

# 西安电子科技大学

## 算法分析与设计（本科） 上机报告



学 院： 软件学院

专 业： 软件工程

方 向： 云计算与大数据方向

姓 名： 孙 晖

学 号： 15130120141

## 一、实验内容

按照老师要求，实现 Merge\_Sort, Insertion\_Sort, Quick\_Sort, priority queue. 并回答下列两个问题：

- [1] Implement **Quicksort** and answer the following questions. (1) How many comparisons will Quicksort do on a list of  $n$  elements that all have the same value? (2) What are the maximum and minimum number of comparisons will Quicksort do on a list of  $n$  elements, give an instance for maximum and minimum case respectively.
- [2] Give a **divide and conquer** algorithm for the following problem: you are given two sorted lists of size  $m$  and  $n$ , and are allowed unit time access to the  $i$ th element of each list. Give an  $O(\lg m + \lg n)$  time algorithm for computing the  $k$ th largest element in the union of the two lists. (For simplicity, you can assume that the elements of the two lists are distinct).

## 二、实验过程

### 2.1 实验一

#### 2.1.1 实验内容

归并排序：算法思想就是将给定的一组数，不断的二分，直到不可再分为止。创建一个新的序列用来存储排好序的一组数，然后将两个子序列分别从第一个数开始比大小，小的那个放在新序列中，且标志向后移一位，直到所有的数都被按照顺序放入新的序列中存储起来，这样就得到了从小到大的一组序列。

#### 2.1.2 实验过程

整个归并函数可以分为 merge\_sort 和 merge 两个部分，采用递归的思想写 merge\_sort，写 merge 函数时要注意边界问题，实际上边界问题是排序算法最令人头疼的问题。按照刘老师上课所讲的，记住算法的图就可以了。值得注意的是，我在用 vector 实现这个算法时一直出错，但是编译器却一直未报错，最终只得换成数组来写，错误的原因至今还不知道，但是已经上传到 github，希望有时间转回头看一下究竟错误在哪里。并且最好用 java 重新写一遍，来锻炼自己。

### 2.1.3 实验结果

```
int main() {  
    int a[] = { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 };  
    merge_sort(a, 0, 9);  
    for (int i = 0; i < 10; i++) {  
        printf("%d ", a[i]);  
    }  
    getchar();  
    return 0;  
}
```

这里在 main 函数直接限定死数组存储的数据从 10 到 1，得出结果为

```
1 2 3 4 5 6 7 8 9 10
```

从 1 到 10.

### 2.1.4 实验小结

归并排序我认为主要的是处理好两个子数组，当其中一个子数组的所有数都已经放入到第三个数组排好序时，接下来应该怎么做？就是要把这里的程序想明白，接下来把另一个数组整体的全部放进去就行了。

## 2.2 实验二

### 2.2.1 实验内容

插入排序：开始先以数组的第一个元素为标度，从接下来的那个元素开始到整个数组的所有数都和已经排好序的那个元素做比较，如果比这个元素大，那么直接插入到后面就行，如果比这个数小，那么就在这个数的前面比较，同时被比较的这个数在数组中后移一位，这样一直比较，知道找到合适的位置插入。

### 2.2.2 实验过程

从整个算法的描述就可以看出来算法是一个循环套着另一个循环，外面的循环用来控制待插入元素，每插入一个元素，就把这个外面这个循环重复一次，里面的循环用于控制待插入元素与已经排好的元素的大小比较，如果待插入元素比排好的最后一个元素大，那么直接放在尾部即可，如果比排好的最后一个元素小，那么就和前一个元素比较，整个数组往后移一位。内部循环就是用来判断大小和移位的。

### 2.2.3 实验结果

```
int main() {  
    int d[] = { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 };  
    InsertSort(d, 10);  
    for (int i = 0; i < 10; i++)  
    {  
        cout << d[i] << " ";  
    }  
    getchar();  
    return 0;  
}
```

程序中直接规定好整个数组的元素。

```
选择c:\users\lenovo\documents\visual studio 2015\Projects\insert_sort\Debug\insert_sort.exe  
1 2 3 4 5 6 7 8 9 10
```

结果输出如图所示。

### 2.2.4 实验小结

相比其他两个算法而言，这个算法是最容易写的算法。这个算法只需要记住有两个循环，以及两个循环各自是干什么的就可以了。

## 2.3 实验三

### 2.3.1 实验内容

快排：把整个待排序数组分成四个部分。

### 2.3.2 实验过程

所谓快排，就是我们以整个数组的最后一个元素为  $x$  为标志，整个数组从第一个元素到  $n-1$  个元素都和这个  $x$  去比较，然后整个数组分为四个区域，分别为 ABCD，其中：

D 就是元素  $x$ 。

C 就是还没有匹配到的元素。

B 就是大于  $x$  的元素

A 是小于  $x$  的元素。

这样看来可以有很多的变种，比如把数组第一个元素选择为  $x$ ，把 A 用于存放大

于 X，B 用于存放小于 X 等等。

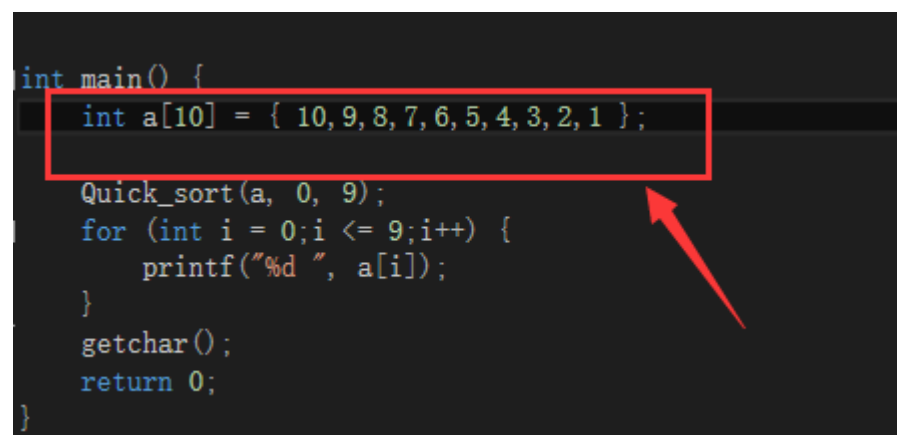
然后就是递归思想。

这里说明一下递归和迭代的区别：递归就是自己调用自己，比如函数 A 一直调用函数 A，迭代就是函数 A 不停的调用函数 B。

把一次上述所示的行为不停的递归就能够做到最后实现排好序。这里需要注意的就是要把 X 和 i+1 最后互换一下，还有就是递归的结束标志要写好，最后，quick\_sort 需要的参数有三个，一个是数组，两个 int 型，表示数组开始的下标和结束的下标。

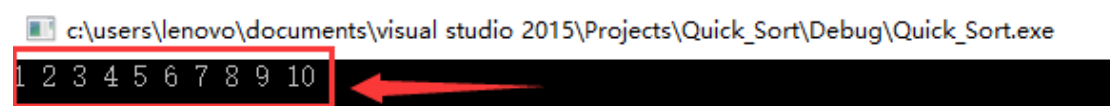
### 2.3.3 实验结果

预先规定好数组的各个元素。



```
int main() {  
    int a[10] = { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 };  
    Quick_sort(a, 0, 9);  
    for (int i = 0; i <= 9; i++) {  
        printf("%d ", a[i]);  
    }  
    getchar();  
    return 0;  
}
```

实验结果输出如下图所示：



```
c:\users\lenovo\documents\visual studio 2015\Projects\Quick_Sort\Debug\Quick_Sort.exe  
1 2 3 4 5 6 7 8 9 10
```

### 2.3.4 实验小结

对于快速排序，只需要记住老师上课的那两个图和整个数组被分为四个区域，每个区域分别代表什么即可。

## 2.4 实验四

### 2.4.1 实验内容

优先队列是一种特殊的队列，在学习堆排序的时候就有所了解。

优先队列是一种数据结构用来维护一组元素的数据结构，每个元素都有一个相关

的值，成为键。

有两种优先队列，分别是大优先队列和小优先队列。老师课堂上主要是讲基于大堆的大优先队列。

优先队列提供以下这些操作：

插入操作：将元素  $x$  插入到  $S$ 。

找最大值：会返回一个  $key$ ，这个  $key$  对应着整个队列的最大元素。

插入和删除的原理主要就是堆的“上浮操作”，“下沉操作”。

## 2.4.2 实验过程

参照网上的博客写，同时复习二叉树的相关知识，包括创建，插入，删除，先、中、后序遍历，这些全部都忘的一干二净。

## 2.4.3 实验小结

了解优先队列具体是什么，以及其使用的方法，java 中提供的有模板，学会使用即可。

## 2.5 实验五

[1] 用快排对  $n$  个数值相同的元素进行排序时进行了多少次比较。

我以 10 个元素的数组做一个例子来看。

最后一次递归，需要 9 次比较。

倒数第二次递归，需要  $3+3=6$  次比较。

倒数第三次递归，需要  $1+1=2$  次比较。

总结来看，每一次那个标志  $x$  在整个序列中间的时候，情况是最优的，也就是说，

$a_n = 2 \times a_{\left(\frac{n-1}{2}\right)} + n - 1$ ，这个公式一直递归下去，当  $n=1$  时求的和就是比较的次数，

值得注意的是，这种情况下，比较次数是最少的，因为每一次，这个标志  $x$  都是放在中间的。

[2] 显然，最优的情况（比较次数最少的情况）就是上面说的这种情况，最差的情况（比较次数最多的情况）就是顺序或者逆序的时候，也就是说，当你想要从小到大时，整个数组却偏偏是从大到小的这样一种情况。

## 2.6 实验六

规定了两个 list 都是可以在单位时间内访问到元素  $i$  的， $O(\lg m + \lg n)$  在这样一个时间复杂度内，找到两个 list 中第  $k$  个最大的元素。（用分治法）。

感觉自己还是没有这个脑子，所以在 geeksforgeeks 上搜了一下答案，明白了具体的解法，假设有着两个 list 分别为  $A$  和  $B$ ，长度分别为  $a, b$ ，上来先  $a, b$  各分一半，加一起和数字  $k$  比较。若  $(a/2 + b/2) < k$ ，就再看  $A[a/2]$  和  $B[b/2]$  谁更大，若前者大于后者，就递归整个  $A$  和

B[b/2+1...n]这个过程，其他相反的情况类似可得。上代码：

```
// Program to find k-th element from two sorted arrays
#include <iostream>
using namespace std;

int kth(int *arr1, int *arr2, int *end1, int *end2, int k)
{
    if (arr1 == end1)
        return arr2[k];
    if (arr2 == end2)
        return arr1[k];
    int mid1 = (end1 - arr1) / 2;
    int mid2 = (end2 - arr2) / 2;
    if (mid1 + mid2 < k)
    {
        if (arr1[mid1] > arr2[mid2])
            return kth(arr1, arr2 + mid2 + 1, end1, end2,
                        k - mid2 - 1);
        else
            return kth(arr1 + mid1 + 1, arr2, end1, end2,
                        k - mid1 - 1);
    }
    else
    {
        if (arr1[mid1] > arr2[mid2])
            return kth(arr1, arr2, arr1 + mid1, end2, k);
        else
            return kth(arr1, arr2, end1, arr2 + mid2, k);
    }
}

int main()
{
    int arr1[5] = {2, 3, 6, 7, 9};
    int arr2[4] = {1, 4, 8, 10};

    int k = 5;
    cout << kth(arr1, arr2, arr1 + 5, arr2 + 4, k - 1);
    return 0;
}
```

同时给出了  $\log k$  的时间复杂度解法：

<https://www.geeksforgeeks.org/k-th-element-two-sorted-arrays/>

### 三. 本次实验小结

- [1] 感觉学的 C/C++ 都还给老师了，两个函数 A 和 B，B 调用 A，结果 B 还写在 A 的前面，编译器报错才发现咋回事。
- [2] Vector 的相关知识全忘光，连咋声明都忘记了。
- [3] 排序算法的边界问题实在是令人头疼。
- [4] 老师上课把所有东西都讲清楚了，很形象，只要认真听就肯定不会有什么难点，错误的地方只能是在编写程序时粗心导致的错误。
- [5] 第五题和第六题是很有趣的问题，第五题我算不出来具体的值，但是知道这个递归的算下去肯定能算出结果，就是总结不出公式，第六题也很有趣。
- [6] 算法是一门很重要的课，最好能有多熟练就多熟练。
- [7] 因为担心整个.doc 文件会出现截图移位现象，转成 pdf 同时放入文件夹中。