1.反转链表：

# include <iostream>

using namespace std;

struct ListNode {

int val;

struct ListNode \*next;

ListNode(int x) :

val(x), next(NULL) {}

};

class Solution {

public:

ListNode\* ReverList(ListNode\* pHead)

{

//反转指针

ListNode \*pNode = pHead; //当前结点

ListNode \*pPrev = nullptr; //上一结点

ListNode \*pNext = nullptr; //下一结点

ListNode \*pReverseHead = nullptr;

//反转链表

while(pNode != nullptr)

{

pNext = pNode->next; ///建立链表

if (pNext == NULL) //判断pNode是否是最后一个结点

pReverseHead = pNode;

pNode->next = pPrev; //指针翻转

pPrev = pNode;

pNode = pNext;

}

return pReverseHead;

}

};

int main()

{

Solution a;

ListNode a1(1);

ListNode a2(2);

ListNode a3(3);

a1.next = &a2;

a2.next = &a3;

ListNode \*pNode = &a1;

while (pNode!= NULL)

{

cout << pNode->val << endl;

pNode = pNode->next;

}

ListNode \*pReverseHead = a.ReverList(&a1);

while (pReverseHead != NULL)

{

cout << pReverseHead->val << endl;

pReverseHead = pReverseHead->next;

}

getchar();

}

2.冒泡排序

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

//冒泡排序和快排

void bubble\_sort(vector<int> &a)

{

int count = a.size();

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

{

for (int j = 0; j< count - i - 1; j++)

{

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

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

}

}

}

int main()

{

//初始化

int n;

int m;

//输入数据

while (cin >> n)

{

vector<int> data;

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

{

cin >> m;

data.push\_back(m);

}

int count = data.size();

//排序

bubble\_sort(data);

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

{

cout << data[i] << ' ';

}

cout << endl;

}

}

3.反向输出链表

//用栈实现反向输出链表

void PrintListReversingly\_Iteratively(ListNode\* pHead)

{

//反转指针

stack<ListNode\*> nodes;

ListNode\* pNode = pHead;

while(pNode!= nullptr)

{

nodes.push(pNode);

pNode = pNode->next;

}

while (!nodes.empty())

{

pNode = nodes.top();

cout << pNode->val << endl;

nodes.pop();

}

}

4.快速排序

//划分，使j左边的数都小于j,j右边的数都大于j

int partiton(int arr[], int left, int right)

{

int i = left + 1;

int j = right;

int temp = arr[left];

while (i <= j)

{

while (arr[i] < temp)

{

i++;

}

while (arr[j] > temp)

{

j--;

}

if (i < j)

swap(arr[i++], arr[j--]);

else i++;

}

swap(arr[j], arr[left]);

return j;

}

void quick\_sort(int arr[], int left, int right)

{

if (left > right)

return;

int j = partiton(arr, left, right);

quick\_sort(arr, left, j - 1);

quick\_sort(arr, j + 1, right);

}

5.二分查找

//递归实现代码

int half\_search(int a[], int left, int right, int target)

{

int mid = (left + right) / 2;

int midValue = a[mid];

if (left < right)

{

if (midValue > target)

return half\_search(a, left, mid - 1, target);

else if (midValue < target)

return half\_search(a, mid + 1, right,target);

else {

return mid;

}

}

return -1;

}

//非递归实现代码

int half\_search1(int a[], int target)

{

int i = 0;

int j = sizeof(a)/sizeof(a[0]);

while (i <= j)

{

int mid = (i + j) / 2;

int midValue = a[mid];

if (midValue < target)

j = mid + 1;

else if (midValue > target)

j = mid - 1;

else

return mid;

}

return -1;

}