数据结构与算法作业5

未来技术学院2022111568 郑乔译

1. 指出堆和二叉排序树的区别：

（1）定义和性质：

堆：堆是一种特殊的树形数据结构，通常是一个完全二叉树。堆满足堆性质，即父节点的值总是大于（最大堆）或等于（最小堆）其子节点的值。堆通常用于实现优先队列，其中最大堆用于实现最大优先队列，最小堆用于实现最小优先队列。

二叉排序树：二叉排序树（BST）是一个二叉树，其中每个节点包含一个键值，且每个节点的键值大于其左子树中任何节点的键值，并小于其右子树中任何节点的键值。BST允许快速插入、删除和搜索操作。

（2）平衡性：

堆：堆通常不是平衡的。它的主要目的是通过维护特定的顺序（最大或最小）来优化某些操作（如提取最大或最小元素）。

二叉排序树：BST可以是不平衡的，这会导致其操作效率降低。为了解决这个问题，人们提出了平衡二叉搜索树（如AVL树、红黑树等），这些树在插入或删除操作后通过旋转等操作来保持平衡。

（3）操作效率：

堆：堆的插入和删除操作的时间复杂度通常是O(log n)，其中n是堆中的元素数量。然而，由于堆不是为搜索操作而设计的，因此搜索操作的效率可能较低。

二叉排序树：BST的插入、删除和搜索操作的时间复杂度在最坏情况下是O(n)（当树不平衡时）。然而，在平均情况下，这些操作的时间复杂度是O(log n)。

（4）应用场景：

堆：堆通常用于实现优先队列，其中元素的优先级由其键值决定。例如，在操作系统中，任务调度器可以使用最小堆来按优先级顺序调度任务。此外，堆也可以用于实现堆排序算法。

二叉排序树：BST通常用于需要快速插入、删除和搜索操作的场景。例如，在数据库中，索引可能使用BST来加速查询操作。此外，BST也可以用于实现关联数组（也称为映射或哈希表）等数据结构。

1. 若只想得到一个序列中第 k (k ≥ 5) 个最小元素之前的部分排序序列，则最好采用什么排序方法?

采用堆排序最合适，建立初始堆（小根堆）所花时间不超过4n，每次选出一个最小元素所花时间为log2n，因此得到第k个最小元素之前的部分序列所花时间大约为4n+klog2n

1. 已知由n(n≥2)个正整数构成的集合A={ak}（0≤k<n），将其划分为两个不相交的子集A1和A2，元素个数分别是n1和n2，A1和A2中元素之和分别为S1和S2。设计一个尽可能高效的划分算法，满足|n1-n2|最小且|S1-S2|最大。要求:
2. 给出算法的基本设计思想；

这段代码将集合 A 中的元素按照大小排序，然后将前面n/2个元素放入子集 A1中，将后面n/2个元素放入子集A2中，（如果n为奇数，则将中间的值分配给A2）确保两个子集的大小尽可能相等。然后计算两个子集的和的差的绝对值。

1. 根据设计思想，采用 C/C++语言描述算法，关键之处给出注释；
2. #include <iostream>  
   #include <vector>  
   #include <algorithm>  
     
   using namespace std;  
     
   void quicksort(vector<int> &a,long long l, long long r)  
   {  
    long long i = l;  
    long long j = r;  
    long long mid = a[l + (r - l) / 2];  
    do  
    {  
    while (a[i] < mid) i++;  
    while (a[j] > mid) j--;  
    if (i <= j)  
    {  
    int temp;  
    temp = a[i];  
    a[i] = a[j];  
    a[j] = temp;  
    i++;  
    j--;  
    }  
    } while (i <= j);  
    if (i < r) quicksort(a,i, r);  
    if (j > l) quicksort(a,l, j);  
   }  
     
     
   int main() {  
    int n;  
    cout << "Enter the number of positive integers in set A (n >= 2): ";  
    cin >> n;  
     
    vector<int> A(n);  
    cout << "Enter " << n << " positive integers for set A: ";  
    for (int i = 0; i < n; ++i) {  
    cin >> A[i];  
    }  
     
    quicksort(A,0, A.size()-1);  
     
    int sum1 = 0, sum2 = 0;  
    vector<int> A1, A2;  
     
    for(int i = 0;i < n/2;i++)  
    {  
    A1.push\_back(A[i]);  
    sum1 += A[i];  
    }  
    for(int i = n/2;i < n;i++)  
    {  
    A2.push\_back(A[i]);  
    sum2 += A[i];  
    }  
    ///若为奇数，中间的给A2(大的数组)  
     
    int diff = abs(sum1 - sum2);  
     
    cout << "Elements in A1: ";  
    for (int num : A1) {  
    cout << num << " ";  
    }  
    cout << "Sum: " << sum1 << endl;  
     
    cout << "Elements in A2: ";  
    for (int num : A2) {  
    cout << num << " ";  
    }  
    cout << "Sum: " << sum2 << endl;  
     
    cout << "Absolute difference between sums of two subsets: " << diff << endl;  
     
    return 0;  
   }

3)说明你所设计算法的平均时间复杂度和空间复杂度。

时间复杂度为nlogn，因为采用了快排算法。

空间复杂度为n，因为使用了一个大小为 n 的 vector 存储输入的正整数集合 A，又使用了两个大小分别为 n/2 的 vector A1 和 A2，存储划分后的两个子集。