华东师范大学数据科学与工程学院上机实践报告

课程名称：算法设计与分析　　**年级：**22 级　　**上机实践成绩：**

指导教师：金澈清　　**姓名：**郭夏辉

上机实践名称：排序算法　　**学号：**10211900416　　**上机实践日期：**2023 年 3 月 23 日

上机实践编号：No.4　　**组号：**1-416

一、目的

1．熟悉算法设计的基本思想

2．掌握计数排序、基数排序的基本思想，并且能够分析算法性能

二、内容与设计思想

1. 随机生成 1～M 范围内的 N 个整数，输入参数包括 N，M，T；可随机生成一个大小为 N、数值范围在[1, M]之间，类型为 T 的数据集合；T 包括三种类型（顺序递增、顺序递减、随机取值）；

2. 编程实现计数排序；

3. 对不同类型（顺序递增、顺序递减、随机取值）的数据集合而言，在相同 M(100000，1000000)的条件下，N 分别等于 0.1M，0.2M，0.5M，1M 时的运行时间；

4. 对不同类型（顺序递增、顺序递减、随机取值）的数据集合而言，在相同 N(100000，1000000)的条件下，M 分别等于 2N，5N，10N，20N 的运行时间；

5. 编程实现基数排序

6. 对不同类型（顺序递增、顺序递减、随机取值）的数据集合而言，在不同数据规模情况下（数据规模 N=10000, 100000, 1000000），M 分别等于 N，5N，10N，20N，算法的运行时间各是多少？

思考题：将快速排序与计数排序进行对比（自由发挥）。

三、使用环境

推荐使用 C/C++集成编译环境。

四、实验过程

1. 写出计数排序、基数排序算法；

2. 分别画出各个实验结果的折线图

4.1 随机数生成器

这里自己结合助教学长对之前实验的讲解以及自己第一次实验中遇到的问题来写了，不仅解决了随机数生成地不均匀问题，而且避免了过小的 RAND_MAX 对 rand()函数得到随机值的上限。具体的实现其实还是在做一个一个映射，只不过是到浮点数上了，高效的同时保证了精确性。

```
    for(int i=0;i<n;i++)a[i]=(double)rand()/RAND_MAX * (m)+1;
```

随机数生成器的源代码如下所示(注意生成的随机数文件第一个数字是生成的数据规模，这样方便了之后的操作)

```
#include <cstdio>
```

```cpp
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <ctime>
#include <algorithm>
#define MAXN 1000006
using namespace std;
int n,m,t;
//t=0 随机取值 t=1 顺序递增 t=2 顺序递减
int a[MAXN];
bool cmp(int a,int b){
    return b<a;
}
int main(){
    srand(time(NULL));
    cin>>n>>m>>t;
    ofstream fout1("data.txt");
    fout1<<n<<endl;
    for(int i=0;i<n;i++)a[i]=(double)rand()/RAND_MAX * (m)+1;

    if(t==1)sort(a,a+n);
    else if(t==2)sort(a,a+n,cmp);

    for(int i=0;i<n;i++)fout1<<a[i]<<" ";

    fout1.close();
    return 0;
}
```

4.2 实现计数排序

在我来看，计数排序是一种非比较型排序，突破了比较排序的时间复杂度下限，基本原理就是将数组中每个数出现的次数记录下来，然后通过倒序把数组从大到小依次从队尾排列到队首。

计数排序的步骤如下所示:

1.扫描输入数组，统计每个元素出现的次数，放入辅助数组中。辅助数组的下标表示输入数组中的元素值，而对应的元素值则表示该值在输入数组中出现的次数。

2.对辅助数组进行变换，使得辅助数组的每个元素值表示该值在有序序列中的最大索引。具体来说，可以对辅助数组进行累加，然后反向遍历辅助数组，更新每个元素值为累加值减去当前元素值，即可得到每个元素在有序序列中的最大索引。

3.反向遍历输入数组，根据每个元素的值在辅助数组中查找其在有序序列中的位置，将其放入有序序列中。

在输入长为 n 的数组，辅助数组长度为 k 时，计数排序的时间复杂度为 O(n+k)。由于计数排序需要额外的辅助数组，因此其空间复杂度为 O(k)。而且，基数排

序是一种稳定的排序算法。

以下是我的计数排序代码。

```
#include <cstdio>
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <ctime>
#include <algorithm>
#define MAXN 1000006
#define MAXM 20000006
using namespace std;
int n,maxx=-66666;
int a[MAXN],b[MAXN],tmp[MAXM];
int main(){
    clock_t startTime, endTime;
    ifstream fin("data.txt");
    ofstream fout("result.txt");

    fin>>n;
    for(int i=1;i<=n;i++){
        fin>>a[i];
        if(a[i]>maxx)maxx=a[i];
    }
    startTime=clock();
    //tmp 是全局的，自动初始化就是 0
    for(int i=1;i<=n;i++)tmp[a[i]]+=1;
    for(int i=1;i<=maxx;i++)tmp[i]+=tmp[i-1];
    for(int i=n;i>=1;i--){
        b[tmp[a[i]]]=a[i];
        --tmp[a[i]];
    }
    endTime=clock();
    for(int i=1;i<=n;i++){
        fout<<b[i]<<" ";
    }
    cout<<"Time: "<<(1000*( (double) (endTime - startTime) /
CLOCKS_PER_SEC))<<"ms"<<endl;

    fin.close();
    fout.close();
}
```

4.3 实现基数排序

在我来看，基数排序也是非比较排序算法，它的基本原理是将待排序的元素按照位数切割成不同的数字，然后按照每个位数分别进行排序，从而得到最终有序的序列。

基数排序的步骤如下所示:

1.扫描输入数组，找到数组中最大的元素，并确定其位数。例如，如果最大元素为 6666，那么它的位数为 4。

2.对于每一位，从低位到高位依次进行排序。可以使用稳定的排序算法，如计数排序或桶排序，对每个位上的数字进行排序。例如，对于一组数字来说，先根据个位进行排序，再根据十位进行排序，最后根据百位进行排序。

3.重复上述步骤，直到排序完成。

基数排序的时间复杂度为 O(d(n+k))，其中 d 是数字位数，n 是输入数组的长度，k 是每个位上可能的取值范围。由于基数排序需要进行多次稳定的排序操作，因此其时间复杂度可能比较高。但是，在位数较小的情况下，基数排序的效率还是很高的。基数排序也是一种稳定的排序算法。

以下是我的基数排序代码。我的第二步采用的是类似于桶排序的方案。与此同时，通过调用 C++的 vector,可以很方便地实现数组长度的动态调整。

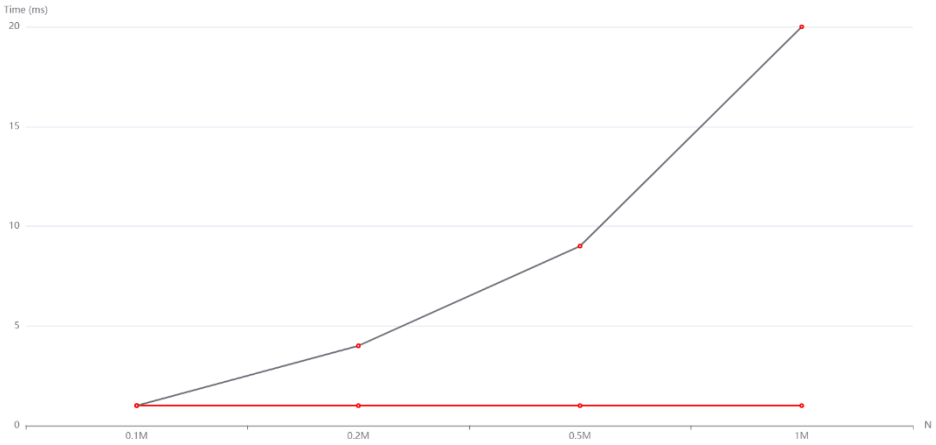
```
#include <cstdio>
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <ctime>
#include <algorithm>
#include <vector>
using namespace std;
int n;
void radixsort(vector<int>& arr);
int getdigit(int num, int digit);

int main(){
    clock_t startTime, endTime;
    ifstream fin("data.txt");
    ofstream fout("result.txt");
    fin>>n;
    vector<int> arr;
    int tmp;
    for(int i=0;i<n;i++){
        fin>>tmp;
        arr.push_back(tmp);
    }

    startTime=clock();
    radixsort(arr);
    endTime=clock();
```

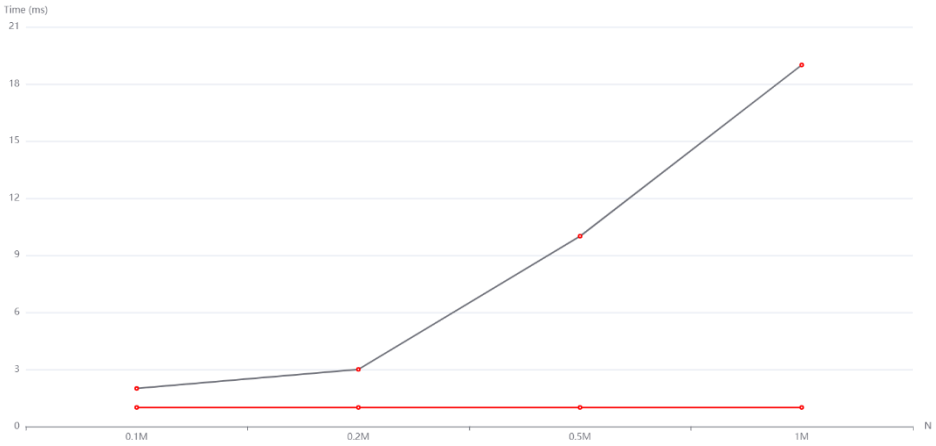
```
    cout<<"Time: "<<(1000*( (double) (endTime - startTime) /
CLOCKS_PER_SEC))<<"ms"<<endl;
    for (int i:arr)fout <<i<<" ";
    fin.close();
    fout.close();
    return 0;
}
void radixsort(vector<int>& arr) {
    if (arr.empty())return;

    int maxnum=arr[0];
    for (int num:arr) if (num>maxnum)maxnum=num;

    int maxdigit=0;
    while (maxnum>0) {
        maxnum/=10;maxdigit++;
    }

    vector<vector<int>> buckets(10);
    //毕竟是根据十进制来基数排序了，每一位的可能性也就是0-9，所以设置了十个大
桶

    for (int i=1;i<=maxdigit;i++) {
        for (int j:arr){
            int idx=getdigit(j,i);
            buckets[idx].push_back(j);
        }
        //重新放回
        int tmp_idx=0;
        for (vector<int>& bucket : buckets) {
            for (int j: bucket) {
                arr[tmp_idx++] = j;
            }
            bucket.clear();
            //最后放好了之后一定别忘了把这个桶给清空了!
        }
    }
}
int getdigit(int num,int digit) {
    //获取数字的指定位数
    int tmp=1;
    for (int i=0;(i+1)<digit; i++)tmp *= 10;
    return (num/tmp)%10;
}
```

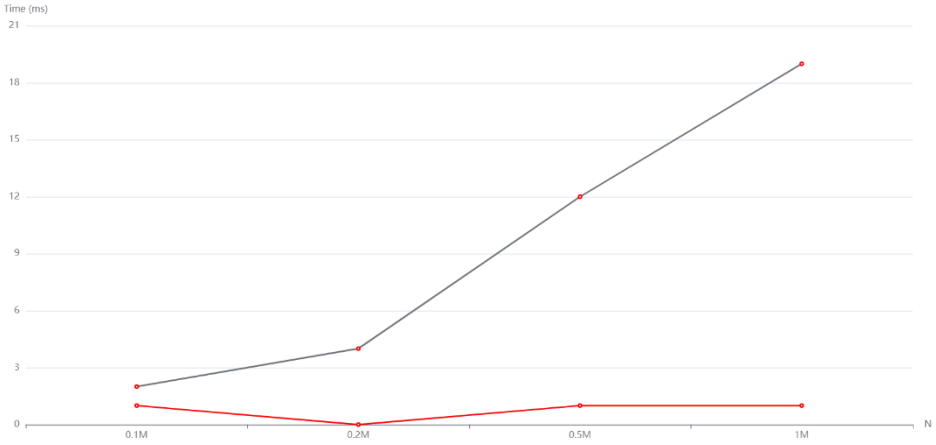
4.4 运行结果

(红线是计数排序，紫线是基数排序)

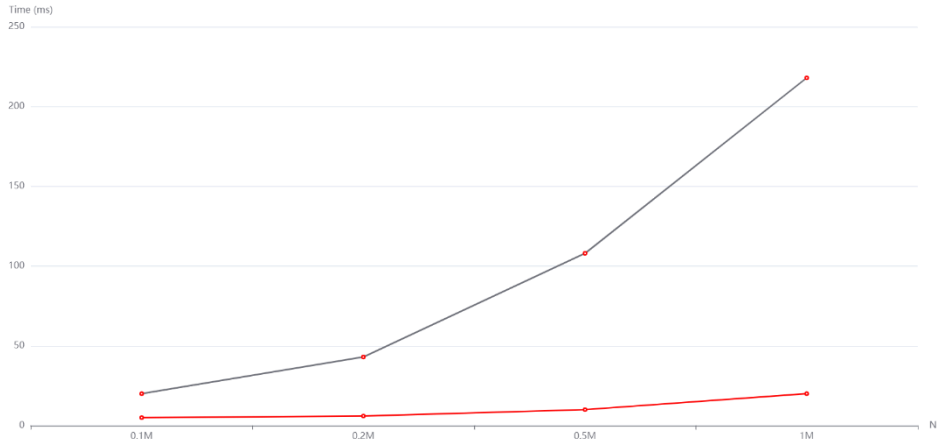
M=100000 N=0.1M，0.2M，0.5M，1M 从上到下依次是随机，递增和递减情况。

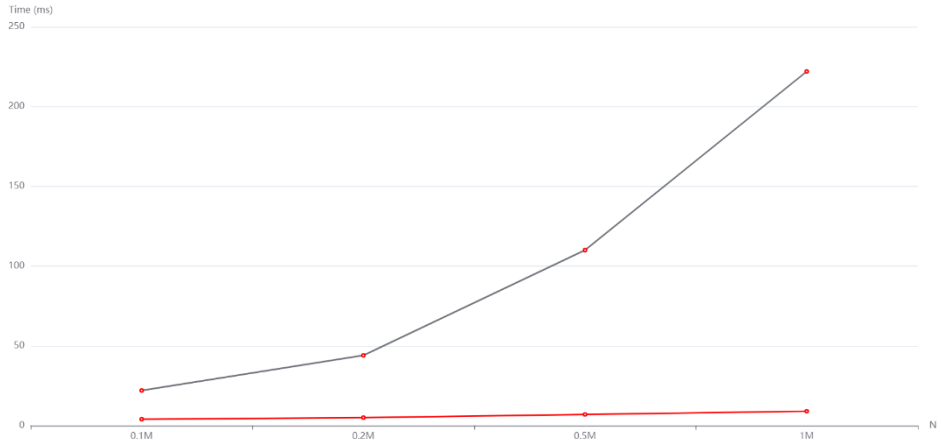




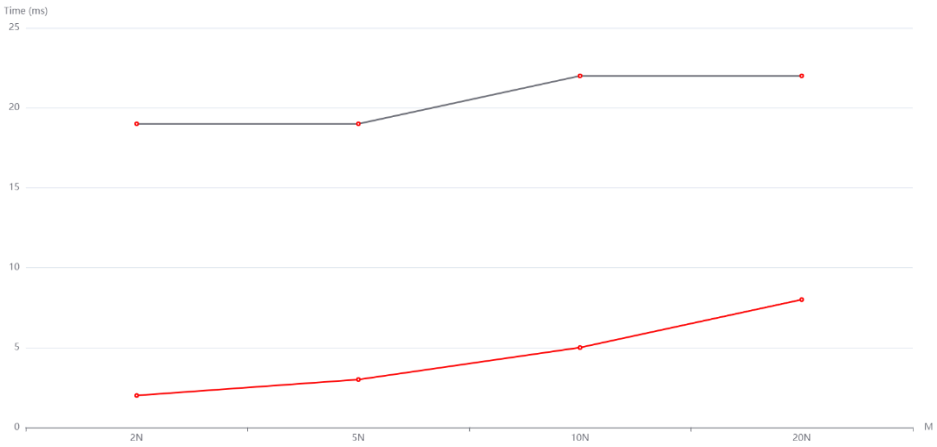


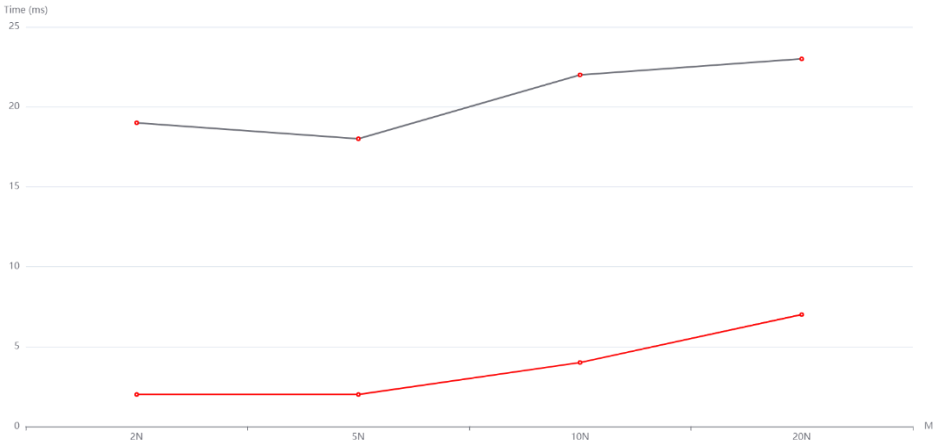
M=1000000 N=0.1M，0.2M，0.5M，1M 从上到下依次是随机，递增和递减情况。





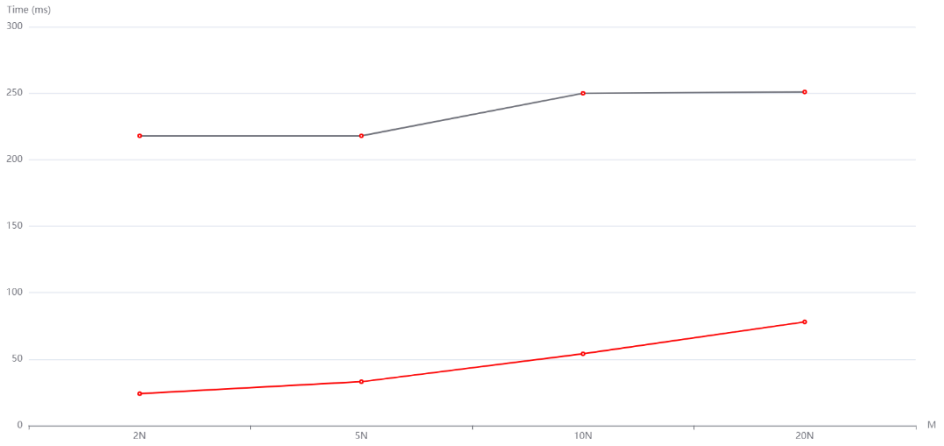
N=100000，M=2N，5N，10N，20N 从上到下依次是随机，递增和递减情况。

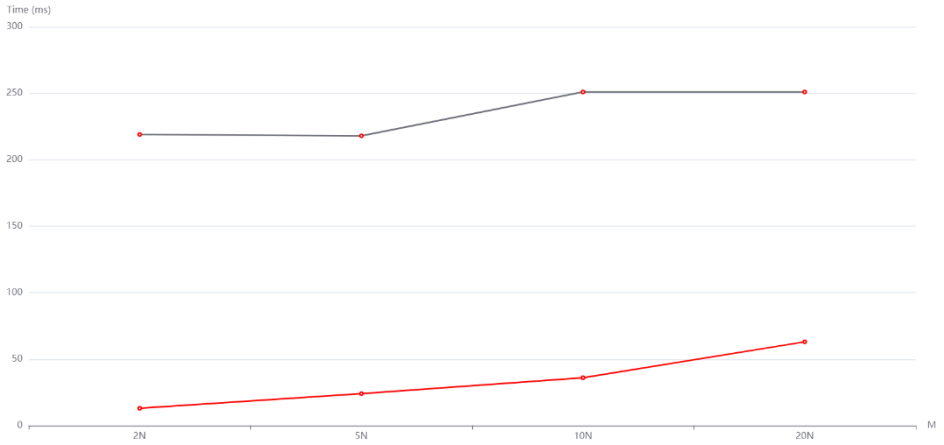

Time (ms)
25
20
15
10
5
0
2N
5N
10N
20N
M

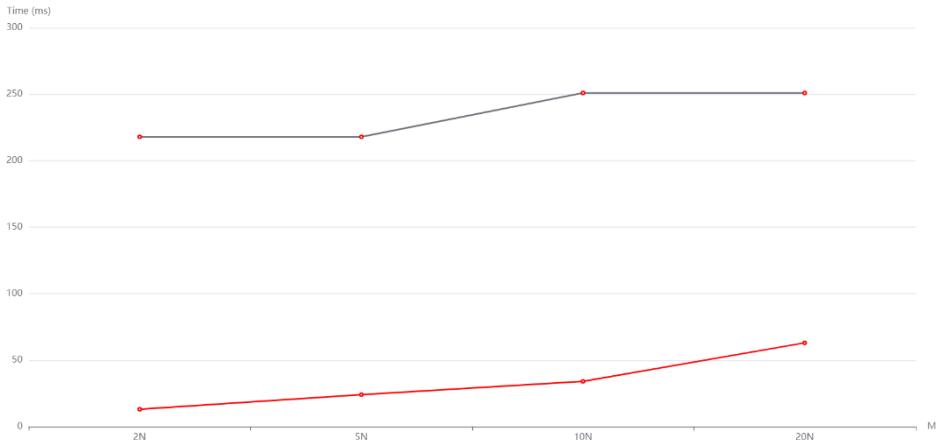

Time (ms)
25
20
15
10
5
0
2N
5N
10N
20N
M


Time (ms)
25
20
15
10
5
0
2N
5N
10N
20N
M

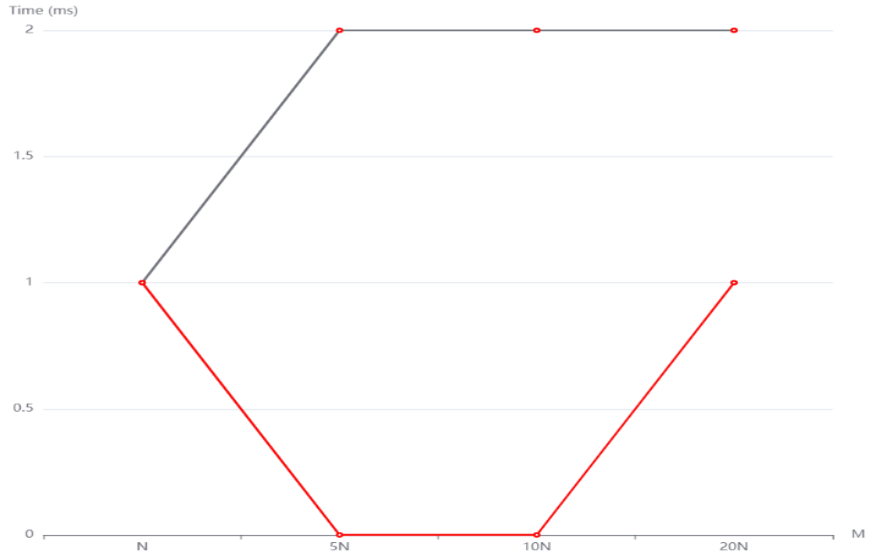
N=1000000，M=2N，5N，10N，20N 从上到下依次是随机，递增和递减情况。

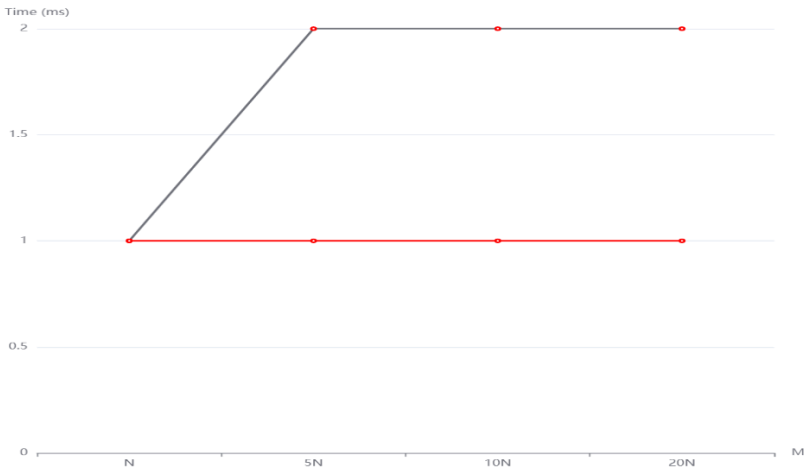


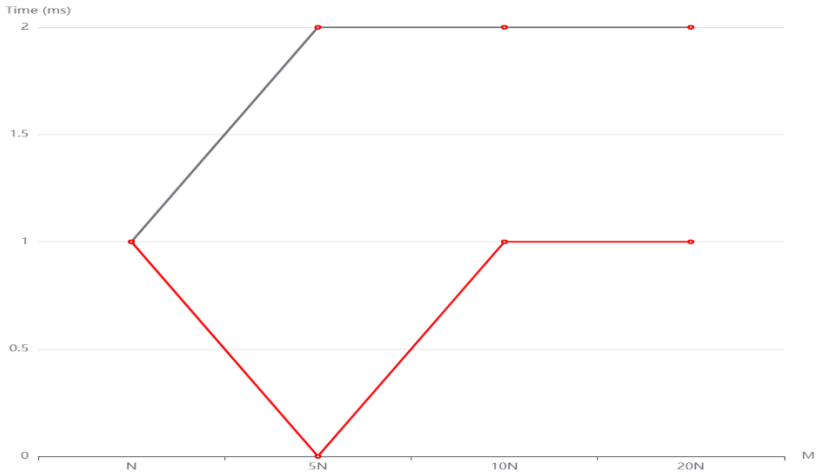




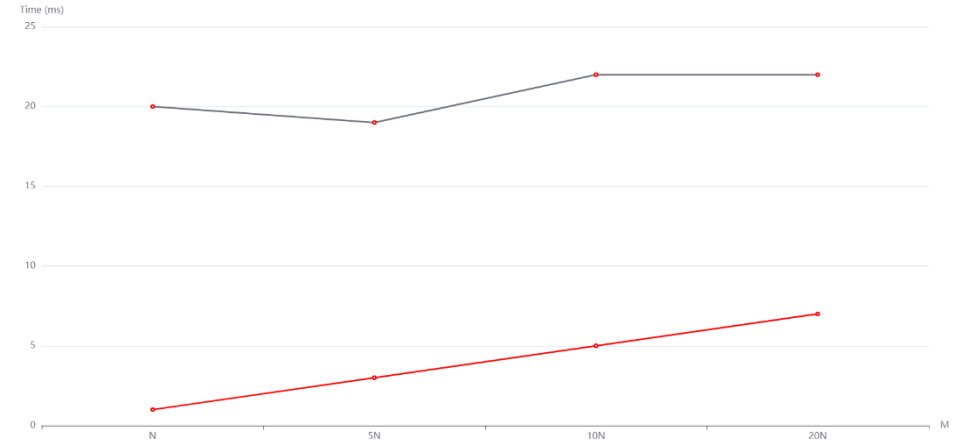
N=10000, M=N，5N，10N，20N 从上到下依次是随机，递增和递减情况。

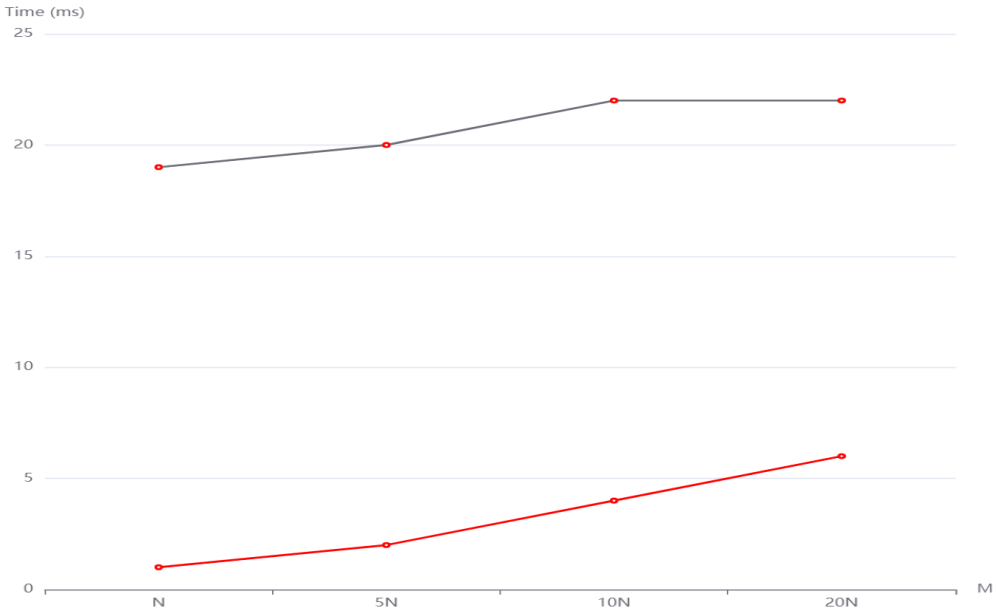


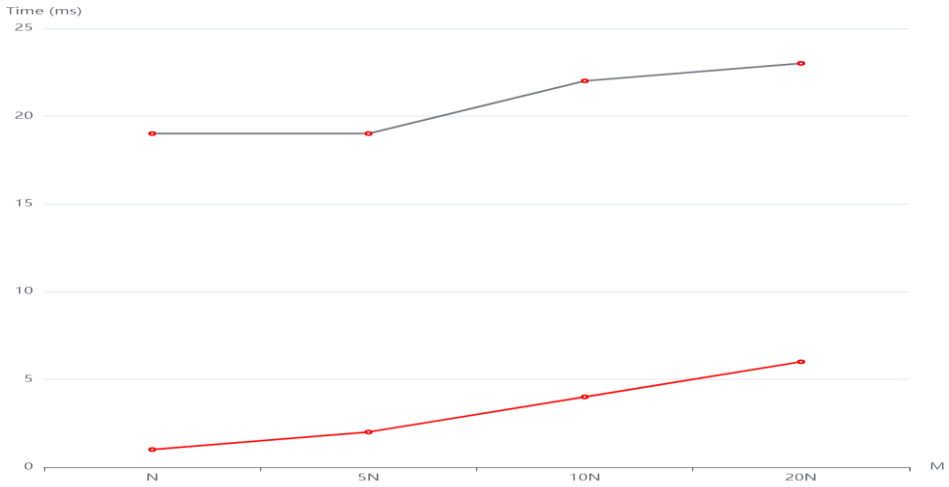




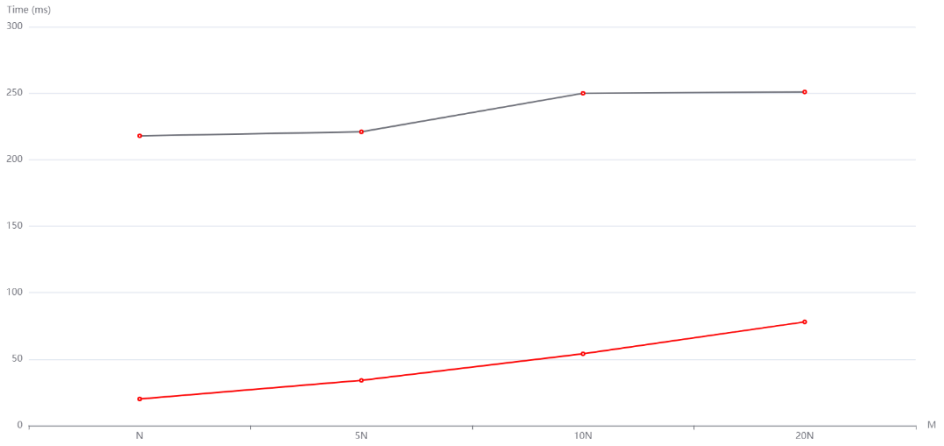
N=100000, M=N，5N，10N，20N 从上到下依次是随机，递增和递减情况。

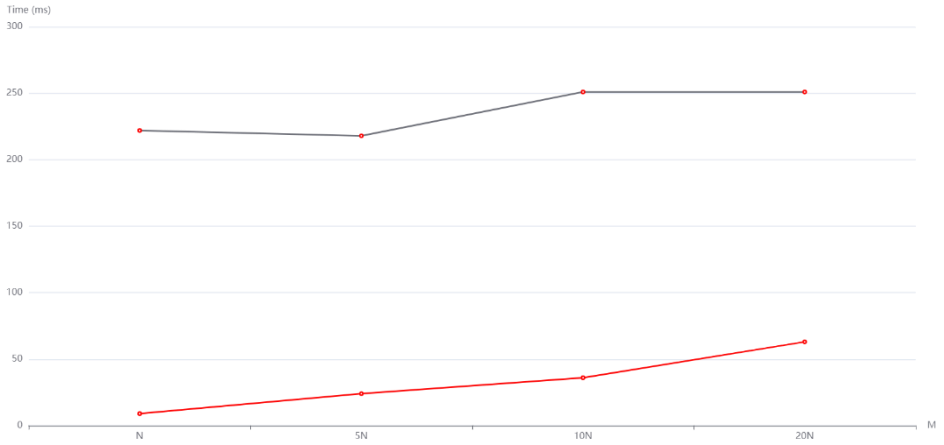


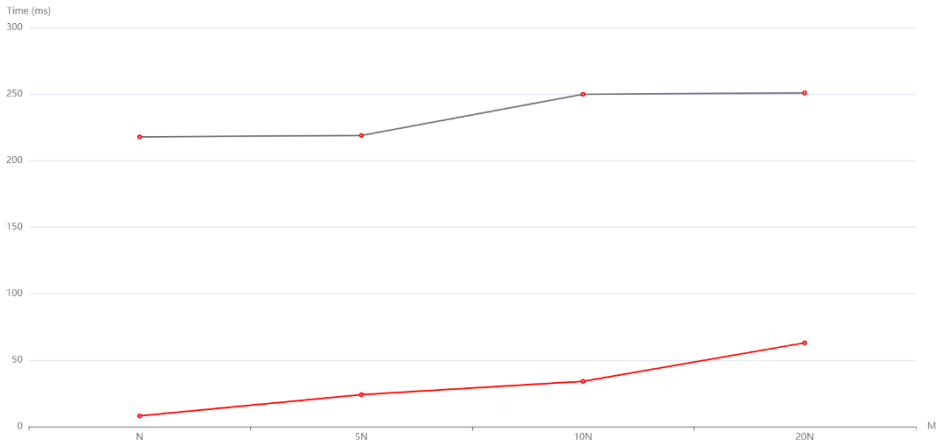




N=1000000, M=N，5N，10N，20N 从上到下依次是随机，递增和递减情况。







五、总结

对上机实践结果进行分析，问题回答，上机的心得体会及改进意见。

快速排序和计数排序的比较

1. 快速排序是不稳定排序而计数排序是一种稳定排序。
2. 快速排序的平均时间复杂度 O(nlgn)计数排序是 O(N+M)
3. 快速排序的平均空间复杂度 O(lgn)计数排序是 O(N+M)
4. 一般情况下，计数排序只能用于排非负整数，除非有特别的措施
5. 快速排序的排序速度不会因为元素取值范围的变化而变化；但是计数排序会，且受到的影响很大。
6. 总而言之，我们还是应该结合实际的数据情况选择什么类型的排序，毕竟选择排序和非选择排序的中心原理是大不相同的。虽然计数排序在理论层面是优于快速排序的，但是也有各种各样的限制，这个应该结合需求而论。

这个实验的整体难度不算大，两种线性时间排序算法也不难掌握，本次实验中最耗费时间的地方在于对于题目要求的 21 种情况分别生成数据然后作图，自己未来还是通过自动化的脚本来实现这一机械化的过程比较简洁高效。甚至可以结合一下自己最近在操作系统课程中所学的多线程并发测试。

通过各种各样情形下的测试，我发现两种排序算法的时间复杂度确实依数据规模的扩大呈近似于线性的关系，空间复杂度也确实对应它们各自的原理满足对应的关系。但是这里的线性也不是绝对的线性，计数排序还与最大值和最小值之间的差值有关，基数排序还与位数和基数有关。

实验中的 N 和 M 分别影响的是数据规模和元素位数。从生成的图中看到，随着 N 和 M 的变化，无论是随机还是递增、递减，基数排序和计数排序运行时间增长趋势都是线性的。然而有的测试点数据规模太小了，使得运行时间出现了抖动（比如数据规模变大了运行时间反而减小）。但整体上是符合我所说的性质的。而且通过实验数据，可以看到即便有 vector 容器的优化，基数排序往往是劣于计数排序的，这是因为基数排序需要进行多次稳定的排序操作，因此其时间复杂度可能比较高。但是，在位数较小的情况下，基数排序的效率还行。