**#include <iostream>**

**#include <vector>**

**#include <cassert>**

**using namespace std;**

**//冒泡排序**

**int swap(int \*a,int \*b){**

**int t=\*a;**

**\*a=\*b;**

**\*b=t;**

**}**

**void bubble\_sort(int a[], int n){**

**int i,j;**

**for(i=0;i<n-1;++i)**

**for(j=0;j<n-i-1;++j)**

**{**

***if(a[j]>a[j+1]) //小心J+1 越界***

**swap(&a[j],&a[j+1]);**

**}**

**}**

**/\***

**选择排序的原理：每次在无序队列中“选择”出最小值，放到有序队列的最后，并从无序队列**

**中去除该值（具体实现略有区别）。**

**选择排序的代码如下：**

**\*/**

**void selection\_sort(int a[],int n){**

**int i,j;**

**int min;**

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

**{**

**min=i;**

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

**{**

**if(a[j]<a[min])**

**min=j;**

**}**

**if(min!=i)**

**swap(&a[min],&a[i]);**

**}**

**}**

**/\***

**插入排序的原理：始终定义第一个元素为有序的，将元素逐个插入到有序排列之中，其特点是要不断的**

**移动数据，空出一个适当的位置，把待插入的元素放到里面去。**

**插入排序的代码如下：**

**\*/**

**void myself\_insert(int \* arr, int len){**

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

**int curr = arr[i];**

**int j = i - 1; //跳到负数去了 when i = 0,既第一个数，直接放置**

**while(arr[j]>curr &&j>=0){**

**arr[j+1] = arr[j];**

**j--;**

**}**

**arr[j+1] = curr;**

**}**

**}**

**//快速排序：时间复杂度o(nlgn) //快速排序法**

**using namespace std;**

**int partition\_array(int \* arr,int left,int right){**

***int pivot = arr[left]; //中轴***

**while(left < right){**

**while(left<right && arr[right] >= pivot){**

**--right;**

**}**

**arr[left] = arr[right];//直到找到小的就交换**

**while(left <right && arr[left] <= pivot){**

**left++;**

**}**

**arr[right] = arr[left];**

**}**

**arr[left] = pivot;**

**return left;**

**}**

**void quick\_sort(int\* arr, int left,int right){**

**if(left < right){**

**int q = partition\_array(arr,left,right);**

**quick\_sort(arr,left,q-1);**

**quick\_sort(arr,q+1,right);**

**}**

**}**

**void myself\_quicksort(int \* arr, int len){**

**quick\_sort(arr,0,len-1);**

**}**

**//归并排序：时间复杂度o(nlgn)**

**int merge\_data(int \* arr,int left,int mid,int right){**

**int length = right-left+1;**

**int \* arr\_cpy = new int[length]; //copy array**

**for(int i = 0, j = left;i < length;i++,j++)**

**arr\_cpy[i] = arr[j];**

**int i = 0;**

**int j = mid-left+1; //分成两块**

**int k = left;**

**for(;i <= mid-left && j<=length -1;k++){**

**if(arr\_cpy[i] < arr\_cpy[j]) //合并，小在前，在后**

**arr[k] = arr\_cpy[i++]; //假设只有两个数字，可以排好了**

**Else**

**arr[k] = arr\_cpy[j++];//假设有四个数字，那么两边的的已经有顺序了**

**}**

**while(i <= mid-left)**

**arr[k++] = arr\_cpy[i++];**

**while(j<=length-1)**

**arr[k++] = arr\_cpy[j++];**

**delete arr\_cpy;**

**}**

**void merge\_sort(int\* arr, int left,int right){**

**if(left<right){**

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

**merge\_sort(arr,left,mid);**

**merge\_sort(arr,mid+1,right); // 先分开**

**merge\_data(arr, left,mid,right); //在合并**

**}**

**}**

**void myself\_gubing(int \* arr, int len){**

**merge\_sort(arr,0,len-1);**

**}**

**时间复杂度**

**1.最坏情况查找最后一个元素（或者第一个元素）Master定理T(n)=T(n/2)+O(1)所以T(n)=O(log2n)**

**2.最好情况查找中间元素O(1)查找的元素即为中间元素（奇数长度数列的正中间，偶数长度数列的中间靠左的元素）**

**#include<iostream>**

**#define N 10**

**using namespace std;**

**int main()**

**{**

**int a[N],front,end,mid,x,i;**

**cout<<"请输入已排好序的a数组元素:"<<endl;**

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

**cin>>a[i];**

**cout<<"请输入待查找的数x:"<<endl;**

**cin>>x;**

**front=0;**

**end=N-1;**

**mid=(front+end)/2;**

**while(front<end&&a[mid]!=x)**

**{**

**if(a[mid]<x)front=mid+1;**

**if(a[mid]>x)end=mid-1;**

**mid=(front+end)/2;**

**}**

**if(a[mid]!=x)**

**printf("没找到！\n");**

**else**

**printf("找到了，在第%d项里",mid+1);**

**return 0;**

**}**

**[编程题]最后一位**

**#include <iostream>**

**using namespace std;**

**long long num\_sum(long long x){**

**long long sum = 0;**

**long long i = 1;**

**while(x/i > 0){**

**sum += x/i;**

**i \*= 10;**

**}**

**return sum;**

**}**

**int main()**

**{**

**long long num;**

**while(cin>>num){**

**long long nfront=0;**

**long long nend = num;**

**long long nmid =0;**

**while(nfront < nend){**

**nmid=nfront + (nend - nfront)/2;**

**if(num\_sum(nmid)< num)nfront = nmid + 1;**

**else if(num\_sum(nmid)> num)nend = nmid;**

**else break;**

**}**

**if(num\_sum(nmid) != num)**

**nmid = -1;**

**cout<<nmid<<endl;**

**}**

**return 0;**

**}**

**动态规划之 矩阵的最小路径和**

**int min\_path(int matrix[100][100], int M, int N){**

**int minvalue = 0;**

**/\***

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

**for(int j = 0;j< M;j++)**

**cout<<matrix[i][j]<<endl;**

**cout<<endl;**

**}**

**\*/**

**int dp[100][100];**

**dp[0][0] = matrix[0][0];**

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

**dp[0][i] = dp[0][i-1] + matrix[0][i];**

**for(int j = 1;j<N;j++)**

**dp[j][0] = dp[j-1][0] + matrix[j][0];**

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

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

**dp[j][i] = min(dp[j-1][i],dp[j][i-1]) + matrix[j][i]; // key point**

**}**

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

**for(int j = 0;j< M;j++)**

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

**cout<<endl;**

**}**

**return dp[N-1][M-1];**

**}**

**回溯算法记忆：**

**void backtracking(const vector<vector<char>>& table, vector<string>& res, string& local, int index, const string& digits)** {

if (index == digits.size())

res.push\_back(local);

else

for (int i = 0; i<table[digits[index] - '0'].size(); i++) {

local.push\_back(table[digits[index] - '0'][i]);

backtracking(table, res, local, index + 1, digits);

local.pop\_back();

}

}

vector<string> letterCombinations(string digits)

{

vector<string> res;

if (digits.size() == 0) return res;

string local;

vector<vector<char>> table(2, vector<char>());

table.push\_back(vector<char>{'a', 'b', 'c'}); // index 2

table.push\_back(vector<char>{'d', 'e', 'f'}); // 3

table.push\_back(vector<char>{'g', 'h', 'i'});

table.push\_back(vector<char>{'j', 'k', 'l'}); // 5

table.push\_back(vector<char>{'m', 'n', 'o'});

table.push\_back(vector<char>{'p', 'q', 'r', 's'}); // 7

table.push\_back(vector<char>{'t', 'u', 'v'});

table.push\_back(vector<char>{'w', 'x', 'y', 'z'}); // 9

backtracking(table, res, local, 0, digits);

return res;

}