顺序查找

public static int SequentialSearch(int[] a, int x) {

int i;

for (i = 0; i < a.length && a[i] != x; i++)

;

if (i == a.length)

return -1;

else

return i;

}

二分查找

public static int binarySearch(Comparable[] a, Comparable x) {

int low = 0, high = a.length - 1;

while (low <= high) {

int mid = (low + high) / 2;

if (a[mid].compareTo(x) < 0)

low = mid + 1;

else if (a[mid].compareTo(x) > 0)

high = mid - 1;

else

return mid;

}

return -1;

}

插入排序

template<class Type> void InsertionSort(datalist<Type> & list){

for (int i=1; i<list.CurrentSize; i++)

Insert(list, i);

}

template<class Type> void Insert(datalist<Type> & list, int i){

Element<Type> temp=list.vector[i]; int j=i ;

while(j>0&& temp.getkey( )<list.vector[j-1].getkey( )){

list.Vector[j]=list.Vector[j-1]; j--

}

list.Vector[j]=temp;

}

public static void insertionSort( Comparable [ ] a ){

int j;

for ( int p = 1; p < a.length; p++ ){

Comparable tmp = a[ p ];

for ( j = p; j > 0 && tmp.compareTo( a[ j – 1 ] ) < 0; j-- )

a[ j ] = a[ j - 1 ];

a[ j ] = tmp;

}

}

折半插入排序（二分法插入排序）

选择排序

public static void SelectionSort(int[] a, int n) {

// sort the n number in a[0:n-1].

for (int size = n; size > 1; size--) {

int j = Max(a, size);

swap(a[j], a[size - 1]);

}

}

冒泡排序

public static void BubbleSort(int[] a, int n) {

// Sort a[0:n-1] using a bubble sort

for (int i = n; i > 1; i--)

Bubble(a, i);

}

public static void Bubble(int[] a, int n) {

// Bubble largest element in a[0:n-1] to right

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

if (a[i] > a[i + 1])

swap(a[i], a[i + 1]);

}

}

秩排序

public static void Rank(int[] a, int n, int[] r) {

// Rank the n elements a[0:n-1]

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

r[i] = 0;

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

for (int j = 0; j < i; j++) {

if (a[j] <= a[i])

r[i]++;

else

r[j]++;

}

}

}

public static void Rearrange(int[] a, int n, int[] r) {

// In-place rearrangement into sorted order

for (int i = 0; i < n; i++) {

while (r[i] != i) {

int t = r[i];

swap(a[i], a[t]);

swap(r[i], r[t]);

}

}

}

最大子序列和

Algorithm 1:

public static int maxSubSum1(int[] a) {

int maxSum = 0;

for (int i = 0; i < a.length; i++)

for (int j = i; j < a.length; j++) {

int thisSum = 0;

for (int k = i; k <= j; k++)

thisSum += a[k];

if (thisSum > maxSum)

maxSum = thisSum;

}

return maxSum;

}

O(N 3 )

Algorithm 2:

public static int maxSubSum2(int[] a) {

int maxSum = 0;

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

int thisSum = 0;

for (int j = i; j < a.length; j++) {

thisSum += a[j];

if (thisSum > maxSum)

maxSum = thisSum;

}

}

return maxSum;

}

O( N 2 )

Algorithm 3:

private static int maxSumRec(int[] a, int left, int right) {

if (left == right)

if (a[left] > 0)

return a[left];

else

return 0;

int center = (left + right) / 2;

int maxLeftSum = maxSumRec(a, left, center);

int maxRightSum = maxSumRec(a, center + 1, right);

int maxLeftBorderSum = 0, leftBorderSum = 0;

for (int i = center; i >= left; i--) {

leftBorderSum += a[i];

if (leftBorderSum > maxLeftBorderSum)

maxLeftBorderSum = leftBorderSum;

}

int maxRightBorderSum = 0, rightBorderSum = 0;

for (int i = center + 1; i <= right; i++) {

rightBorderSum += a[i];

if (rightBorderSum > maxRightBorderSum)

maxRightBorderSum = rightBorderSum;

}

return max3(maxLeftSum, maxRightSum, maxLeftBorderSum

+ maxRightBorderSum);

}

public static int maxSubSum3(int[] a) {

return maxSumRec(a, 0, a.length - 1);

}

O(N logN)

找一个序列中第k小的元素

int selectkth(int a[], int k, int n) {

int i, j, mini, temp;

for (i = 0; i < k; i++) {

mini = i;

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

if (a[j] < a[mini])

mini = j;

tmp = a[i];

a[i] = a[mini];

a[mini] = tmp;

}

return a[k - 1];

}

约瑟夫环问题

rear: 每次指向要出队列的前一个结点

出队列的人也用链表来表示：

head: 指向出队列结点链表的开头结点

p: 指向出队列结点链表的尾结点

以上rear, head, p都是ListNode的一个对象引用。

1. w = m;

2. for( int i = 1; i<= n-1; i++){

1) for (int j =1; j<=w-1; j++)

rear = rear.link;

2) if (i = = 1){

head = rear.link ; p = head; }

else{

p.link = rear.link;

p = rear.link;

}

3) rear.link = p.link;

}

3. P.link = rear;

rear.link = null;

void YANGHUI(int n){

Queue<int> q; q.makeEmpty( );

q.Enqueue(1);

q.Enqueue(1);

int s=0;

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

cout << endl;

for (int k=1;k<=10-i;k++)

cout<<„ „;

q.Enqueue(0);

for (int j=1;j<=i+2;j++){

int t=q.Dequeue( );

q.Enqueue(s+t);

s=t;

if (j!=i+2)

cout<< s <<„ „;

}

}

}

Print the coefficients of the binomial expansion

用可变长度的二维数组来实现：

public class Yanghui

{ public static void main(String args[ ] )

{ int n = 10;

int mat[ ][ ] = new int [n ][ ]; // 申请第一维的存储空间

int i, j;

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

{ mat[i] = new int [i+1]; // 申请第二维的存储空间 ， 每次长度不同

mat[i][0] = 1; mat[i][i] = 1;

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

mat[i][j] = mat[i-1][j-1] + mat[i-1][j];

}

for ( i = 0; i< mat.length; i++)

{ for ( j = 0; j < n-i; j++) System.out.print(“ “);

for ( j = 0; j < mat[i].length; j++)

System.out.print(“ “ + mat[i][j]);

System.out.println( );

}

}}