学习情况表

|  |  |  |  |
| --- | --- | --- | --- |
| **姓名** | 杜金瑞 | **学号** | 2020905073 |
| **学院** | 信息工程学院 | **专业** | 计算机科学与技术 |

|  |
| --- |
| 学习情况简述 |
| C:\Users\29043\Documents\Tencent Files\2904326062\Image\C2C\60685BB2C8B2326E4A6DFA067093F11F.jpg |
| 本周练习过的代码（例） |

public class BinarySearch {  
  
 private BinarySearch(){}  
  
 //非递归实现二分查找法  
 public static<E extends Comparable<E>> int search(E[] arr,E target){  
  
 int l=0,r=arr.length-1;  
  
 //在arr[l,r]中查找target  
 while(l<=r){  
  
 int mid=l+(r-l)/2;  
  
 if(arr[mid].compareTo(target)==0)  
 return mid;  
  
 if(arr[mid].compareTo(target)<0)  
 l=mid+1;  
 else r=mid-1;  
  
 }  
 return -1;  
 }  
  
 //递归实现二分查找法  
 public static<E extends Comparable<E>> int searchR(E[] arr,E target){  
 return *searchR*(arr,0, arr.length-1,target);  
 }  
  
 public static<E extends Comparable<E>> int searchR(E[] arr,int l,int r,E target){  
  
 if(l>r) return -1;  
  
 int mid=l+(r-l)/2;  
  
 if(arr[mid].compareTo(target)==0)  
 return mid;  
  
 if(arr[mid].compareTo(target)<0)  
 return *searchR*(arr, mid+1,r,target);  
  
 return *searchR*(arr,l,mid-1,target);  
 }  
  
  
}

import java.util.Arrays;  
  
public class eatingBanana {  
  
 public int minEatingSpeed(int[] piles, int h) {  
  
 int l=1,r= Arrays.*stream*(piles).max().getAsInt();  
  
 while(l<r){  
  
 int mid=l+(r-l)/2;  
  
 if(eatingTime(piles,mid)<=h)  
 r=mid;  
 else  
 l=mid+1;  
 }  
 return l;  
 }  
  
 public int eatingTime(int[] piles,int target){  
  
 int res=0;  
 for(int pile:piles){  
 res+=pile/target +(pile%target>0 ? 1:0);  
 }  
 return res;  
 }  
  
 public int shipWithinDays(int[] weights, int days) {  
  
 int l=Arrays.*stream*(weights).max().getAsInt(),r=Arrays.*stream*(weights).sum();  
  
 while(l<r){  
  
 int mid=l+(r-l)/2;  
  
 if(days(weights,mid)<=days)  
 r=mid;  
 else l=mid+1;  
 }  
 return l;  
 }  
  
 public int days(int[] weights, int k){  
  
 int res=0,cur=0;  
  
 for(int weight:weights)  
  
 if(cur+weight<=k){  
 cur+=weight;  
 }  
 else{  
 res++;  
 cur=weight;  
 }  
 res++;  
 return res;  
 }  
}

import java.util.Random;  
  
public class SelectK {  
  
 private SelectK(){}  
  
 public int findKthLargest(int[] nums, int k) {  
  
 Random rnd = new Random();  
 return selectK(nums, nums.length - k, rnd);  
 }  
  
  
 private int selectK(int[] arr, int k, Random rnd){  
  
 int l = 0,r=arr.length;  
  
 //在arr[l,r)范围中寻找第k小元素  
 while(l > r){  
  
 int p = partition(arr,l,r-1,rnd);  
  
 if(k==p)  
 return arr[p];  
  
 if(k>p)  
 l=p+1;  
 else  
 r=p;  
  
 }  
 throw new RuntimeException("No Solution");  
 }  
  
  
 private int partition(int[] arr, int l, int r, Random rnd){  
  
 // 生成 [l, r] 之间的随机索引  
 int p = l + rnd.nextInt(r - l + 1);  
 swap(arr, l, p);  
  
 // arr[l+1...i-1] <= v; arr[j+1...r] >= v  
 int i = l + 1, j = r;  
 while(true){  
  
 while(i <= j && arr[i] < arr[l])  
 i ++;  
  
 while(j >= i && arr[j] > arr[l])  
 j --;  
  
 if(i >= j) break;  
  
 swap(arr, i, j);  
  
 i ++;  
 j --;  
 }  
  
 swap(arr, l, j);  
 return j;  
 }  
  
 private void swap(int[] arr, int i, int j){  
  
 int t = arr[i];  
 arr[i] = arr[j];  
 arr[j] = t;  
 }  
}

class Solution {  
 public int search(int[] nums, int target) {  
  
 return search(nums,0,nums.length-1,target);  
 }  
  
 public int search(int[] arr,int l,int r,int target){  
  
 if(l>r) return -1;  
  
 int mid=l+(r-l)/2;  
  
 if(arr[mid]==(target))  
 return mid;  
  
 if(arr[mid]<(target))  
 return search(arr, mid+1,r,target);  
  
 return search(arr,l,mid-1,target);  
 }  
  
 // > target的最小值  
 public static<E extends Comparable<E>> int upper(E[] data,E target){  
  
 int l=0,r=data.length;  
 while(l<r){  
  
 int mid=l+(r-l)/2;  
  
 if(data[mid].compareTo(target)>0)  
 r=mid;  
 if(data[mid].compareTo(target)<=0)  
 l=mid+1;  
 }  
 return l;  
 }  
  
 // > target 返回最小值索引  
 // == target 返回最大值索引  
 public static <E extends Comparable <E>> int ceil(E[] data ,E target){  
  
 int p = *upper*(data,target); //大于target的最小索引  
 if(p-1>=0&&data[p-1].compareTo(target)==0)  
 return p-1;  
 return p;  
  
 }  
  
 // >= target 的最小索引  
 public static <E extends Comparable <E>> int lower\_ceil(E[] data ,E target){  
  
 int l=0,r=data.length;  
  
 while(l<r){  
  
 int mid=l+(r-l)/2;  
 if(data[mid].compareTo(target)<0)  
 l=mid+1;  
 else  
 r=mid;  
 }  
 return l;  
 }  
  
 // < target 的最大值索引  
 public static <E extends Comparable<E>> int lower(E[] arr,E target){  
  
 int l=-1,r=arr.length-1;  
  
 while(l<r){  
  
 int mid=l+(r-l+1 )/2;  
  
 if(arr[mid].compareTo(target)<0)  
 l=mid;  
 else  
 r=mid-1;  
 }  
 return l;  
 }  
  
 // < target ，返回最大值索引  
 // == target，返回最小索引  
 public static <E extends Comparable<E>> int lower\_floor(E[] data,E target){  
  
 int l=*lower*(data,target);  
  
 if(l + 1 < data.length && data[l + 1].compareTo(target) == 0)  
 return l + 1;  
 return l;  
  
 }  
  
 // <= target 最大索引  
 public static <E extends Comparable<E>> int upper\_floor(E[] data, E target){  
  
 int l = -1, r = data.length - 1;  
  
 while(l < r){  
  
 //上取整  
 int mid = l + (r - l + 1) / 2;  
  
 if(data[mid].compareTo(target) <= 0)  
 l = mid;  
 else  
 r = mid - 1;  
 }  
 return l;  
 }  
  
  
 public static <E extends Comparable<E>> int search2(E[] data, E target){  
  
 // >= target 的最小值索引  
 int l = 0, r = data.length;  
  
 // 在 data[l, r] 中寻找解  
 while(l < r){  
  
 int mid = l + (r - l) / 2;  
 if(data[mid].compareTo(target) < 0)  
 l = mid + 1;  
 else  
 r = mid;  
 }  
 if(l < data.length && data[l].compareTo(target) == 0)  
 return l;  
 return -1;  
 }  
  
 public static void main(String[] args) {  
 Integer[] arr={1,1,3,3,5,5};  
 for(int i=0;i<=6;i++)  
 System.*out*.print(Solution.*lower*(arr,i)+" ");  
  
 }  
}