

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：石季凡

上机实践名称：排序算法

学号：

上机实践日期：

10225501403

2022.3.23

上机实践编号：No.1

组号：1-403

一、目的

1. 熟悉算法设计的基本思想
2. 掌握排序算法的基本思想，并且能够分析算法性能

二、内容与设计思想

1. 设计一个数据生成器，输入参数包括 N, s, t, T ；可随机生成一个大小为 N 、数值范围在 $[s, t]$ 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 merge sort 算法和 insertion sort 算法。
3. 对于顺序递增类型的数据集合而言，在不同数据规模情况下（数据规模为 $10^2, 10^3, 10^4, 10^5, 10^6$ ）下，两种算法的运行时间各是多少？
4. 对于顺序递减类型的数据集合而言，在不同数据规模情况下（数据规模为 $10^2, 10^3, 10^4, 10^5, 10^6$ ）下，两种算法的运行时间各是多少？
5. 对于随机取值类型的数据集合而言，在不同数据规模情况下（数据规模为 $10^2, 10^3, 10^4, 10^5, 10^6$ ）下，两种算法的运行时间各是多少？
6. 补充题：编程实现 bubble sort 算法，并与上面两个算法进行对比。

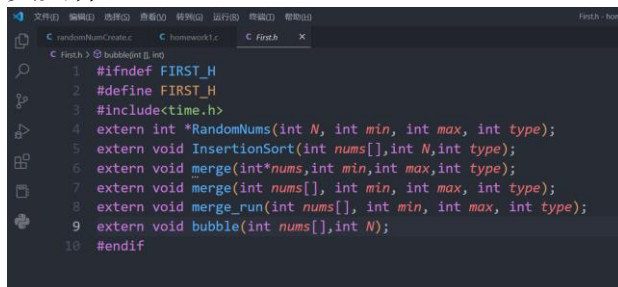
三、使用环境

推荐使用 C/C++ 集成编译环境。

四、实验过程

1. 写出数据生成器和三种算法的源代码；

头文件



```
1 #ifndef FIRST_H
2 #define FIRST_H
3 #include<time.h>
4 extern int *RandomNums(int N, int min, int max, int type);
5 extern void InsertionSort(int nums[],int N,int type);
6 extern void merge(int*nums,int min,int max,int type);
7 extern void merge(int nums[], int min, int max, int type);
8 extern void merge_run(int nums[], int min, int max, int type);
9 extern void bubble(int nums[],int N);
10 #endif
```

源代码

Bubble

```
文件(F) 编辑(E) 选择(S) 查看(V) 转到(G) 运行(R) 终端(T) 帮助(H)
C randomNumCreate.c  C homework1.c  C FirstBack.c X
C FirstBack.c > RandomNums(int, int, int)
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "First.h"
4  #include <time.h>
5  void bubble(int nums[],int N)
6  {
7      int i=0,j;
8      for(i=0;i<N;i++)
9      {
10         for(j=i+1;j<N;j++)
11         {
12             if(nums[i]>nums[j])
13             {
14                 int exchange=nums[i];
15                 nums[j]=nums[i];
16                 nums[i]=exchange;
17             }
18         }
19     }
20     return;
21 }
```

Merge

```
文件(F) 编辑(E) 选择(S) 查看(V) 转到(G) 运行(R) 终端(T) 帮助(H)
C randomNumCreate.c  C homework1.c  C FirstBack.c X
C FirstBack.c > RandomNums(int, int, int)
21 }
22 void merge(int nums[], int min, int max, int type)
23 {
24     if (type == 1)
25     {
26         int mid = (min + max) / 2;
27         int *tmp = (int *)malloc(sizeof(int) * (max - min + 1));
28         int i = min, j = mid + 1, index = 0;
29         while (i <= mid && j <= max)
30         {
31             if (nums[i] <= nums[j])
32                 tmp[index++] = nums[i++];
33             else
34                 tmp[index++] = nums[j++];
35         }
36         while (i <= mid)
37             tmp[index++] = nums[i++];
38         while (j <= max)
39             tmp[index++] = nums[j++];
40         i = min, index = 0;
41         while (i <= max)
42         {
43             nums[i++] = tmp[index++];
44         }
45         free(tmp);
46         tmp=NULL;
47     }
48     else return;
49 }
50 void merge_run(int nums[], int min, int max, int type)
51 {
52     if (min >= max || nums==NULL)
53         return;
54     int mid=(min+max)/2;
55     merge_run(nums,min,mid,type);
56     merge_run(nums,mid+1,max,type);
57     merge(nums,min,max,type);
58     return;
59 }
```

Insertion

```
60 void InsertionSort(int nums[], int N, int type)
61 {
62     if (type == 1)
63     {
64         int i = 0;
65         for (i = 1; i < N; i++)
66         {
67             int key = nums[i];
68             int j = i - 1;
69             for (; j >= 0; j--)
70             {
71                 if (key >= nums[j])
72                 {
73                     nums[j + 1] = key;
74                     break;
75                 }
76                 nums[j + 1] = nums[j];
77             }
78             if (j == -1)
79                 nums[0] = key;
80         }
81     }
82     else
83     {
84         int i = N;
85         for (i = 1; i < N; i++)
86         {
87             int key = nums[i];
88             int j = i - 1;
89             for (; j >= 0; j--)
90             {
91                 if (key <= nums[j])
92                 {
93                     nums[j + 1] = key;
94                     break;
95                 }
96                 nums[j + 1] = nums[j];
97             }
98             if (j == -1)
99                 nums[0] = key;
100         }
101     }
102     return;
103 }
104 int *RandomNums(int N, int min, int max, int type)
105 {
106     int i = 0;
107     int *ReturnNums = (int *)malloc(sizeof(int) * N);
108     srand((unsigned)time(NULL));
109     for (i = 0; i < N; i++)
110     {
111         *ReturnNums + i = min + rand() % (max - min + 1);
112     }
113     if (type == 1)
114         merge_run(ReturnNums, 0, N-1, 1);
115     else if (type == -1)
116         merge_run(ReturnNums, 0, N-1, -1);
117     return ReturnNums;
118 }
```

RandomNumsCreate

```
104 int *RandomNums(int N, int min, int max, int type)
105 {
106     int i = 0;
107     int *ReturnNums = (int *)malloc(sizeof(int) * N);
108     srand((unsigned)time(NULL));
109     for (i = 0; i < N; i++)
110     {
111         *(ReturnNums + i) = min + rand() % (max - min + 1);
112     }
113     if (type == 1)
114         merge_run(ReturnNums, 0, N-1, 1);
115     else if (type == -1)
116         merge_run(ReturnNums, 0, N-1, -1);
117     return ReturnNums;
118 }
```

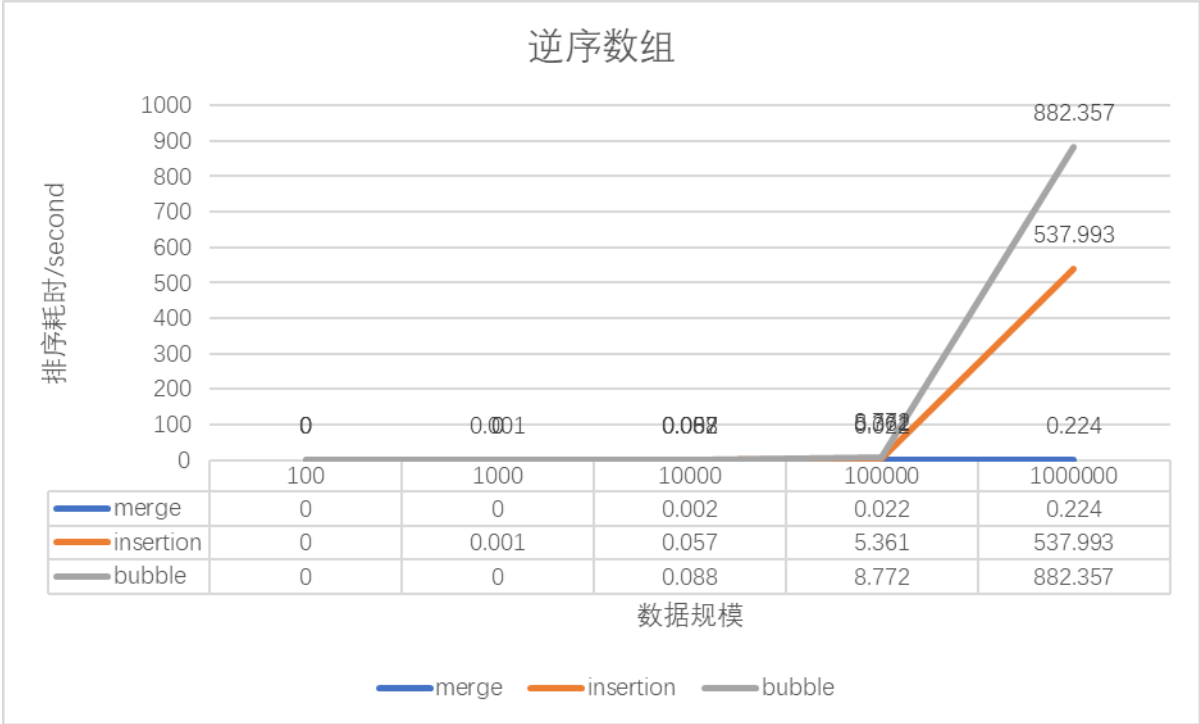
主函数代码

```
1 #include "First.h"
2 #include <stdio.h>
3 #include <stdlib.h>
4 //1顺序 -1逆序 0随机
5 int main()
6 {
7     clock_t start_merge, end_merge, start_insertion, end_insertion, start_bubble, end_bubble; //定义6个时间变量分别记录三个算法的起始于完成时间
8     int N = 1000000, min = 0, max = 1000000, type = -1;
9     scanf("%d %d %d", &N, &min, &max, &type);
10    printf("数据规模: %d 数据排列类型: %d\n", N, type);
11    int *nums = (int *)malloc(sizeof(int) * N);
12    nums = RandomNums(N, min, max, type); //根据四个变量生成随机数
13    int *UsingNums = (int *)malloc(sizeof(int) * N);
14    int i=0;
15    ;
16    //merge
17    for(i=0;i<N;i++) UsingNums[i]=nums[i]; //初始化要使用的数组
18    start_merge = clock();
19    merge_run(UsingNums, 0, N - 1, 1);
20    end_merge = clock();
21    printf("time_merge=%lf\n", (double)(end_merge - start_merge) / CLOCKS_PER_SEC);
22    //insertion
23    for(i=0;i<N;i++) UsingNums[i]=nums[i]; //初始化要使用的数组
24    start_insertion=clock();
25    InsertionSort(UsingNums,N,1);
26    end_insertion=clock();
27    printf("time_insertion=%lf\n", (double)(end_insertion - start_insertion) / CLOCKS_PER_SEC);
28    //bubble
29    for(i=0;i<N;i++) UsingNums[i]=nums[i]; //初始化要使用的数组
30    start_bubble=clock();
31    bubble(UsingNums,N);
32    end_bubble=clock();
33    printf("time_bubble=%lf\n", (double)(end_bubble - start_bubble) / CLOCKS_PER_SEC);
34    free(nums);
35    nums=NULL;
36    return 0;
37 }
```

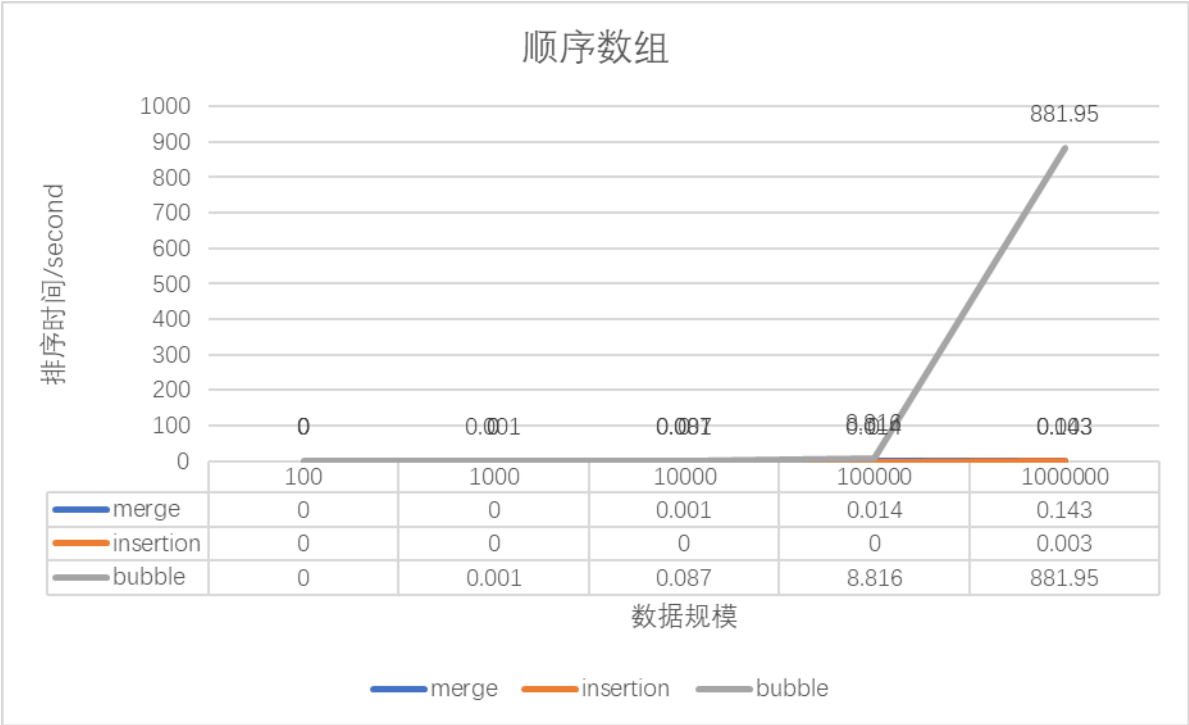
2. 分别画出各个实验报告的折线图

运行结果

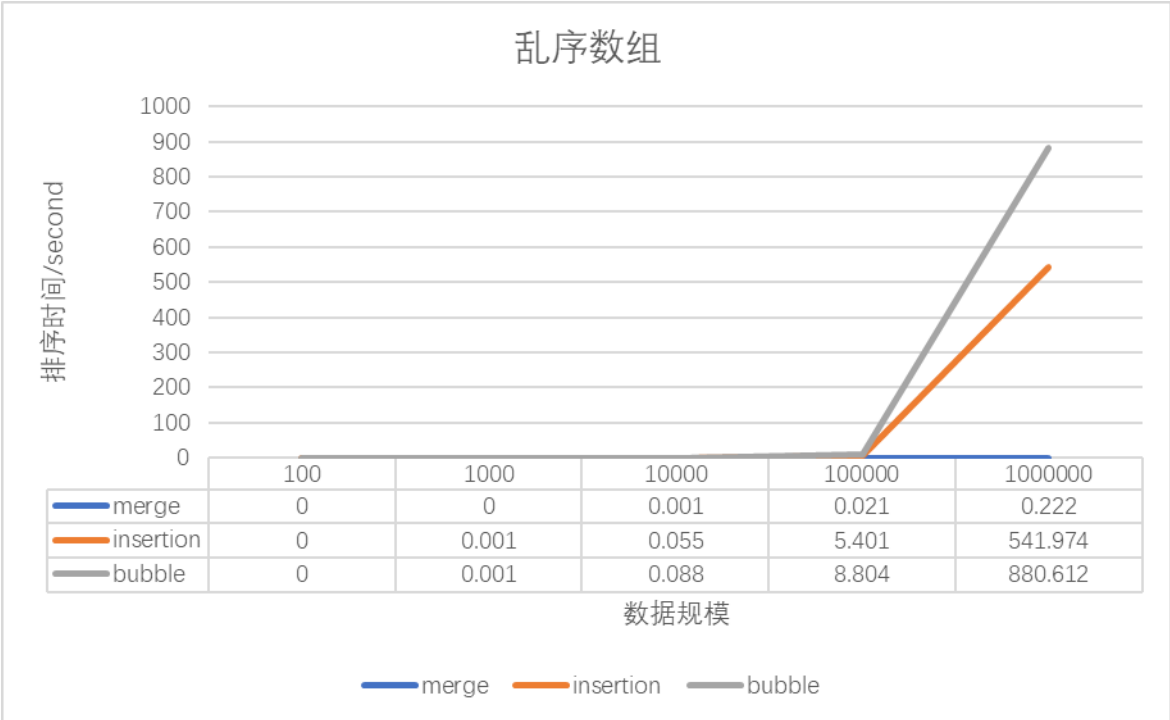
当所排序的数组为逆序时得到如下结果



当所排序数组为顺序数组时得到如下结果



当所排序的数组为乱序数组时



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。

本次实验由于涉及多个自定义函数，因此我将代码分为了三个文件，利于管理和调试。

对于各个算法而言，归并排序的算法时间复杂度为 $O(n \lg n)$ ，其中 \lg 是以 2 为底数，插入排序的算法时间复杂度为 $O(n^2)$ ，冒泡排序算法的时间复杂度为 $O(n^2)$ （以上时间复

杂度均为最坏情况，即逆序排序）

同时可以看到我们上面的实验数据，对于逆序数组这一组而言，分别都基本了各自的时间复杂度函数，如 merge 算法，其每一组相邻数据的比值满足 $n_1 \lg n_1 / n_2 \lg n_2 = 10 \lg n_1 / \lg n_2$ ，而 insertion 算法与 bubble 算法每组相邻的数据比值都约等于 10^2 ，满足 $O(n^2)$ 。同时，由于 insertion 算法的特性，导致其在进行顺序数组的排序时有着其他两种算法无法比拟的优势，但是也仅限于顺序的这种极为罕见的情况下。

根据以上的实验结果来看，在时间上最为高效的算法必然是 merge 算法，其不管是顺序乱序还是倒序，都有非常快的排序速度，在乱序以及倒序的情况下对于另外两种算法堪称“降维打击”的优势。而对于空间的占用来看，insertion 算法与 bubble 算法都是在原数组上进行操作，而 merge 算法则每一次合并都需要新申请一次空间，占用的内存空间较另外两种算法多。

这次上机实验让我对于三种排序算法有了更深的理解，同时对于空间与时间的关系有了初步的了解。

改进意见：实验所测的数据分度跨度过大，即每次数量级以 10 倍增长，这导致画出的数据图像不能很明显的与 $y=x^2$ 、 $y=n \lg n$ 契合。建议应当以固定数字为分度进行实验。