理。

其实是 AI 生成的结论。

伪代码:

```C++
int findPeakElement(vector<int>& arr) {
    int n = arr.size();
    if(n == 1) return 0;
    if(arr[0] > arr[1]) return 0;
    else if(arr[n-1] > arr[n-2]) return n - 1;
    /*在边界不为峰值点时，能确定中间必有峰值点*/
    int l = 1, r = n - 2, ans = 0;
    while(l <= r){
        int m = l + (r - l) / 2;
        if(arr[m-1] > arr[m]){
            r = m - 1;
        }else if(arr[m+1] > arr[m]){
            l = m + 1;
        }else{
            ans = m;
```

```
16              break;
17          }
18      }
19      return ans;
20  }
```

时间复杂度分析：每次二分后，待寻找的范围[l, r]都缩小一半，易知时间复杂度为$\Theta(\log n)$

1.3 逆序对数量

问题描述：：对于给定的一段正整数序列，逆序对就是序列中 $a_i > a_j$ 且 $i < j$ 的有序对。

算法思想:如果我们想要将一个序列排成从小到大有序的，那么每次划分后合并时左右子区间都是从小到大排好序的，我们只需要统计右边区间每一个数分别会与左边区间产生多少逆序对即可。由于两边分别有序，所以求解两边区间产生的逆序对数量能够很快。

伪代码:

输入: 第一行，一个数 n，表示序列中有 n 个数。第二行 n 个数，表示给定的序列。序列中每个数字不超过 10^9。

输出:输出序列中逆序对的数目

```C++
#include<bits/stdc++.h>
using namespace std;
#define int long long

int n;
const int MAX = 5e5 + 1;
int arr[MAX]={0};
int help[MAX] = {0};
int compute();
int f(int l, int r);
int merge(int l, int m, int r);
signed main(){
    cin>>n;
    for(int i=1;i<=n;i++){
        cin>>arr[i];
    }
    cout<<compute()<<endl;
    return 0;
}
int compute(){
    return f(1, n);
}
int f(int l, int r){
    if(l==r){
```

```
25          return 0;
26      }
27      int mid = l + (r - l) / 2;
28
29      return f(l, mid) + f(mid+1, r) + merge(l, mid, r);
30  }
31  int merge(int l, int m, int r){
32      int ans = 0;
33      for(int i=m, j=r; i>=l;i--){
34          while(j>=m+1&&arr[i]<=arr[j]){
35              j--;
36          }
37          ans += j-m;
38      }
39      int i = l;
40      int a = l, b = m+1;
41      while(a<=m&&b<=r){
42          help[i++] = arr[a]<=arr[b]?arr[a++]:arr[b++];
43      }
44      while(a<=m){
45          help[i++] = arr[a++];
46      }
47      while(b<=r){
48          help[i++] = arr[b++];
49      }
50      for(int i=l;i<=r;i++){
51          arr[i] = help[i];
52      }
53      return ans;
54  }
55
```

时间复杂度分析：

f 函数的时间复杂度$T(n)$满足(merge 函数的执行次数为 n):

$$T(n) = 2T\left(\frac{n}{2}\right) + n \tag{1}$$

求得 $T(n) = \Theta(n \log n)$