 **第一章 习 题**

**1.简要回答术语：数据，数据元素，数据结构，数据类型。**

数据：指能被计算机识别、存储和加工处理的信息载体。

数据元素：就是数据的基本单位，在某些情况下，市局元素也称为元素、结点、顶点、记录。数据元素有时可以由若干数据项组成。

数据类型：是一个值的集合以及在这些值上定义的一组操作的总称。通常数据类型可以看作是程序设计语言中已实现的数据结构。

数据结构：指的是数据之间的相互关系，即数据的组织形式。一般包括三个方面的内容：数据的逻辑结构、存储结构和数据的运算。

**。**

逻辑结构：实体数据元素间逻辑关系即实体性质理解基础进行抽象模型。  
物理结构：数据元素计算机存储即计算机数据理解逻辑结构计算机语言映射。

**3.算法分析的目的是什么？如何评价一个算法？**

算法分析的目的是分析算法的效率以求改进。

算法的正确性，易读性，健壮性，时空效率（运行）。

1. **解释程序设计中的数据类型和数据结构中的抽象数据类型的概念以及相互关系。**

在高级语言中，为了有效地组织数据，规范数据的使用，提高程序的可读性，方便用户使用，引入了整型，实型等基本数据类型。不同的高级语言会定义不同的基本数据类型。是一个值的集合以及在这些值上定义的一组操作的总称。通常数据类型可以看作是程序设计语言中已实现的数据结构。

所谓抽象数据类型(Abstract Data Type，ADT)是指这样一种数据类型，它不再单纯是一组值的集合，还包括作用在值集上的操作的集合，即在构造数据类型的基础上增加了对数据的操作，且类型的表示细节及操作的实现细节对外是不可见得。

1. **分析以下程序段的时间复杂度。**

# ⑴ Sum1( int n )

**{ int p=1, sum=0, m ;**

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

**{ p\*=m ; sum+=p ; }**

**return (sum) ;**

**}**

⑵ Sum2( int n )

{ int sum=0, m, t ;

for (m=1; m<=n; m++)

{ p=1 ;

for (t=1; t<=m; t++) p\*=t ;

sum+=p ;

}

return (sum) ;

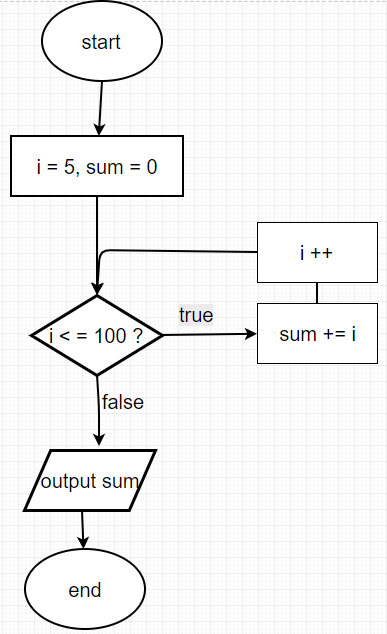
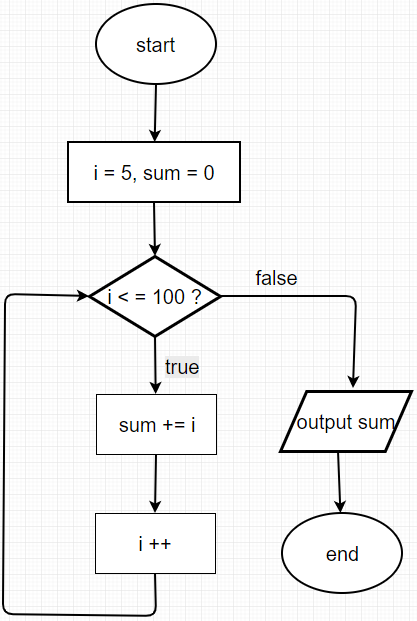
}

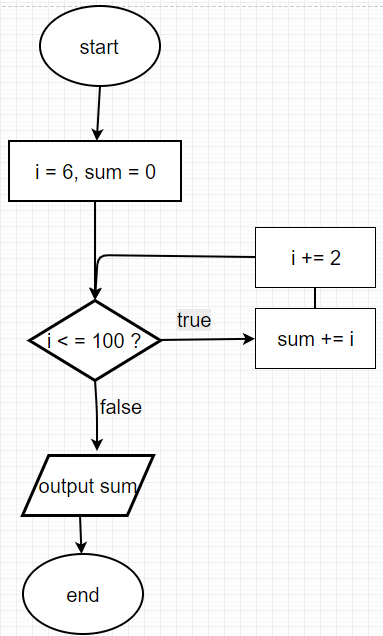
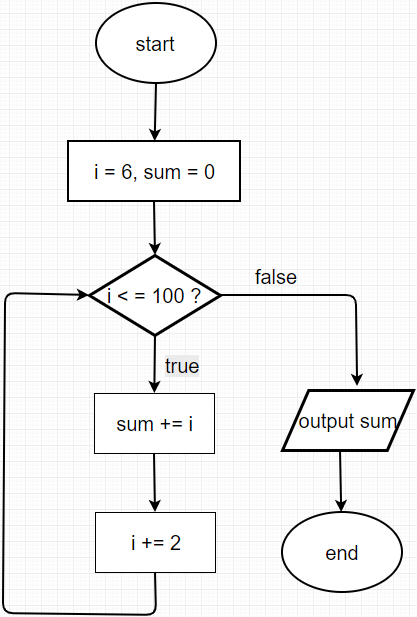
O(n) O(n²)

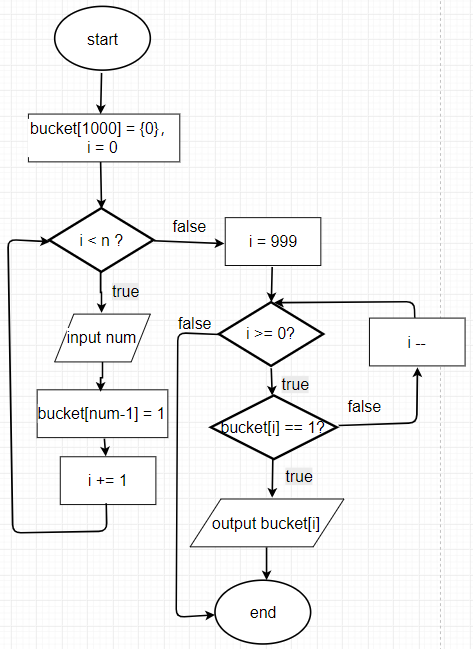
1. **绘制流程图部分。**

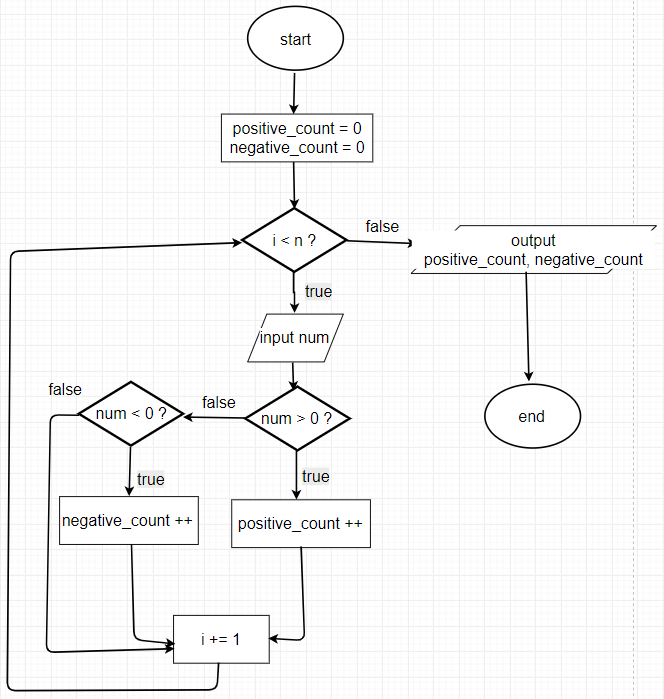
* **分别用while 和for 两种循环，求5~100自然数累加和的算法。**
* **分别用while 和for 两种循环，求5~100自然数中所有偶数的累加和。**
* **已知一组（n个）自然数的范围是1~1000， 求出其中最大值和最小值。**
* **已知一组（n个）自然数的范围是-1000~1000， 统计其中正数和负数**

的数目。









1. **指针的练习题目（自行在计算机上进行编程练习）。**

① 建立单链表

② 单链表上按位置查找

③ 单链表上按值查找

④ 删除单链表中的元素

⑤ 两个递增有序的单链表合并

#include<iostream>

using namespace std;

typedef struct node {

int value;

struct node\* next;

}Node;

Node\* create\_single\_linked\_list() {

//输入格式：1 2 3\n

Node\* head = new Node;

cin >> head->value;

Node\* pointer = head;

char character;

while ((character = getchar()) != '\n') {

Node\* new\_node = new Node;

cin >> new\_node->value;

pointer->next = new\_node;

pointer = new\_node;

}

pointer->next = NULL;

return head;

}

Node\* search\_by\_position(Node\*head, int index) {

int i;

Node\*pointer = head;

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

if (pointer->next == NULL) {

cerr << "search\_by\_position: index out of range";

return NULL;

}

pointer = pointer->next;

}

return pointer;

}

Node\* search\_by\_value(Node\* head, int value) {

//找到值等于value的第一个节点

Node\* pointer = head;

while (pointer != NULL) {

if (pointer->value == value) {

return pointer;

}

pointer = pointer->next;

}

return NULL;

}

void remove\_node(Node\* head, int index) {

if (index == 0) {

delete head;

return;

}

else if (index < 0) {

cerr << "remove\_node: index out of range";

return;

}

Node\* last\_node = search\_by\_position(head, index - 1);

if (last\_node == NULL) {

cerr << "remove\_node: index out of range";

return;

}

Node\* current\_node = last\_node->next;

Node\* next\_node = current\_node->next;

last\_node->next = next\_node;

delete current\_node;

}

Node\* merge(Node\* head1, Node\* head2) {

//假设head1和head2都递增有序, 返回新的合并后的链表

Node\* head3, \* pointer1, \* pointer2, \* pointer3;

if (head1->value < head2->value) {

head3 = new Node;

head3->value = head1->value;

pointer1 = head1->next;

pointer2 = head2;

}

else {

head3 = new Node;

head3->value = head2->value;

pointer1 = head1;

pointer2 = head2->next;

}

pointer3 = head3;

while (true) {

if (pointer1 == NULL) {

if (pointer2 == NULL) {

break;

}

else {

pointer3->next = new Node;

pointer3 = pointer3->next;

pointer3->value = pointer2->value;

pointer2 = pointer2->next;

}

}

else if (pointer2 == NULL) {

pointer3->next = new Node;

pointer3 = pointer3->next;

pointer3->value = pointer1->value;

pointer1 = pointer1->next;

}

else if(pointer1->value < pointer2->value){

pointer3->next = new Node;

pointer3 = pointer3->next;

pointer3->value = pointer1->value;

pointer1 = pointer1->next;

}

else {

pointer3->next = new Node;

pointer3 = pointer3->next;

pointer3->value = pointer2->value;

pointer2 = pointer2->next;

}

}

pointer3->next = NULL;

return head3;

}

int main() {

/\*Node\* head1 = create\_single\_linked\_list();

Node\* target = search\_by\_value(head1, 3);

//remove\_node(head1, 3);

//Node\* head2 = create\_single\_linked\_list();

//Node\* head3 = merge(head1, head2);

Node\* node3;

for (int i = 0; (node3 = search\_by\_position(head3, i)) != NULL; i++) {

cout << node3->value << ' ';

}

return 0;\*/

}

2