数据结构实验报告

学号-姓名	桑龙龙-20030540015	实验时间	2020 年 12 月 12 日
诚信声明	本实验及实验报告所写内容为本人所作 桑龙龙		
实验题目	实验六 常用的排序方法 题目一 简单排序方法 题目二 快速排序 题目三 堆排序 题目四 归并排序		
实验过程中遇到的主要问题	无		
实验小结	本次试验进行了常见排序算法的学习。简单排序方法有直接插入排序，选择排序，冒泡排序，他们的时间复杂度都是 O(n2)，一般不不会作为我们的排序选择。快速排序、堆排序、归并排序都有优秀的时间复杂度 O（nlog2n），其中归并排序是稳定的排序算法，快速排序、堆排序都不是稳定的排序算法。以前认识中，快速排序是相较其他两者较为优秀的排序算法，但是在 1e6 及以下数据的测试下，归并排序与其他两者相比有更低的时间消耗，相比起来较小的数据下，归并排序的额外空间消耗并不重要。在测试中，归并排序，堆排序有更为稳定的性能，不会因为极端数据导致时间复杂度降低，依旧保持在 O（nlog2n），但是快速排序在极端情况下时间复杂度会降低为 O(n2)。测试数据的结果见下表。		

0（n2）排序方式								
排序方式	数据描述	数据规模	组数	总耗时（ms）	总比较次数	总交换次数	平均比较次数	平均交换次数
选择排序	随机数据	1000	100	288	-	-	-	-
直接插入排序	随机数据	1000	1000	1028	8577408	245394856	8577.408	245394.9
	正序	1000	1000	72	8977000	0	8977	0
	逆序	1000	1000	1976	7990689	499489906	7990.689	499489.9
冒泡排序	随机数据	1000	100	584	49879385	24833792	498793.9	248337.9
	正序	1000	100	4	99900	0	999	0
	逆序	1000	100	748	49949900	49949104	499499	499491

快速排序							
数据描述	数据规模	组数	总耗时（ms）	总比较次数	总交换次数	平均比较次数	平均交换次数
随机数据	10000	100	224	15246177	3159284	152461.8	31592.84
	10000	100	220	16190000	3135100	161900	31351
	10000	100	216	15512900	3143800	155129	31438
正序	10000	1	240	49995000	9999	49995000	9999
逆序	10000	1	232	49995000	9999	49995000	9999
随机数据	100000	10	272	19996200	3921627	1999620	392162.7
	100000	10	268	20527240	3912790	2052724	391279
	100000	10	288	19778060	3935520	1977806	393552
随机数据	1000000	1	316	24273048	4712212	24273048	4712212

归并排序					
数据描述	数据规模	组数	排序总耗时（ms）	总比较次数	平均比较次数
随机数据	10000	100	196	12047200	1204.72
	10000	100	204	12045777	1204.5777
	10000	100	188	12045600	1204.56
随机数据	100000	10	236	15361180	153.6118
随机数据	1000000	1	280	18673989	18.673989
随机数据	1000000	1	268	18674525	18.674525
正序	10000	100	136	6467100	646.71
逆序	10000	100	120	6900800	690.08

堆排序（大顶堆）							
数据描述	数据规模	组数	总耗时（ms）	总比较次数	总交换次数	平均比较次数	平均交换次数
随机数据	10000	100	444	17275000	11958700	172750	119587
	10000	100	440	17265006	11946978	172650.06	119469.78
	10000	100	456	17285014	11957550	172850.14	119575.5
	100000	10	480	20793880	13974830	2079388	1397483
	100000	10	484	20794280	13974460	2079428	1397446
	100000	10	484	20793570	13973830	2079357	1397383
	1000000	1	588	25765564	17332930	25765564	17332930
由大到小插入	10000	100	380	15788000	10669700	157880	106697
	10000	100	388	15787100	10668500	157871	106685
	10000	100	388	15787515	10671405	157875.15	106714.05
	100000	10	484	20794110	13974830	2079411	1397483
	100000	10	480	20793000	13974370	2079300	1397437
	100000	10	484	20794350	13974820	2079435	1397482
	1000000	1	604	25766214	17333321	25766214	17333321
由小到大插入	10000	100	680	26684476	21719359	266844.76	217193.59
	10000	100	668	26681900	21716600	266819	217166
	10000	100	604	26687700	21726300	266877	217263
	100000	10	784	35024745	28371000	3502474.5	2837100
	100000	10	780	35024339	28370084	3502433.9	2837008.4
	100000	10	792	35024100	28370289	3502410	2837028.9
	1000000	1	988	43283001	34894619	43283001	34894619

数据结构 （自定义数据类型）	无
主要算法 （或算法说明）	

```
/*
    1、选择排序
    2、直接插入排序
    3、冒泡排序
    4、快速排序
*/

#include <stdio.h>
#include <stdlib.h>

template <typename T>
void selectSort(T* arr,int lower,int upper,bool(*cmp)(T&,T&)){

    //选择排序
    //对[arr+lower,arr+upper)范围进行排序
    int v;
    for(int i=upper-1;i>=lower;i--){
        v=i;
        for(int j=lower;j<i;j++){
            if(cmp(arr[v],arr[j])) v=j;
        }
        std::swap(arr[i],arr[v]);
    }
}

template <typename T>
void insertSort(T* arr,int lower,int upper,bool(*cmp)(T&,T&)){

    //直接插入排序
    //对[arr+lower,arr+upper)范围进行排序
    int low,high,mid,tar;
    T temp;
    for(int i=lower+1;i<upper;i++){
        low=lower,high=i-1,tar=i,temp=arr[i];
        while(high>=low){
            //二分查找查找插入位置
            mid=low+(high-low)/2;
            if(cmp(arr[i],arr[mid])){
                tar=mid;
                high=mid-1;
            }else{
```

```
                low=mid+1;
            }
        }
        for(int j=i-1;j>=tar;j--) arr[j+1]=arr[j];
        arr[tar]=temp;
    }
}

template <typename T>
void bubbleSort(T* arr,int lower,int upper,bool(*cmp)(T&,T&)){

    //冒泡排序
    //对[arr+lower,arr+upper)范围进行排序
    bool flag=true;
    /*
        标识是否已经有序
        如果有序提前退出
    */
    for(int i=upper-1;i>1 && flag;i--){
        flag=false;
        for(int j=0;j<i;j++){
            if(cmp(arr[j+1],arr[j])) std::swap(arr[j+1],arr[j]),flag=true;
        }
    }
}


template <typename T>
void quickSort(T* arr,int lower,int upper,bool(*cmp)(T&,T&)){

    //快速排序
    //排序范围[arr+lower,arr+upper)
    //以 arr[lower]作为 pivot
    if(lower>=upper) return;
    int low=lower,high=upper-1;
    while(high>low){
        while(high>low && !cmp(arr[high],arr[lower])) high--;

        while(high>low && !cmp(arr[lower],arr[low])) low++;
        std::swap(arr[low],arr[high]);
    }
    std::swap(arr[low],arr[lower]);
    quickSort(arr,lower,low,cmp);
    quickSort(arr,low+1,upper,cmp);
```

```
}

int main(){
}
```

```cpp
/*
    5、归并排序
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define N 100005
using T=int;
T arr[N],temp[N];
void mergeSort(T* arr,int lower,int upper,bool (*cmp)(T&,T&b))
{
    //归并排序
    //排序范围[arr+lower,arr+upper)
    if(1+lower>=upper) return;
    int mid=lower+(upper-lower)/2;
    mergeSort(arr,lower,mid,cmp);
    mergeSort(arr,mid,upper,cmp);
    int i=lower,j=mid,k=lower;
    while(i<mid && j<upper){
        if(cmp(arr[i],arr[j])) temp[k++]=arr[i++];
        else temp[k++]=arr[j++];
    }
    while(i<mid) temp[k++]=arr[i++];
    while(j<upper) temp[k++]=arr[j++];
    memcpy(arr+lower,temp+lower,sizeof(T)*(upper-lower));
}

int main(){

}
```

```
/*
    6、堆排序
*/
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <limits.h>
#define N 100005

template <class T>
class heap{
    private:
        bool (*cmp)(T&,T&);
        //堆排序的比较函数
        int tot,capacity;
        //tot 是目前元素个数，capacity 是堆排序实际大小
        T* arr;
        bool full(){
            //堆空间是否已满
            return tot==capacity-1;
        }
    public:
        heap(bool(*cmp)(T&,T&)){
            this->cmp=cmp;
            arr=new T[N];
            tot=0;
            capacity=N;
        }
        heap(int n,bool(*cmp)(T&,T&)){
            this->cmp=cmp;
            arr=new T[n];
            tot=0;
            capacity=n;
        }
        ~heap(){
            delete []arr;
        }
        void clear(){
            //清空堆中元素
            tot=0;
        }

        bool push(T val){
            /*
```

```
                将 val 插入到堆中，插入成功返回 true
                失败返回 false
            */
            if(full()) return false;
            arr[++tot]=val;
            int i=tot;
            while(i!=1 && cmp(arr[i/2],arr[i])){
                std::swap(arr[i/2],arr[i]);
                i/=2;
            }
            return true;
        }
        T& top(){
            //返回堆顶元素
            return arr[1];
        }
        bool empty(){
            //检查堆是否为空
            return tot==0;
        }
        int size(){
            //返回堆大小
            return tot;
        }
        void pop(){
            //删除堆顶元素
            if(empty()) return;
            std::swap(arr[1],arr[tot]);
            tot=max(tot-1,0);
            int i=1;
            while(i*2<=tot){
                i*=2;
                if(i+1<=tot && cmp(arr[i],arr[i+1])) i++;
                if(cmp(arr[i/2],arr[i])) std::swap(arr[i/2],ar
r[i]);
                else break;
            }
        }
};


bool cmp(int &a,int &b){
    return a<b;
}
void randTest(int n,heap<int>& Q){
```

```
    srand(time(0));
    Q.clear();
    printf("insert:\n");
    int v;
    for(int i=0;i<n;i++){
        //建立堆
        v=rand()%INT_MAX;
        printf("%d ",v);
        Q.push(v);
    }
    printf("\n\nheap sorted:\n");
     while(!Q.empty()){
         //弹出元素，堆排序完成
         printf("%d ",Q.top());
         Q.pop();
     }
}
int main(){
    heap<int> Q(10000,cmp);
    //创建一个空间为 10000 的大顶堆
    randTest(20,Q);
    //测试
}
```