《算法设计与分析-堆排序》实验报告

学　号：1004191211　　姓　名：郎文鹏

日　期：2021/10/14　　得　分：

一、实验内容：

算法设计与分析课程实验内容——堆排序。

二、所用算法的基本思想及复杂度分析：

1. 基本思想

简单选择排序是循环在无序数据中选择一个新确定最大（小）的元素，共n次循环且每次循环需要进行$n-1$次比较操作。该算法没有将每次循环的比较结果保存下来导致在下一次循环中重复进行了很多次上次循环中做过的比较操作，因此比较次数比较多，当数据量增大的时候该算法所表现出来的效果较差。

堆排序是建立在简单选择排序思想上的一种高效的改进算法，该算法结合上述思想利用二叉堆式建树的数据结构尽可能减少每次操作所产生的重复与无用操作问题，利用大（小）顶堆的自身结构特点保证每次剩余的无用元素尽可能仍然保持相对大小关系。以堆调整操作为核心具体分为三个步骤：

1. 初始化堆：利用数组编号关系将给定原始序列通过堆调整操作转换为一个大（小）顶堆。
2. 选取堆顶：将堆顶（第一个）元素与堆尾（最后一个）元素进行交换确定一个新的最大（小）元素。
3. 堆调整：在每次确定一个新的元素后只需要通过少量的堆调整操作即可使大（小）顶堆重新恢复有序状态，有效避免了上述算法所带来过多比较操作的问题，并继续步骤 2 直至整个序列有序。

2. 复杂度分析

堆排序复杂度主要取决于上述涉及到堆调整操作的两个步骤：初始化堆和堆调整。

➢初始化堆时由于整个序列处于无序状态，凡是非叶子结点都需要进行一次

堆调整操作，所以从处于$\frac{n}{2}$位置及之前的结点按照从下至上从、从右至左的顺序进行堆调整。假设元素个数为n，则堆的高度$k = log(n+1) \approx log(n)$，非叶子结点个数为$2^{k-1}-1$,假设每个非叶子结点都需要调整则第$i$的非叶子结点需要操作次数为$k-i$，第$i$层的所有结点所做的操作为$k*2^{i-1}-i*2^{i-1}$次，共$k-1$层非叶子结点所以总的操作次数为$\sum_{i=1}^{k-1} k*2^{i-1}-i*2^{i-1} = 2^k-k+1$，将$k \approx log(n)$代入上式得$n-log\,n+1$，所以初始化堆复杂度为$O(n)$。

➢堆调整和初始化堆分析差不多，假设根结点和排在最后的序号为m的叶子结点交换并进行调整，那么堆调整为原来m结点所在的层数，即$logm$次共n个结点，所以总的操作次数为$\sum_{m=1}^{n-1} logm = log(n-1)! \approx nlogn$，所以堆调整复杂度为$O(nlogn)$。

根据分布加法原理可知整个堆排序算法总体复杂度为$O(nlogn)$。

三、源程序及注释：

➢生成随机数代码

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const ll a = 1, b = 10000;  //规定生成随机数的范围

signed main(int argc, char *argv[]) {
    freopen("IO\\in.txt", "w", stdout);
    stringstream ss;
    ll seed = time(NULL);
    if (argc > 1) {  //如果传入了参数
        ss.clear();
        ss << argv[1];
        ss >> seed;  //把参数转换成整数赋值给 seed
    }
    //rand_max=32767
    auto random = [] {  //加强随机数范围,生成数在[a,b]
        return a + rand() * rand() % (b - a + 1);
    };
    srand(seed);

    /**
     * @author pengpenglnag
     * 下面利用利用 rand()或者自定义的 random()生成随机数
```

```
     */

    int len = 10000;
    cout << len << endl;
    for (int i = 1; i <= len; ++i) {
        cout << left << setw(6) << random();
        if (i % 5 == 0) cout << endl;
    }
    cout << endl;
    return 0;
}
```

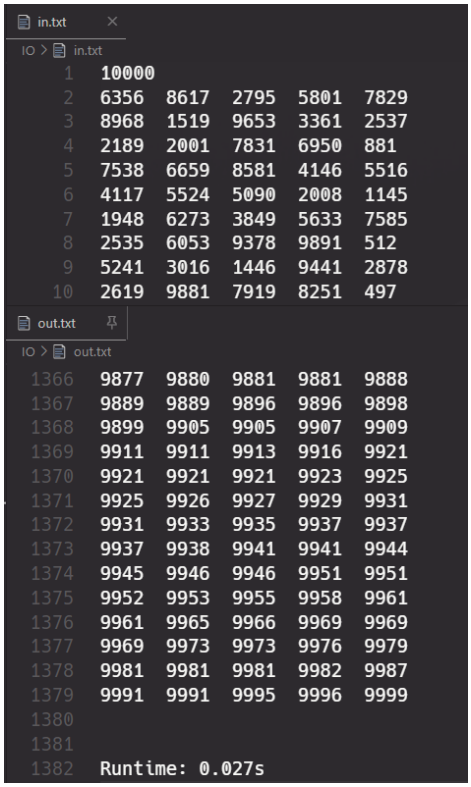
➢堆排序代码

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1e5 + 5;

int arr[maxn];

void HeapAdjust(int s, int m) {  //s 是堆的第一个元素，m 是堆的最后一个元素
    arr[0] = arr[s];             //将堆的第一个值赋给哨兵位置
    for (int j = 2 * s; j <= m; j *= 2) {
        if (j < m && arr[j] < arr[j + 1]) j++;  //找出孩子中最大的
        if (arr[0] >= arr[j]) break;            //符合大顶推退出调整
        arr[s] = arr[j];                        //把较大的字结点往上移动替换它的父结点
        s = j;                                  //更新当前要移动的点要放的位置
    }
    arr[s] = arr[0];  //交换
}

void solve() {
    int n;
    cin >> n;
    for (int i = 1; i <= n; ++i) cin >> arr[i];
    for (int i = n / 2; i > 0; --i)  //建堆从第 n/2 个位置依次往上调整
        HeapAdjust(i, n);
    for (int i = n; i > 1; --i) {  //交换堆顶和堆底，执行 n-1 次
        swap(arr[1], arr[i]);
        HeapAdjust(1, i - 1);  //堆调整从第一个位置调整
    }
    for (int i = 1; i <= n; ++i) {
        cout << left << setw(6) << arr[i];
```

```
        if (i % 5 == 0) cout << endl;
    }
    cout << endl;
}

signed main() {
    ios::sync_with_stdio(false), cin.tie(0);
    freopen("IO\\in.txt", "r", stdin);
    freopen("IO\\out.txt", "w", stdout);
    clock_t start, end;
    start = clock();
    solve();
    end = clock();
    cout << endl
         << "Runtime: " << (double)(end - start) / CLOCKS_PER_SEC <<
"s\n";  //看下时间
    return 0;
}
```

四、运行输出结果：

完整数据见*in.xt*与*out.txt*文件（本地或者链接均可）



五、调试和运行程序过程中产生的问题、采取的措施及获得的相关经验教训：

问题一：复杂度分析起初无法得到正确结果

解决措施：参考了网上代码对应的复杂度分析过程得知需要从非叶子结点与层数寻找关系，设置临时参数进行推导后再统一即可

获得教训：复杂度分析需要从已知变量出发，从变化中确定其他相关参数进行推导

问题二：大量数据的测试过程无法进行

解决措施：学习了文件读入读出与随机函数后，写一份随机代码生成数据并存入一个文件，跑堆排序代码从上述文件读入后将运行结果写入另一个文件

获得教训：无