算法设计与分析_第3次作业

堆

1. 将一个一维数组构造成一个堆
2. 插入一个元素，并调整堆

请代码实现上述堆的基本操作，并自己构造测试数据验证代码的正确性。

算法设计

（*写清思路*）

在一个主函数中有两个函数：一个是HeapSort 堆排序（构造堆）函数，一个是HeapAdjust堆筛选函数。

HeapSort中为堆排序，目的是修复堆结构；HeapAdjust中为堆筛选，目的是修复堆的偏序结构。

步骤一：构造初始堆，将给定无序序列构造成一个大顶堆，此时，整个数组最大值就是堆的顶端

步骤二：将顶端最大值输出并与末尾的数交换，此时，数组末尾的数是整个数组的最大值，剩余等待排序的数组元素个数为n-1

步骤三：将除了末尾之外的n-1个数再次构成大根堆，再将顶端数与n-1位置的数交换，此时，n-2是整个数组第二大的值，如此反复执行，可以得到一个有序数组

算法分析

（*分析算法时间复杂度*）

堆排序分为两步：初始化堆以及调整堆

1. 初始化堆

   设元素个数为n，则堆的高度$h = log(n+1) \approx logn$，非叶子结点的个数为$2^{h-1}-1$。

   假设每一个非叶子结点都需要调整，则第i层的非叶子结点需要的操作次数为$h-i$，

   第i层有2^{i-1}个节点，则第i层的所有结点需要的操作为$h \times 2^{i-1} - i \times 2^{i-1}$，共$k-1$层非叶子结点，总操作次数为$\sum_{i=1}^{h-1}(k \times 2^{i-1} - i \times 2^{i-1}) = 2^h - h + 1$

   将$h = log(n+1) \approx logn$带入，得$n - log(n) + 1$

   所以初始化堆的复杂度为$O(n)$

2. 调整堆

   假设根节点和最后一个序号为m的叶子结点交换，那么调整的操作次数=原来m结点所在层数=堆的高度=$log(m)$，共有n个结点，调整的总操作次数为$\sum_{m=1}^{n-1} logm$，化简可得$log(n-1)! \approx nlogn$

   所以调整堆的复杂度为$O(nlogn)$

3. 综上所述：总体时间复杂度为$O(nlogn)$

算法实现

(*用任意语言实现上述算法。*)

```c
#include "stdio.h"
// 小顶堆
void SmallHeapAdjust(int nums[], int i, int n){
    int root, child;
    root = nums[i];
    for(child = 2 * i; child <= n; child = child * 2){
        if(child < n && nums[child] < nums[child + 1]) child++;
        if(root > nums[child]) break;
        nums[i] = nums[child];
        i = child;
    }
    nums[i] = root;
}
// 大顶堆
void BigHeapAdjust(int nums[], int i, int n){
    int root, child;
    root = nums[i];
    for(child = 2 * i; child <= n; child = child * 2){
        if(child < n && nums[child] > nums[child + 1]) child++;
        if(root < nums[child]) break;
        nums[i] = nums[child];
        i = child;
    }
    nums[i] = root;
}
// 堆排序
void HeapSort(int nums[], int n){
    int t, i;
    for(i = n / 2; i > 0; i--) {
        SmallHeapAdjust(nums, i, n);
    }
    for (i = n; i > 0; i--) {
        t = nums[1];
        nums[1] = nums[i];
        nums[i] = t;
        SmallHeapAdjust(nums, 1, i-1);
    }
    for(i = 1; i <= n; i++){
        printf("%d ", nums[i]);
    }
}

int main() {
    int n, i, num, index;
    scanf("%d", &n);
    int nums[n];
    for (i = 1; i <= n; i++){
        scanf("%d", &nums[i]);
    }
    HeapSort(nums, n);
    scanf("%d", &index);
    // 将插入的元素放到数组最后一个
    nums[n + 1] = index;
```

```
    printf("\n");
    HeapSort(nums, n + 1);
    return  0;
}
```