**算法实训**

**实验报告**

**专业：数据科学与大数据技术2018班**

**名字： 陈炳宇**

**学号： 201806404139**

**实验一:循环和递归实现二分查找： （python）**

**循环：**

**#include <stdio.h>**

**#include <stdlib.h>**

**int BinaryFind(int arr[], int size, int to\_find)**

**{**

**int left = 0;**

**int right = size - 1;**

**while (left <= right)**

**{**

**int mid = (left + right) / 2;**

**if (to\_find < arr[mid])**

**{**

**right = mid - 1;**

**}**

**else if (to\_find > arr[mid])**

**{**

**left = mid + 1;**

**}**

**else**

**{**

**return mid;**

**}**

**}**

**return -1;//未找到返回 -1**

**}**

**int main()**

**{**

**int arr[] = { 1, 3, 4, 6, 7, 8, 10, 13, 14};**

**int to\_find = 4;**

**int size = sizeof(arr) / sizeof(arr[0]);**

**int i = BinaryFind(arr, size, to\_find);**

**//判定找到与否**

**if (i == -1)**

**{**

**printf("没找到！\n");**

**}**

**else**

**{**

**printf("找到了！下标为：%d\n", i);**

**}**

**system("pause");**

**return 0;**

**}**

**递归：**

**#include<stdio.h>**

**#define N 10 //定义元素个数；**

**int a[N]={1,2,3,4,5,6,7,8,9,10}; //定义查找数组；**

**int half(int low,int high,int key) //low表示起点 high表示终点；**

**{**

**if(low<high)**

**{**

**int mid=(low+high)/2; //设置中间值，用中间值与key比较；**

**if(a[mid]==key) //等于的话，说明找到了这个元素；**

**return mid;**

**else if(a[mid]>key) //要找的值在mid的左边；**

**return half(low,mid-1,key); //重新定义区间长度，调用half函数；**

**else //要找的值在mid的右边；**

**return half(mid+1,high,key); //重新定义区间长度，调用half函数；**

**}**

**return -1;**

**}**

**int main(void)**

**{**

**int key;**

**printf("输入要查找的关键字: ");**

**scanf("%d",&key);**

**int pos=half(0,N-1,key);**

**printf("原数据表: ");**

**for(int i=0;i<N;i++)**

**printf("%d ",a[i]);**

**printf("\n\n");**

**if(pos>=0)**

**printf("查找成功,该关键字位于数组的第%d个元素!\n",pos+1);**

**else**

**printf("查找失败!!!\n");**

**return 0;**

**}**

**实验二：递归调运实现快速排序和合并排序（Python，c语言）**

**快速排序**

**#include <stdio.h>**

**int partions(int l[],int low,int high)**

**{**

**int prvotkey=l[low];**

**l[0]=l[low];**

**while (low<high)**

**{**

**while (low<high&&l[high]>=prvotkey)**

**--high;**

**l[low]=l[high];**

**while (low<high&&l[low]<=prvotkey)**

**++low;**

**l[high]=l[low];**

**}**

**l[low]=l[0];**

**return low;**

**}**

**void qsort(int l[],int low,int high)**

**{**

**int prvotloc;**

**if(low<high)**

**{**

**prvotloc=partions(l,low,high); //将第一次排序的结果作为枢轴**

**qsort(l,low,prvotloc-1); //递归调用排序 由low 到prvotloc-1**

**qsort(l,prvotloc+1,high); //递归调用排序 由 prvotloc+1到 high**

**}**

**}**

**void quicksort(int l[],int n)**

**{**

**qsort(l,1,n); //第一个作为枢轴 ，从第一个排到第n个**

**}**

**int main()**

**{**

**int a[11]={0,2,32,43,23,45,36,57,14,27,39};**

**for (int b=1;b<11;b++)**

**printf("%3d",a[b]);**

**printf("\n");**

**quicksort(a,11);**

**for(int c=1;c<11;c++)**

**printf("%3d",a[c]);**

**}**

**合并排序**

**#include<stdio.h>**

#include <iostream>

using namespace std;

void Merge(int A[], int TmpA[], int L, int R, int RightEnd)

{

int LeftEnd, ElementNum, Tmp;

LeftEnd = R - 1;

Tmp = L;

ElementNum = RightEnd - L + 1;

while (L <= LeftEnd && R <= RightEnd)

{

if (A[L] <= A[R])

TmpA[Tmp++] = A[L++];

else

TmpA[Tmp++] = A[R++];

}

while (L <= LeftEnd)

TmpA[Tmp++] = A[L++];

while (R <= RightEnd)

TmpA[Tmp++] = A[R++];

for (int i = 0; i < ElementNum; i++, RightEnd--)

A[RightEnd] = TmpA[RightEnd];

}

void Msort(int A[], int TmpA[], int L, int RightEnd)

{

int Center;

if (L < RightEnd)

Center = (L + RightEnd) / 2;

Msort(A, TmpA, L, Center);

Msort(A, TmpA, Center + 1, RightEnd);

Merge(A, TmpA, L, Center + 1, RightEnd);

}

}

void MergeSort(int A[], int N)

{

int \*TmpA;

TmpA = new int[N];

if (TmpA)

{

Msort(A, TmpA, 0, N - 1);

delete[] TmpA;

}

else

cout << "空间不足" << endl;

}

int main()

{

int N, i;

printf("请输入要排列的个数：");

cin >> N;

int A[N];

printf("请输入%d个整数排列: ", N);

for(i = 0; i < N; i++)

{

cin >> A[i];

}

printf("它们从小到大的顺序为: ");

MergeSort(A,N);

for(i=0;i < N; i++)

{

cout << A[i] << " ";

}

cout << endl;

return 0;

}

**实验三：给n个数中找出最小k个数（C语言）**

**import java.util.ArrayList;**

**import java.util.PriorityQueue;**

**import java.util.Comparator;**

**public class Solution {**

**public ArrayList<Integer> GetLeastNumbers\_Solution(int[] input, int k) {**

**ArrayList<Integer> result = new ArrayList<Integer>();**

**int length = input.length;**

**if(k > length || k == 0){**

**return result;**

**}**

**PriorityQueue<Integer> maxHeap = new PriorityQueue<Integer>(k, new Comparator<Integer>() {**

**@Override**

**public int compare(Integer o1, Integer o2) {**

**return o2.compareTo(o1);**

**}**

**});**

**for (int i = 0; i < length; i++) {**

**if (maxHeap.size() != k) {**

**maxHeap.offer(input[i]);**

**} else if (maxHeap.peek() > input[i]) {**

**Integer temp = maxHeap.poll();**

**temp = null;**

**maxHeap.offer(input[i]);**

**}**

**}**

**for (Integer integer : maxHeap) {**

**result.add(integer);**

**}**

**return result;**

**}**

**}**

**实验四：循环和递归实现01背包问题动态规划**

**循环**

**#include<iostream>**

**using namespace std;**

**#include <algorithm>**

**int main()**

**{**

**int w[5] = { 0 , 2 , 3 , 4 , 5 }; //商品的体积2、3、4、5**

**int v[5] = { 0 , 3 , 4 , 5 , 6 }; //商品的价值3、4、5、6**

**int bagV = 8; //背包大小**

**int dp[5][9] = { { 0 } }; //动态规划表**

**for (int i = 1; i <= 4; i++) {**

**for (int j = 1; j <= bagV; j++) {**

**if (j < w[i])**

**dp[i][j] = dp[i - 1][j];**

**else**

**dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - w[i]] + v[i]);**

**}**

**}**

**//动态规划表的输出**

**for (int i = 0; i < 5; i++) {**

**for (int j = 0; j < 9; j++) {**

**cout << dp[i][j] << ' ';**

**}**

**cout << endl;**

**}**

**return 0;**

**}**

**递归**

**#include <stdio.h>**

**#include <malloc.h>**

**#define M 5**

**int \*choice,\*saved;**

**int maxV = -1;**

**void KNAPSACK(int \*v,int \*s,int C,int cur);**

**int main(void)**

**{**

**int s[M] = {3,5,7,8,9},v[M] = {4,6,7,9,10},C = 11;**

**choice = (int\*)malloc(sizeof(int)\*M);**

**saved = (int\*)malloc(sizeof(int)\*M);**

**int i=0,sum=0;**

**KNAPSACK(v,s,C,0);**

**printf("选择");**

**for(i=0;i<M;i++)**

**{**

**if(saved[i])**

**{**

**printf("第%d项,",i+1);**

**sum+=s[i];**

**}**

**}**

**printf("物品放入背包,");**

**printf("物品总体积为:%d,总价值为%d.",sum,maxV);**

**return 0;**

**}**

**//v数组价值，s数组体积，C背包容量，cur当前位**

**void KNAPSACK(int \*v,int \*s,int C,int cur)**

**{**

**if(cur==M-1)**

**{**

**int value=0,bag=0,i=0;**

**for(i=0;i<M;i++)**

**{**

**if(choice[i])**

**{**

**bag+=s[i];**

**value+=v[i];**

**}**

**}**

**if(maxV<value&&bag<=C)**

**{**

**maxV = value;**

**for(i=0;i<M;i++) saved[i] = choice[i];**

**}**

**return;**

**}**

**else**

**{**

**choice[cur] = 0;**

**KNAPSACK(v,s,C,cur+1);**

**choice[cur] = 1;**

**KNAPSACK(v,s,C,cur+1);**

**}**

**}**

**实验五：循环和递实现公共最长子序列动态规划**

**递归**

**def lcs1(A, B):**

**# 普通递归**

**if not(A and B):**

**return ''**

**elif A[-1] == B[-1]:**

**return lcs1(A[:-1], B[:-1]) + A[-1]**

**else:**

**if len(lcs1(A[:-1], B)) > len(lcs1(A, B[:-1])):**

**return lcs1(A[:-1], B)**

**else:**

**return lcs1(A, B[:-1])**

**循环**

**def findlongestsubstring(source, dest): # 输入值， 要比较的值**

**inLen = len(source)**

**outLen = len(dest)**

**target = []**

**cell = [ [0 for j in range(inLen+1)] for i in range(outLen+1)]**

**for i in range(1, outLen+1):**

**for j in range(1, inLen+1):**

**if dest[i-1] == source[j-1]:**

**cell[i][j] = cell[i-1][j-1] + 1**

**target.append(dest[i-1])**

**else:**

**cell[i][j] = max(cell[i-1][j], cell[i][j-1])**

**for i in range(1, outLen+1):**

**for j in range(1, inLen+1):**

**print(cell[i][j], end=', ')**

**print()**

**print("The Longest sub sequence of \"%s\" and \"%s\" is %s" % (source, dest, target))**

**if \_\_name\_\_ == "\_\_main\_\_":**

**findlongestsubstring("fish", "fosh")**

**实验六：背包问题贪心算法（python）**

**#include<stdio.h>**

**void Sort(int n, float v[],float w[])**

**{ float t[100];**

**float a,b,c;**

**for(int i=1;i<=n;i++)**

**t[i]=v[i]/w[i];**

**for(int k=1;k<=n;k++)**

**for(int j=k+1;j<=n;j++)**

**{**

**if(t[k]<t[j])**

**{**

**a=t[k];**

**t[k]=t[j];**

**t[j]=a;**

**b=w[k];**

**w[k]=w[j];**

**w[j]=b;**

**c=v[k];**

**v[k]=v[j];**

**v[j]=c;**

**}**

**}**

**}**

**void Knapsack(int n, float M, float v[],float w[],float x[])**

**{**

**Sort(n,v,w);**

**int i;**

**for (i=1;i<=n;i++) x[i]=0;**

**float c=M;**

**for (i=1;i<=n;i++) {**

**if (w[i]>c) break;**

**x[i]=1;**

**c-=w[i];**

**}**

**if (i<=n) x[i]=c/w[i];//最后一个物品运行部分转入**

**}**

**void main()**

**{ int n,i;**

**float m;**

**float v[100],w[100],x[100];**

**printf("输入物品数目和背包容量：");**

**scanf("%d %f",&n,&m);**

**printf("输入物品价值:\n");**

**for(i=1;i<=n;i++)**

**{**

**scanf("%f",&v[i]);**

**}**

**printf("输入物品重量:\n");**

**for(i=1;i<=n;i++)**

**{**

**scanf("%f",&w[i]);**

**}**

**Knapsack(n,m,v,w,x);**

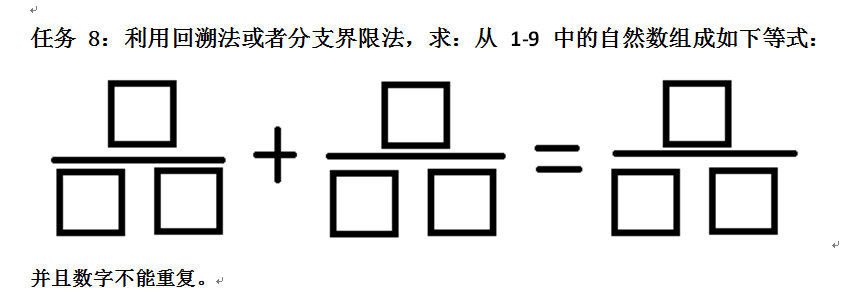
**for(i=1;i<=n;i++)**

**printf("价值%f重%f的物品占比：%f\n",v[i],w[i],x[i]);**

**}**

**实验七：回溯或分支界限实现1-9其中六个自然数组成一个三角形，其各边和等**

**实验八：回溯或分支界限实现1-9填入合理并数字不重复**



w