1

快排:先治后分 时间复杂度:O(nlogn) 最差情况:O(n^2)

```
void quickly_sort(int arr[], int low, int high) {
  if (low < high) {
    // 先治后分
    int pivot = qucikly_sort(arr, low, high);  //治
    quickly_sort(arr, low, pivot-1); //分
    quickly_sort(arr, pivot + 1, high); //分
  }
}
```

快排最优情况: 每一次pivot都刚好可以平分整个数组
最坏情况: 每一次pivot刚好都是最大或者最小的数

快速排序的根本处理:如何在选定一个元素后，将这个元素放在合适的位置，使得元素的左边都是小于这个元素的，右边都是大于这个元素的

归并:先分后治 最好最差时间复杂度:O(nlogn)

```
void merge_sort_up2down(int a[], int start, int end) {
    if(a==NULL || start >= end)
        return ;
    // 先分后治
    int mid = (end + start)/2;
    merge_sort_up2down(a, start, mid); // 分
    merge_sort_up2down(a, mid+1, end); // 分
    merge(a, start, mid, end); // 治
}
```

归并排序的根本处理:如何将两个有序的数组，合并成一个有序的数组。遍历比较两个数组的元素，按大小放进新数组。

快排

1.原理图:

2

```
#include<stdlib.h>
#include<stdio.h>
#include<iostream>
#include<string>
#include<vector>
#include<map>
using namespace std;

//将数组的某一段元素进行划分，小的在左边，大的在右边
int divide(int a[], int start, int end){
        int base = a[end];
        while(start < end){
            while(start < end && a[start] <= base)
                //从左往右找比基准值大的
                start++;
            if(start < end){
                int temp = a[start];
                a[start] = a[end];
                a[end] = temp;
```

3

```
                end--;
            }
            while(start < end && a[end] > base)
                //从右往左找比基准值小的
                end--;
            if(start < end){
                int temp = a[start];
                a[start] = a[end];
                a[end] = temp;
                start++;
            }
        }
        return end;
    }
void sort(int a[], int start, int end){
        if(start > end)
            return;
        else{
            int partition = divide(a, start, end);
            sort(a, start, partition-1);
            sort(a, partition+1, end);
        }
    }
int main(){
    int a[] = {2,7,4,5,10,1,9,3,8,6};
    int b[] = {1,2,3,4,5,6,7,8,9,10};
    int c[] = {10,9,8,7,6,5,4,3,2,1};
    int d[] = {1,10,2,9,3,2,4,7,5,6};

    sort(a, 0, 9);

    cout << "排序的结果";
    for(int x : a){
        cout << x  << " ";
    }
    cout << endl;
    return 0;
}
```

运行结果：

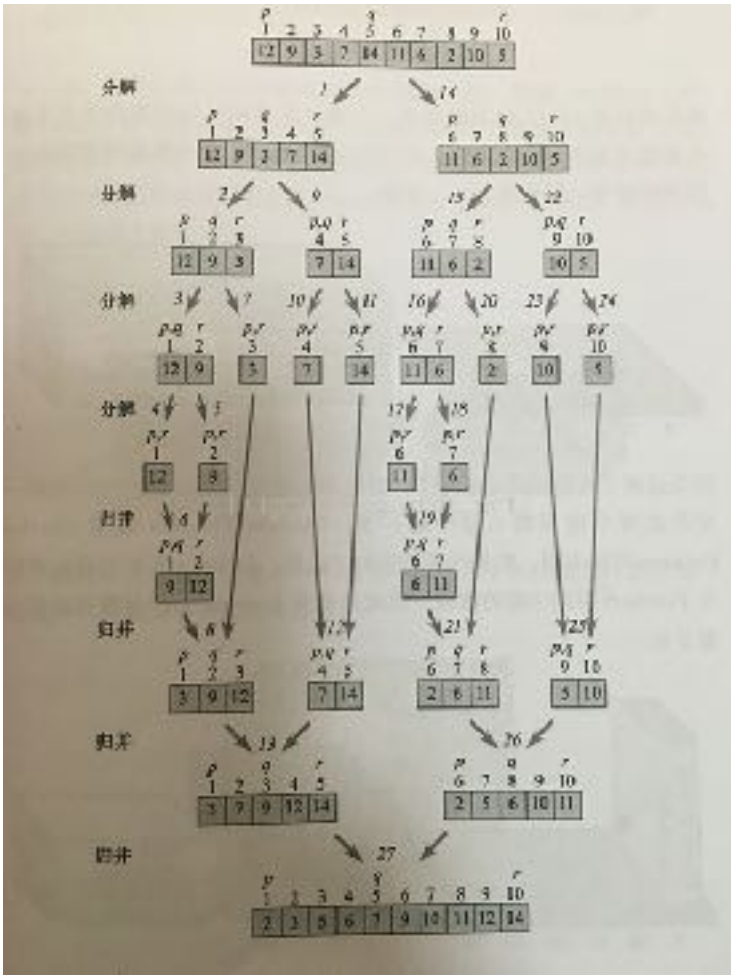
排序的结果

1 2 3 4 5 6 7 8 9 10

归并:

4

1.归并排序原理：




```
/*
合并[left,mid),[mid,right)两个有序数组
*/
void merge(int *arry, int left, int mid, int right)
{
    int *temp = new int[right - left];
    int t = 0;
    int i = left;
    int j = mid;
    while (i < mid || j < right)
    {
        if (i>= mid)
        {
            temp[t++] = arry[j++];
        }
        else if (j>= right)
        {
            temp[t++] = arry[i++];
        }
        else
        {
            if (arry[i] < arry[j])
            {
                temp[t++] = arry[i++];
```

```
            }
            else
            {
                temp[t++] = arry[j++];
            }
        }
    }
    t = 0;
    for (int i = left; i < right; i++)
    {
        arry[i] = temp[t++];
    }
    delete[] temp;
}
/*
归并排序
算法主体
参数arry，数据数组
参数left，数组起始索引
参数right，数组末端索引
*/
void mysort(int *arry, int left, int right)
{
    if (left + 1 < right)
    {
        int mid = (left + right) / 2;
        mysort(arry, left, mid);
        mysort(arry, mid, right);
        merge(arry, left, mid, right);
    }
}

int main()
{
    const int count = 15;//测试数据个数
    int *arry = new int[count];
    srand(time(0));
    for (int i = 0; i < count; i++)
    {
        arry[i] = rand() % 100;
        cout << arry[i] << endl;
    }
    cout << endl;

    mysort(arry, 0, count);//测试算法
    for (int i = 0; i < count; i++)//显示结果
    {
        cout << arry[i] << endl;
    }

    return 0;
}
```