**数据结构课程设计**

**----个人设计报告**

专 业： 软件工程

班 级： 卓越软件2201

姓 名： 刘子睿

学 号： 221310304

指导教师： 周军锋

日 期： 2023年6月20日至2023年7月07日

**目录**

[1 课程设计目的 5](#_Toc139545270)

[2 任务完成情况 5](#_Toc139545271)

[3设计报告 6](#_Toc139545272)

[3.1 快速排序的应用 6](#_Toc139545273)

[3.1.1 题目以及要求 6](#_Toc139545274)

[3.1.2概要设计 6](#_Toc139545275)

[3.1.3 算法分析 6](#_Toc139545276)

[3.1.4使用说明: 7](#_Toc139545277)

[3.1.5 测试结果与分析 8](#_Toc139545278)

[3.2顺序表元素的快速删除 13](#_Toc139545279)

[3.2.1 题目以及要求 13](#_Toc139545280)

[3.2.2概要设计 13](#_Toc139545281)

[3.2.3 算法分析 13](#_Toc139545282)

[3.2.4使用说明: 14](#_Toc139545283)

[3.2.5 测试结果与分析 14](#_Toc139545284)

[3.3判断单链表的对称性 17](#_Toc139545285)

[3.3.1 题目以及要求 17](#_Toc139545286)

[3.2.2概要设计 17](#_Toc139545287)

[3.2.3 算法分析 17](#_Toc139545288)

[3.3.4使用说明: 18](#_Toc139545289)

[3.3.5 测试结果与分析 19](#_Toc139545290)

[3.4 搜索插入位置 20](#_Toc139545291)

[3.4.1题目及其要求 20](#_Toc139545292)

[3.4.2概要设计 20](#_Toc139545293)

[3.4.3算法分析 20](#_Toc139545294)

[3.4.4使用说明 21](#_Toc139545295)

[3.4.5测试结果与分析 21](#_Toc139545296)

[3.5有效的完全平方数 23](#_Toc139545297)

[3.5.1题目以及要求 23](#_Toc139545298)

[3.5.2概要设计 23](#_Toc139545299)

[3.5.3算法分析 23](#_Toc139545300)

[3.5.4使用说明 24](#_Toc139545301)

[3.5.5测试结果与分析 24](#_Toc139545302)

[3.6寻找比目标字母大的最小字母 26](#_Toc139545303)

[3.6.1题目以及要求 26](#_Toc139545304)

[3.6.2概要设计 26](#_Toc139545305)

[3.6.3算法分析 26](#_Toc139545306)

[3.6.4使用说明 27](#_Toc139545307)

[3.6.5测试结果与分析 27](#_Toc139545308)

[3.7矩阵中战斗力最弱的 K 行 29](#_Toc139545309)

[3.7.1题目以及要求 29](#_Toc139545310)

[3.7.2概要设计 29](#_Toc139545311)

[3.7.3算法分析: 29](#_Toc139545312)

[3.7.4使用说明: 29](#_Toc139545313)

[3.7.5测试结果与分析 30](#_Toc139545314)

[3.8找到和最大的长度为 K 的子序列 32](#_Toc139545315)

[3.8.1题目以及要求 32](#_Toc139545316)

[3.8.2算法设计 32](#_Toc139545317)

[3.8.3算法分析 32](#_Toc139545318)

[3.8.4使用说明 33](#_Toc139545319)

[测试结果与分析 33](#_Toc139545320)

[3.9无人机方阵 35](#_Toc139545321)

[3.9.1题目以及要求 35](#_Toc139545322)

[3.9.2概要设计 35](#_Toc139545323)

[3.9.3算法分析 35](#_Toc139545324)

[3.9.4使用说明 36](#_Toc139545325)

[3.9.5测试结果与分析 36](#_Toc139545326)

[3.10完成一半题目 39](#_Toc139545327)

[3.10.1题目描述 39](#_Toc139545328)

[3.10.2概要设计 39](#_Toc139545329)

[3.10.3算法分析 39](#_Toc139545330)

[3.10.4使用说明 40](#_Toc139545331)

[3.10.5测试结果与分析 40](#_Toc139545332)

[3.11 错误的集合 42](#_Toc139545333)

[3.11.1题目以及要求 42](#_Toc139545334)

[3.11.2概要设计 42](#_Toc139545335)

[3.11.3算法分析 42](#_Toc139545336)

[3.11.4使用说明 42](#_Toc139545337)

[3.11.5测试结果与分析 43](#_Toc139545338)

[3.12至少是其他数字两倍的最大数 45](#_Toc139545339)

[3.12.1题目以及要求 45](#_Toc139545340)

[3.12.2概要设计 45](#_Toc139545341)

[3.12.3算法分析 45](#_Toc139545342)

[3.12.4使用说明 45](#_Toc139545343)

[3.12.5测试结果与分析 46](#_Toc139545344)

[3.13 K 次取反后最大化的数组和 48](#_Toc139545345)

[3.13.1题目以及要求 48](#_Toc139545346)

[3.13.2概要设计 48](#_Toc139545347)

[3.13.3算法分析 48](#_Toc139545348)

[3.13.4使用说明 49](#_Toc139545349)

[3.13.5测试结果与分析 49](#_Toc139545350)

[3.14检查整数及其两倍数是否存在 52](#_Toc139545351)

[3.14.1题目以及要求 52](#_Toc139545352)

[3.14.2概要设计 52](#_Toc139545353)

[3.14.3算法分析 52](#_Toc139545354)

[3.14.4使用说明 52](#_Toc139545355)

[3.14.5测试结果与分析 53](#_Toc139545356)

[3.15数据流的第 K 大数值 55](#_Toc139545357)

[3.15.1题目以及要求 55](#_Toc139545358)

[3.15.2概要设计 55](#_Toc139545359)

[3.15.3算法分析 55](#_Toc139545360)

[3.15.4使用说明 56](#_Toc139545361)

[3.15.5测试结果与分析 56](#_Toc139545362)

## 课程设计目的

1. 学习获取知识的方法；
2. 提高发现问题、分析问题和解决实际问题的能力；
3. 加强创新意识和创新精神；
4. 掌握面向实际背景思考问题的方法。

## 2 任务完成情况

任务完成情况介绍，如2-1.(仅供参考，请根据实际完成情况填写)

**表2-1 任务完成情况表**

|  |  |  |
| --- | --- | --- |
| 任务序号 | 任务名称 | 完成情况 |
| 1 | 快速排序的应用 | AC |
| 2 | 顺序表元素的快速删除 | AC |
| 3 | 判断单链表的对称性 | AC |
| 4 | 有效的完全平方数 | AC |
| 5 | 寻找比目标字母大的最小字母 | AC |
| 6 | 寻找比目标字母大的最小字母 | AC |
| 7 | 矩阵中战斗力最弱的 K 行 | AC |
| 8 | 找到和最大的长度为 K 的子序列 | AC |
| 9 | 无人机方阵 | AC |
| 10 | 完成一半题目 | AC |
| 11 | 错误的集合 | AC |
| 12 | 至少是其他数字两倍的最大数 | AC |
| 13 | K 次取反后最大化的数组和 | AC |
| 14 | 检查整数及其两倍数是否存在 | AC |
| 15 | 数据流的第 K 大数值 | AC |

## 3设计报告

### 3.1 快速排序的应用

#### 3.1.1 题目以及要求

已知线性表按顺序存于内存，每个元素都是非零整数。在使用顺序表ADT的基础上，试设计基于快速排序的思想把所有值为负数的元素移到全部正数值元素前边的算法：例变为。

提示：要求重排*n*个元素且以顺序存储结构存储的线性表，使得所有值为负数的元素移到正数元素的前面。这可采用快速排序的思想来实现。只是比较的标准是元素是否为负数。因此枢轴元素的值为0(不是线性表中的元素)。基本思路是，先设置好上、下界和枢轴值(0)，然后执行一趟快速排序，即利用震荡交替法分别从线性表的两端查找正数和负数，找到后互相交换，直到上下界相遇。

参考函数原型：

template<class ElemType>

void Rearrange( SqList<ElemType> &A );

#### 3.1.2概要设计

设计一个deque\_chain双向链表，里面有三个保护成员： headNode， tailNode， listSize;

设计void arrange(deque\_chain<int>&a)函数实现快速排序，用void print()函数进行顺序表的输出。

#### 3.1.3 算法分析

**算法说明:**

采用双指针的思想，设置两个指针指向头和尾，一个从前往后移动，一个从后往前移动，如果从后往前的指针遇到负数且从前往后的指针没有遇到负数，则互相交换，直到两指针相遇。

核心代码部分：

**void** arrange(deque\_chain<**int**>&a)

{

chainNode<**int**>\*p=a.gethead()->next;

chainNode<**int**>\*q=a.getlast();

**int** len=0,height=a.getsize()-1;

**while**(len<height)

{

**if**(q->element<0)

{

**if**(p->element>=0)

{

swap(p->element,q->element);

p=p->next;

q=q->fa;

len++;

height--;

a.print();

**continue**;

}

**else**

{

p=p->next;

len++;

**continue**;

}

}**else**

{

q=q->fa;

height--;

**continue**;

}

}

}

**时间复杂度分析:**

将整个链表遍历一遍，故时间复杂度为;

#### 3.1.4使用说明:

输入说明 :

第一行：顺序表A的长度

第二行：顺序表A的数据元素(数据元素之间以空格分隔)(按照题目要求,**数据元素必须是整数**)

**运行界面如下:在xcode编译器的环境下可以正常运行**

文本

描述已自动生成

#### 3.1.5测试结果与分析

(1)特殊情况,全是正数,运行正常

IN:

8

1 3 2 4 5 6 7 8

OUT:

1 3 2 4 5 6 7 8

(2)特殊数据,全是负数,运行正常

IN:

7

-2 -1 -7 -5 -4 -3 -6

OUT:

-2 -1 -7 -5 -4 -3 -6

(3)普通数据,正负交替,运行正常

IN:

9

2 -1 4 3 -5 -6 7 -2 9

OUT:

2 -1 4 3 -5 -6 7 -2 9

-2 -1 4 3 -5 -6 7 2 9

-2 -1 -6 3 -5 4 7 2 9

-2 -1 -6 -5 3 4 7 2 9

(4)特殊情况,负数全在后面,运行正常

IN:

10

1 3 9 5 8 -4 -2 -8 -9 -6

OUT:

1 3 9 5 8 -4 -2 -8 -9 -6

-6 3 9 5 8 -4 -2 -8 -9 1

-6 -9 9 5 8 -4 -2 -8 3 1

-6 -9 -8 5 8 -4 -2 9 3 1

-6 -9 -8 -2 8 -4 5 9 3 1

-6 -9 -8 -2 -4 8 5 9 3 1

(5)普通数据,数据量适中,运行正常

IN:

20

3 5 6 -7 8 2 -10 45 24 -16 -18 22 37 65 -41 39 11 -13 -88 96

OUT：

3 5 6 -7 8 2 -10 45 24 -16 -18 22 37 65 -41 39 11 -13 -88 96

-88 5 6 -7 8 2 -10 45 24 -16 -18 22 37 65 -41 39 11 -13 3 96

-88 -13 6 -7 8 2 -10 45 24 -16 -18 22 37 65 -41 39 11 5 3 96

-88 -13 -41 -7 8 2 -10 45 24 -16 -18 22 37 65 6 39 11 5 3 96

-88 -13 -41 -7 -18 2 -10 45 24 -16 8 22 37 65 6 39 11 5 3 96

-88 -13 -41 -7 -18 -16 -10 45 24 2 8 22 37 65 6 39 11 5 3 96

(6)大数据测试，测试通过

IN:

100

-377 -726 -119 -794 931 620 147 806 866 49 -478 715 -670 -268 -979 156 -278 503 -149 -703 -60 -61 998 319 430 268 322 638 725 925 -633 -686 234 -554 968 679 -979 285 -253 37 -186 948 -301 -326 643 958 386 366 700 560 545 -313 242 218 174 793 -391 510 -238 861 278 -214 -364 -194 768 899 -887 83 322 817 349 349 902 -318 -917 -856 794 -134 937 -239 -960 914 -764 789 319 -431 -293 -41 864 649 -796 -597 -225 992 328 -718 -504 -262 -166 373

OUT：

-377 -726 -119 -794 931 620 147 806 866 49 -478 715 -670 -268 -979 156 -278 503 -149 -703 -60 -61 998 319 430 268 322 638 725 925 -633 -686 234 -554 968 679 -979 285 -253 37 -186 948 -301 -326 643 958 386 366 700 560 545 -313 242 218 174 793 -391 510 -238 861 278 -214 -364 -194 768 899 -887 83 322 817 349 349 902 -318 -917 -856 794 -134 937 -239 -960 914 -764 789 319 -431 -293 -41 864 649 -796 -597 -225 992 328 -718 -504 -262 -166 373

-377 -726 -119 -794 -166 620 147 806 866 49 -478 715 -670 -268 -979 156 -278 503 -149 -703 -60 -61 998 319 430 268 322 638 725 925 -633 -686 234 -554 968 679 -979 285 -253 37 -186 948 -301 -326 643 958 386 366 700 560 545 -313 242 218 174 793 -391 510 -238 861 278 -214 -364 -194 768 899 -887 83 322 817 349 349 902 -318 -917 -856 794 -134 937 -239 -960 914 -764 789 319 -431 -293 -41 864 649 -796 -597 -225 992 328 -718 -504 -262 931 373

-377 -726 -119 -794 -166 -262 147 806 866 49 -478 715 -670 -268 -979 156 -278 503 -149 -703 -60 -61 998 319 430 268 322 638 725 925 -633 -686 234 -554 968 679 -979 285 -253 37 -186 948 -301 -326 643 958 386 366 700 560 545 -313 242 218 174 793 -391 510 -238 861 278 -214 -364 -194 768 899 -887 83 322 817 349 349 902 -318 -917 -856 794 -134 937 -239 -960 914 -764 789 319 -431 -293 -41 864 649 -796 -597 -225 992 328 -718 -504 620 931 373

-377 -726 -119 -794 -166 -262 -504 806 866 49 -478 715 -670 -268 -979 156 -278 503 -149 -703 -60 -61 998 319 430 268 322 638 725 925 -633 -686 234 -554 968 679 -979 285 -253 37 -186 948 -301 -326 643 958 386 366 700 560 545 -313 242 218 174 793 -391 510 -238 861 278 -214 -364 -194 768 899 -887 83 322 817 349 349 902 -318 -917 -856 794 -134 937 -239 -960 914 -764 789 319 -431 -293 -41 864 649 -796 -597 -225 992 328 -718 147 620 931 373

-377 -726 -119 -794 -166 -262 -504 -718 866 49 -478 715 -670 -268 -979 156 -278 503 -149 -703 -60 -61 998 319 430 268 322 638 725 925 -633 -686 234 -554 968 679 -979 285 -253 37 -186 948 -301 -326 643 958 386 366 700 560 545 -313 242 218 174 793 -391 510 -238 861 278 -214 -364 -194 768 899 -887 83 322 817 349 349 902 -318 -917 -856 794 -134 937 -239 -960 914 -764 789 319 -431 -293 -41 864 649 -796 -597 -225 992 328 806 147 620 931 373

-377 -726 -119 -794 -166 -262 -504 -718 -225 49 -478 715 -670 -268 -979 156 -278 503 -149 -703 -60 -61 998 319 430 268 322 638 725 925 -633 -686 234 -554 968 679 -979 285 -253 37 -186 948 -301 -326 643 958 386 366 700 560 545 -313 242 218 174 793 -391 510 -238 861 278 -214 -364 -194 768 899 -887 83 322 817 349 349 902 -318 -917 -856 794 -134 937 -239 -960 914 -764 789 319 -431 -293 -41 864 649 -796 -597 866 992 328 806 147 620 931 373

-377 -726 -119 -794 -166 -262 -504 -718 -225 -597 -478 715 -670 -268 -979 156 -278 503 -149 -703 -60 -61 998 319 430 268 322 638 725 925 -633 -686 234 -554 968 679 -979 285 -253 37 -186 948 -301 -326 643 958 386 366 700 560 545 -313 242 218 174 793 -391 510 -238 861 278 -214 -364 -194 768 899 -887 83 322 817 349 349 902 -318 -917 -856 794 -134 937 -239 -960 914 -764 789 319 -431 -293 -41 864 649 -796 49 866 992 328 806 147 620 931 373

-377 -726 -119 -794 -166 -262 -504 -718 -225 -597 -478 -796 -670 -268 -979 156 -278 503 -149 -703 -60 -61 998 319 430 268 322 638 725 925 -633 -686 234 -554 968 679 -979 285 -253 37 -186 948 -301 -326 643 958 386 366 700 560 545 -313 242 218 174 793 -391 510 -238 861 278 -214 -364 -194 768 899 -887 83 322 817 349 349 902 -318 -917 -856 794 -134 937 -239 -960 914 -764 789 319 -431 -293 -41 864 649 715 49 866 992 328 806 147 620 931 373

-377 -726 -119 -794 -166 -262 -504 -718 -225 -597 -478 -796 -670 -268 -979 -41 -278 503 -149 -703 -60 -61 998 319 430 268 322 638 725 925 -633 -686 234 -554 968 679 -979 285 -253 37 -186 948 -301 -326 643 958 386 366 700 560 545 -313 242 218 174 793 -391 510 -238 861 278 -214 -364 -194 768 899 -887 83 322 817 349 349 902 -318 -917 -856 794 -134 937 -239 -960 914 -764 789 319 -431 -293 156 864 649 715 49 866 992 328 806 147 620 931 373

-377 -726 -119 -794 -166 -262 -504 -718 -225 -597 -478 -796 -670 -268 -979 -41 -278 -293 -149 -703 -60 -61 998 319 430 268 322 638 725 925 -633 -686 234 -554 968 679 -979 285 -253 37 -186 948 -301 -326 643 958 386 366 700 560 545 -313 242 218 174 793 -391 510 -238 861 278 -214 -364 -194 768 899 -887 83 322 817 349 349 902 -318 -917 -856 794 -134 937 -239 -960 914 -764 789 319 -431 503 156 864 649 715 49 866 992 328 806 147 620 931 373

-377 -726 -119 -794 -166 -262 -504 -718 -225 -597 -478 -796 -670 -268 -979 -41 -278 -293 -149 -703 -60 -61 -431 319 430 268 322 638 725 925 -633 -686 234 -554 968 679 -979 285 -253 37 -186 948 -301 -326 643 958 386 366 700 560 545 -313 242 218 174 793 -391 510 -238 861 278 -214 -364 -194 768 899 -887 83 322 817 349 349 902 -318 -917 -856 794 -134 937 -239 -960 914 -764 789 319 998 503 156 864 649 715 49 866 992 328 806 147 620 931 373

-377 -726 -119 -794 -166 -262 -504 -718 -225 -597 -478 -796 -670 -268 -979 -41 -278 -293 -149 -703 -60 -61 -431 -764 430 268 322 638 725 925 -633 -686 234 -554 968 679 -979 285 -253 37 -186 948 -301 -326 643 958 386 366 700 560 545 -313 242 218 174 793 -391 510 -238 861 278 -214 -364 -194 768 899 -887 83 322 817 349 349 902 -318 -917 -856 794 -134 937 -239 -960 914 319 789 319 998 503 156 864 649 715 49 866 992 328 806 147 620 931 373

-377 -726 -119 -794 -166 -262 -504 -718 -225 -597 -478 -796 -670 -268 -979 -41 -278 -293 -149 -703 -60 -61 -431 -764 -960 268 322 638 725 925 -633 -686 234 -554 968 679 -979 285 -253 37 -186 948 -301 -326 643 958 386 366 700 560 545 -313 242 218 174 793 -391 510 -238 861 278 -214 -364 -194 768 899 -887 83 322 817 349 349 902 -318 -917 -856 794 -134 937 -239 430 914 319 789 319 998 503 156 864 649 715 49 866 992 328 806 147 620 931 373

-377 -726 -119 -794 -166 -262 -504 -718 -225 -597 -478 -796 -670 -268 -979 -41 -278 -293 -149 -703 -60 -61 -431 -764 -960 -239 322 638 725 925 -633 -686 234 -554 968 679 -979 285 -253 37 -186 948 -301 -326 643 958 386 366 700 560 545 -313 242 218 174 793 -391 510 -238 861 278 -214 -364 -194 768 899 -887 83 322 817 349 349 902 -318 -917 -856 794 -134 937 268 430 914 319 789 319 998 503 156 864 649 715 49 866 992 328 806 147 620 931 373

-377 -726 -119 -794 -166 -262 -504 -718 -225 -597 -478 -796 -670 -268 -979 -41 -278 -293 -149 -703 -60 -61 -431 -764 -960 -239 -134 638 725 925 -633 -686 234 -554 968 679 -979 285 -253 37 -186 948 -301 -326 643 958 386 366 700 560 545 -313 242 218 174 793 -391 510 -238 861 278 -214 -364 -194 768 899 -887 83 322 817 349 349 902 -318 -917 -856 794 322 937 268 430 914 319 789 319 998 503 156 864 649 715 49 866 992 328 806 147 620 931 373

-377 -726 -119 -794 -166 -262 -504 -718 -225 -597 -478 -796 -670 -268 -979 -41 -278 -293 -149 -703 -60 -61 -431 -764 -960 -239 -134 -856 725 925 -633 -686 234 -554 968 679 -979 285 -253 37 -186 948 -301 -326 643 958 386 366 700 560 545 -313 242 218 174 793 -391 510 -238 861 278 -214 -364 -194 768 899 -887 83 322 817 349 349 902 -318 -917 638 794 322 937 268 430 914 319 789 319 998 503 156 864 649 715 49 866 992 328 806 147 620 931 373

-377 -726 -119 -794 -166 -262 -504 -718 -225 -597 -478 -796 -670 -268 -979 -41 -278 -293 -149 -703 -60 -61 -431 -764 -960 -239 -134 -856 -917 925 -633 -686 234 -554 968 679 -979 285 -253 37 -186 948 -301 -326 643 958 386 366 700 560 545 -313 242 218 174 793 -391 510 -238 861 278 -214 -364 -194 768 899 -887 83 322 817 349 349 902 -318 725 638 794 322 937 268 430 914 319 789 319 998 503 156 864 649 715 49 866 992 328 806 147 620 931 373

-377 -726 -119 -794 -166 -262 -504 -718 -225 -597 -478 -796 -670 -268 -979 -41 -278 -293 -149 -703 -60 -61 -431 -764 -960 -239 -134 -856 -917 -318 -633 -686 234 -554 968 679 -979 285 -253 37 -186 948 -301 -326 643 958 386 366 700 560 545 -313 242 218 174 793 -391 510 -238 861 278 -214 -364 -194 768 899 -887 83 322 817 349 349 902 925 725 638 794 322 937 268 430 914 319 789 319 998 503 156 864 649 715 49 866 992 328 806 147 620 931 373

-377 -726 -119 -794 -166 -262 -504 -718 -225 -597 -478 -796 -670 -268 -979 -41 -278 -293 -149 -703 -60 -61 -431 -764 -960 -239 -134 -856 -917 -318 -633 -686 -887 -554 968 679 -979 285 -253 37 -186 948 -301 -326 643 958 386 366 700 560 545 -313 242 218 174 793 -391 510 -238 861 278 -214 -364 -194 768 899 234 83 322 817 349 349 902 925 725 638 794 322 937 268 430 914 319 789 319 998 503 156 864 649 715 49 866 992 328 806 147 620 931 373

-377 -726 -119 -794 -166 -262 -504 -718 -225 -597 -478 -796 -670 -268 -979 -41 -278 -293 -149 -703 -60 -61 -431 -764 -960 -239 -134 -856 -917 -318 -633 -686 -887 -554 -194 679 -979 285 -253 37 -186 948 -301 -326 643 958 386 366 700 560 545 -313 242 218 174 793 -391 510 -238 861 278 -214 -364 968 768 899 234 83 322 817 349 349 902 925 725 638 794 322 937 268 430 914 319 789 319 998 503 156 864 649 715 49 866 992 328 806 147 620 931 373

-377 -726 -119 -794 -166 -262 -504 -718 -225 -597 -478 -796 -670 -268 -979 -41 -278 -293 -149 -703 -60 -61 -431 -764 -960 -239 -134 -856 -917 -318 -633 -686 -887 -554 -194 -364 -979 285 -253 37 -186 948 -301 -326 643 958 386 366 700 560 545 -313 242 218 174 793 -391 510 -238 861 278 -214 679 968 768 899 234 83 322 817 349 349 902 925 725 638 794 322 937 268 430 914 319 789 319 998 503 156 864 649 715 49 866 992 328 806 147 620 931 373

-377 -726 -119 -794 -166 -262 -504 -718 -225 -597 -478 -796 -670 -268 -979 -41 -278 -293 -149 -703 -60 -61 -431 -764 -960 -239 -134 -856 -917 -318 -633 -686 -887 -554 -194 -364 -979 -214 -253 37 -186 948 -301 -326 643 958 386 366 700 560 545 -313 242 218 174 793 -391 510 -238 861 278 285 679 968 768 899 234 83 322 817 349 349 902 925 725 638 794 322 937 268 430 914 319 789 319 998 503 156 864 649 715 49 866 992 328 806 147 620 931 373

-377 -726 -119 -794 -166 -262 -504 -718 -225 -597 -478 -796 -670 -268 -979 -41 -278 -293 -149 -703 -60 -61 -431 -764 -960 -239 -134 -856 -917 -318 -633 -686 -887 -554 -194 -364 -979 -214 -253 -238 -186 948 -301 -326 643 958 386 366 700 560 545 -313 242 218 174 793 -391 510 37 861 278 285 679 968 768 899 234 83 322 817 349 349 902 925 725 638 794 322 937 268 430 914 319 789 319 998 503 156 864 649 715 49 866 992 328 806 147 620 931 373

-377 -726 -119 -794 -166 -262 -504 -718 -225 -597 -478 -796 -670 -268 -979 -41 -278 -293 -149 -703 -60 -61 -431 -764 -960 -239 -134 -856 -917 -318 -633 -686 -887 -554 -194 -364 -979 -214 -253 -238 -186 -391 -301 -326 643 958 386 366 700 560 545 -313 242 218 174 793 948 510 37 861 278 285 679 968 768 899 234 83 322 817 349 349 902 925 725 638 794 322 937 268 430 914 319 789 319 998 503 156 864 649 715 49 866 992 328 806 147 620 931 373

-377 -726 -119 -794 -166 -262 -504 -718 -225 -597 -478 -796 -670 -268 -979 -41 -278 -293 -149 -703 -60 -61 -431 -764 -960 -239 -134 -856 -917 -318 -633 -686 -887 -554 -194 -364 -979 -214 -253 -238 -186 -391 -301 -326 -313 958 386 366 700 560 545 643 242 218 174 793 948 510 37 861 278 285 679 968 768 899 234 83 322 817 349 349 902 925 725 638 794 322 937 268 430 914 319 789 319 998 503 156 864 649 715 49 866 992 328 806 147 620 931 373

### 3.2顺序表元素的快速删除

#### 3.2.1 题目以及要求

已知长度为n的线性表A采用顺序存储结构，请在应用顺序表ADT的基础上，设计一时间复杂度为0(n)、空间复杂度为0(1)的算法，该算法删除线性表中所有值为item的数据元素。要求：线性表元素个数n很大，而值为item的数据元素个数很少，要求移动元素个数尽量少；删除后的数组元素与原数组元素不必保持顺序一致。

提示：算法参照快速排序的思想，使用两个指针，分别从左往右扫描找到需要被删除的元素、从右往左扫描找到不被删除的元素，然后交换。然后继续扫描。

参考函数原型：

//删除线性表中所有值为item的数据元素，时间复杂度为0(n)、空间复杂度为0(1)

template<class ElemType>

void DeleteItem( SqList<ElemType> &A, ElemType item );

#### 3.2.2概要设计

设计一个deque\_chain双向链表，里面有三个保护成员： headNode， tailNode， listSize;

设计void Delete(deque\_chain<int>&a,int num)函数实现删除线性表中指定的数据元素，并用void print()函数打印出来。

#### 3.2.3 算法分析

**算法说明:**

采用双指针的思想，设置两个指针指向头和尾，头指针从左往右移动，尾指针不动，头指针从左往右扫描找到需要被删除的元素时，先判断尾指针指向的元素是否是要被删除的元素，如果不是则交换，如果是则尾指针向前移动直到发现不需要被删除的元素为止。

核心代码部分:

void Delete(deque\_chain<int>&a,int num)

{

chainNode<int>\*p=a.get\_head()->next;

chainNode<int>\*q=a.get\_tail();

int l=0,h=a.get\_size()-1;

while(l<h)

{

if(p->element==num)

{

if(q->element!=num)

{

swap(q->element,p->element);

l++;

h--;

p=p->next;

q=q->fa;

a.pop\_back();

}

else

{

q=q->fa;

a.pop\_back();

h--;

}

}

else

{

p=p->next;

l++;

}

}

if(a.get\_tail()->element==num)

{

a.pop\_back();

}

}

**时间复杂度分析:**

只循环一遍链表,故时间复杂度为;

#### 3.2.4使用说明:

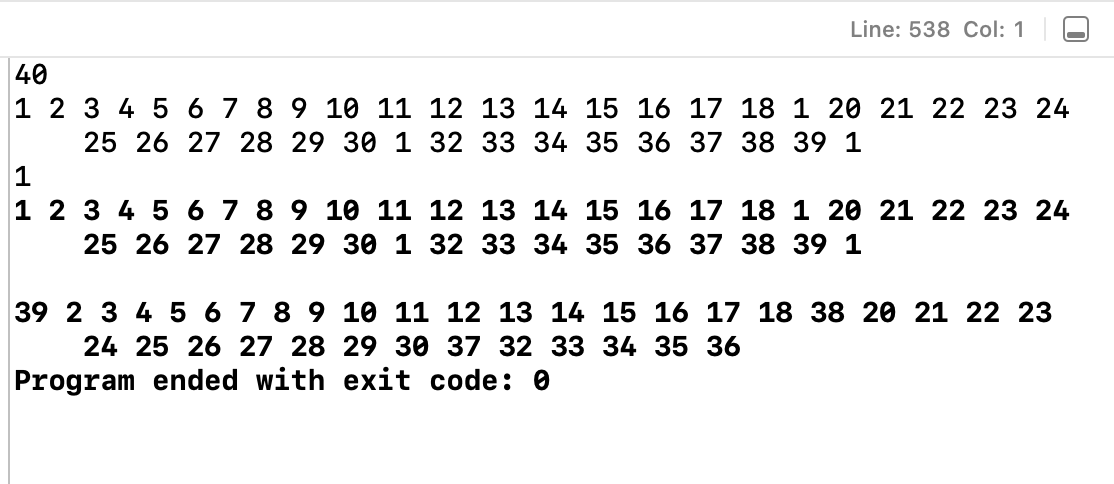
输入说明 :

第一行：顺序表A的长度

第二行：顺序表A的数据元素(数据元素之间以空格分隔)

第三行：待删除数据元素

**运行界面如下:在xcode编译器的环境下可以正常运行**



#### 测试结果与分析

(1)特殊情况，仅1个元素，且需要被删除，返回空表

IN:

1

1

1

OUT:

1

empty

(2)特殊情况，全部元素都需要被删除，返回空表

IN:

5

2 2 2 2 2

2

OUT:

2 2 2 2 2

empty

(3)特殊情况，需要删除的元素都在末尾

IN:

10

1 6 8 9 2 5 5 5 5 5

5

OUT:

1 6 8 9 2 5 5 5 5 5

1 6 8 9 2

(4)特殊情况,没有要被删除的元素

IN:

20

19 20 3 4 2 5 89 38 24 38 99 31 8 3 2 35 17 51 23 66

33

OUT:

19 20 3 4 2 5 89 38 24 38 99 31 8 3 2 35 17 51 23 66

19 20 3 4 2 5 89 38 24 38 99 31 8 3 2 35 17 51 23 66

(5)普通数据，正常数据测试

IN:

15

2 4 1 7 8 19 23 5 2 9 5 3 4 9 4

4

OUT:

2 4 1 7 8 19 23 5 2 9 5 3 4 9 4

2 9 1 7 8 19 23 5 2 9 5 3

(6)大数据测试，测试通过

IN:

200

-707 -802 57 983 93 -583 643 -522 -339 -688 478 931 -555 832 549 -61 -417 72 538 813 181 252 -88 926 802 713 -308 470 612 -725 -541 474 291 -200 -106 -469 388 -289 -776 828 -298 343 185 -469 -134 550 703 281 151 -13 960 -382 261 -600 -49 -864 344 556 840 -477 845 -18 -278 598 416 86 251 -654 -311 -28 -97 288 360 625 -565 -469 503 25 -773 697 -157 436 -469 -914 -968 -893 -637 439 -15 437 -469 450 -589 552 440 -802 -283 -359 648 854 -447 -426 339 703 812 866 12 23 118 -189 625 -777 23 -318 173 -939 193 -610 -1000 221 -695 -350 508 -132 464 -876 355 864 -450 -516 725 819 -731 702 780 862 226 -948 -502 110 487 807 47 -394 -435 -404 190 -469 -323 724 -349 367 119 -714 -58 877 -445 956 -82 -819 -419 -966 299 356 -418 151 768 -386 -768 160 127 -615 909 0 193 439 255 894 -406 516 228 -637 803 -134 -469 702 160 913 -621 -907 709 -554 -268 -706 -88 370 -187 145 988 -689

-469

OUT:

-707 -802 57 983 93 -583 643 -522 -339 -688 478 931 -555 832 549 -61 -417 72 538 813 181 252 -88 926 802 713 -308 470 612 -725 -541 474 291 -200 -106 -469 388 -289 -776 828 -298 343 185 -469 -134 550 703 281 151 -13 960 -382 261 -600 -49 -864 344 556 840 -477 845 -18 -278 598 416 86 251 -654 -311 -28 -97 288 360 625 -565 -469 503 25 -773 697 -157 436 -469 -914 -968 -893 -637 439 -15 437 -469 450 -589 552 440 -802 -283 -359 648 854 -447 -426 339 703 812 866 12 23 118 -189 625 -777 23 -318 173 -939 193 -610 -1000 221 -695 -350 508 -132 464 -876 355 864 -450 -516 725 819 -731 702 780 862 226 -948 -502 110 487 807 47 -394 -435 -404 190 -469 -323 724 -349 367 119 -714 -58 877 -445 956 -82 -819 -419 -966 299 356 -418 151 768 -386 -768 160 127 -615 909 0 193 439 255 894 -406 516 228 -637 803 -134 -469 702 160 913 -621 -907 709 -554 -268 -706 -88 370 -187 145 988 -689

-707 -802 57 983 93 -583 643 -522 -339 -688 478 931 -555 832 549 -61 -417 72 538 813 181 252 -88 926 802 713 -308 470 612 -725 -541 474 291 -200 -106 -689 388 -289 -776 828 -298 343 185 988 -134 550 703 281 151 -13 960 -382 261 -600 -49 -864 344 556 840 -477 845 -18 -278 598 416 86 251 -654 -311 -28 -97 288 360 625 -565 145 503 25 -773 697 -157 436 -187 -914 -968 -893 -637 439 -15 437 370 450 -589 552 440 -802 -283 -359 648 854 -447 -426 339 703 812 866 12 23 118 -189 625 -777 23 -318 173 -939 193 -610 -1000 221 -695 -350 508 -132 464 -876 355 864 -450 -516 725 819 -731 702 780 862 226 -948 -502 110 487 807 47 -394 -435 -404 190 -88 -323 724 -349 367 119 -714 -58 877 -445 956 -82 -819 -419 -966 299 356 -418 151 768 -386 -768 160 127 -615 909 0 193 439 255 894 -406 516 228 -637 803 -134 -706 702 160 913 -621 -907 709 -554 -268

### 3.3判断单链表的对称性

#### 3.3.1 题目以及要求

设带头结点的单链表的头指针为head，结点结构由data和next两个域构成，其中data域为字符型。在使用单链表ADT的基础上，设计一算法判断该链表的前*n*个字符是否中心对称。例如 *x, xyx,  xyyx*都是中心对称。

参考函数原型：

//判断单链表的前n个结点是否中心对称

template<class ElemType>

bool Judge\_Symmetry( LinkList<ElemType> &L, int num );

#### 3.2.2概要设计

使用单链表deque\_chain()，在此基础上设计**bool** judge1()函数和**bool** judge2(**int** n)函数判断该链表的前n个字符是否中心对称，并用**void** print()打印单链表。

#### 3.2.3 算法分析

**算法说明:**

先将字符串流转换成单链表，如果元素个数小于n，则用**bool** judge1()函数判断整个单链表是否对称，一个指针指向头节点，一个指针指向尾结点，头指针从前往后移动，尾指针从后往前移动，如果头尾指针指向的元素不相同则退出循环输出false，否则继续直至头尾指针相遇；如果元素个数大于n，则用**bool** judge2()函数判断对称性，原理同**bool** judge1()函数。

核心代码部分:

**bool** judge1()

{

chainNode<T>\*p1=headNode->next;

chainNode<T>\*p2=tailNode;

**while**(p1->next!=p2)

{

**if**(p1->element==p2->element)

{

p1=p1->next;

p2=p2->fa;

}

**else**

**return** 0;

**break**;

}

**return** 1;

}

**bool** judge2(**int** n)

{

chainNode<T>\*p1=headNode->next;

chainNode<T>\*p2=headNode->next;

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

{

p2=p2->next;

}

**while**(p1->next!=p2)

{

**if**(p1->element==p2->element)

{

p1=p1->next;

p2=p2->fa;

}

**else**

{

**return** 0;

**break**;

}

}

**return** 1;

}

**时间复杂度分析:**

只遍历了一遍链表,故时间复杂度为

#### 3.3.4使用说明:

输入说明 :

第一行：顺序表A的数据元素的数据类型标记

第二行：待判断对称性的链表长度

第三行：单链表A的数据元素(数据元素之间以空格分隔)

注意：单链表A中的元素个数可能小于，也可能大于。如果元素个数小于，则判断整个单链表是否对称。

**运行界面如下:在xcode编译器的环境下可以正常运行**

文本

描述已自动生成

#### 测试结果与分析

(1)err测试,通过

IN:

6

OUT:

err

(2)int类型，元素个数小于n，特殊情况，仅一个元素，通过;

IN:

0

5

3

OUT:

3

true

1. double类型,满足元素个数小于n，普通情况，对称，通过

IN:

1

9

1.3 4.2 5.7 6.6 3.3 8.5

OUT:

1.3->4.2->5.7->6.6->3.3->8.5

false

1. string类型,满足元素个数大于n，普通情况，不对称，通过

IN:

3

5

ab& d\*c fdq fdq d\*c ab&

OUT:

ab&->d\*c->fdq->fdq->d\*c->ab&

false

1. char类型,满足元素个数大于n，普通情况，对称，通过

IN:

2

6

\* @ ! ! @ \* b

OUT:

\*->@->!->!->@->\*->b

true

### 3.4 搜索插入位置

#### 3.4.1题目及其要求

给定一个排序数组和一个目标值，在数组中找到目标值，并返回其索引。如果目标值不存在于数组中，返回它将会被按顺序插入的位置。

请必须使用时间复杂度为的算法。

#### 3.4.2概要设计

采用二分查找思想，设计一个main函数，如果能在数组中找到目标值，则返回其索引；如果目标值不存在于数组中，则返回它将会被按顺序插入的位置。

#### 3.4.3算法分析

**算法说明:**

不断用二分查找进行算法设计

本题核心代码思路如下:

**int** main()

{

**int** n;

cin>>n;

**int** a[10000];

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

{

cin>>a[i];

}

**int** target;

**int** point=0;

cin>>target;

**int** low=0;

**int** high=n-1;

**int** mid;

**while**(low<=high)

{

mid=(low+high)/2;

**if**(a[mid]==target)

{

cout<<mid;

point=1;

**break**;

}

**else** **if**(a[mid]>target)

{

high=mid-1;

}

**else**

{

low=mid+1;

}

}

**if**(point==0)

{

**if**(target>a[n-1])

cout<<mid+1;

**else**

cout<<mid;

}

**return** 0;

}

**时间复杂度分析：**

使用二分查找的方式,故时间复杂度为

#### 3.4.4使用说明

输入说明 :

输入三行：

第一行输入一个整数n表示数组nums的长度。

第二行输入n个整数表示数组nums的元素。

第三行输入一个整数表示需要查找的目标值target.

提示:



为无重复元素的升序排列数组



**运行界面如下:在xcode编译器的环境下可以正常运行**

图表

低可信度描述已自动生成

#### 3.4.5测试结果与分析

(1)特殊情况，表中仅有一个元素，插入统一的元素，运行正确:

IN:

1

1

1

OUT:

0

(2)特殊情况，插入表头，正确

IN:

6

1 2 3 4 5 6

0

OUT:

0

(3)特殊情况，插入表尾，正确

IN:

10

1 2 3 4 5 6 7 8 9 10

11

OUT:

10

(4)普通插入(元素在数组中)，正确

IN:

20

1 2 3 4 5 6 7 8 9 10 23 31 47 58 62 69 72 77 89 99

23

OUT:

10

(5)普通插入(元素不在数组中),正确

IN:

20

1 2 4 5 6 7 9 10 23 31 35 43 47 58 62 69 72 77 89 99

33

OUT:

10

### 3.5有效的完全平方数

#### 3.5.1题目以及要求

给定一个 正整数 *num* ，编写一个函数，如果 *num* 是一个完全平方数，则返回 *true* ，否则返回 *false* 。

进阶：不要 使用任何内置的库函数，如  *sqrt* 。

#### 3.5.2概要设计

利用二分查找的思想，设计一个**bool** judge(**long** **long** num)函数判断*num* 是不是一个完全平方数，定义long long数据类型防止数据溢出。

#### 3.5.3算法分析

**算法说明:**

通过二分查找的方式不断查找完全平方数，直到输出结果。

本题核心代码如下:

**bool** judge(**long** **long** num)

{

**long** **long** low=1;

**long** **long** high=num;

**long** **long** mid;

**while**(low<=high)

{

mid=(low+high)/2;

**if**((mid\*mid)==num)

{

**return** **true**;

}

**else** **if**(mid\*mid>num)

{

high=mid-1;

}

**else**

{

low=mid+1;

}

}

**return** **false**;

}

**int** main()

{

**long** **long** num;

cin>>num;

**if**(judge(num))

{

cout<<"true";

}

**else**

cout<<"false";

**return** 0;

}

**时间复杂度分析:**

由于是二分查找的思想,故时间复杂度是的

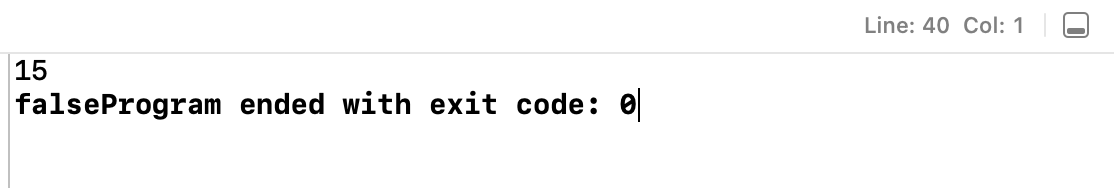
#### 3.5.4使用说明

输入说明 :

输入一个整数num.

提示：

**运行界面如下:在xcode编译器的环境下可以正常运行**



#### 3.5.5测试结果与分析

(1)特殊情况:1

IN:

1

OUT:

true

(2)普通情况，完全平方数

IN:

81

OUT:

true

(3)普通情况，看起来不像完全平方数的数字

IN:

361

OUT:

true

(4)普通情况，看起来像完全平方数的数字

IN:

999

OUT:

false

(5)大数据(边界2^ 31)

IN:

2147483648

OUT:

false

### 3.6寻找比目标字母大的最小字母

#### 3.6.1题目以及要求

给你一个排序后的字符列表 letters ，列表中只包含小写英文字母。另给出一个目标字母 target，请你寻找在这一有序列表里比目标字母大的最小字母。

在比较时，字母看成是依序循环出现的。即：如果目标字母大于等于letters中最后一个字符，则返回第0个字符。

#### 3.6.2概要设计

设计一个main函数，遍历字符列表，不断进行比较直至输出结果。

#### 3.6.3算法分析

**算法说明:**

利用for循环对当前字母和给定字母进行比较，如果当前字母大于给定字母，则返回当前字母；如果遍历完都没有找到大于给定字母的字母，则返回第一个字符。

**本题核心代码如下:**

**int** main()

{

**int** len;

**char** letters[10000];

**char** target;

cin>>len;

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

{

cin>>letters[i];

}

cin>>target;

**int** i=0;

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

{

**if**(letters[i]>target) {

cout << letters[i];

**break**;

}

}

**if**(i==len)

{

cout<<letters[0];

}

**return** 0;

}

**时间复杂度分析:**

遍历整个表两遍,时间复杂度为*O*(2*n*)～*O*(*n*)，写复杂度时常数系数可省略。

#### 3.6.4使用说明

输入说明 :

输入三行：

第一行输入一个整数n表示数组的长度。

第二行输入n个小写英文字母表示数组letters的元素。

第三行输入一个小写英文字母表示目标字符target.

提示：



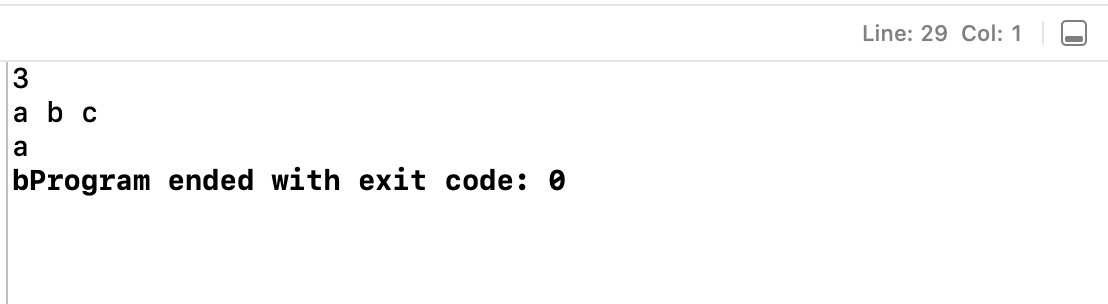
    letters[i] 是一个小写字母

    letters 按非递减顺序排序

    letters 最少包含两个不同的字母

target 是一个小写字母

**运行界面如下:在xcode编译器的环境下可以正常运行**



#### 3.6.5测试结果与分析

(1)特殊数据,所有字母均一样,查找的字符不等于这个字符

IN:

10

a a a a a a a a a a

b

OUT:

a

(2)特殊数据, 所有字母均一样,查找的字符等于这个字符

IN:

10

b b b b b b b b b b

b

OUT:

b

(3)普通数据，target在字符集中

IN:

10

a a b c d d h i j k

c

OUT:

d

(4)普通数据， target不在字符集中

IN:

10

a b c d e h i j k m

g

OUT:

h

(5)大数据测试，target在字符集中

IN:

130

a a a a a b b b b b c c c c c d d d d d e e e e e f f f f f g g g g g h h h h h I I I I I j j j j j k k k k k l l l l l m m m m m n n n n n o o o o o p p p p p q q q q q r r r r r s s s s s t t t t t u u u u u v v v v v w w w w w x x x x x y y y y y z z z z z

m

OUT:

n

### 3.7矩阵中战斗力最弱的 K 行

#### 3.7.1题目以及要求

给你一个大小为的矩阵 mat，矩阵由若干军人和平民组成，分别用 1 和 0 表示。

请你返回矩阵中战斗力最弱的 *k* 行的索引(行号)，按从最弱到最强排序。

如果第 *i* 行的军人数量少于第 *j* 行，或者两行军人数量相同但 *i* 小于 *j*，那么我们认为第 *i* 行的战斗力比第 *j* 行弱。

军人总是排在一行中的靠前位置，也就是说 1 总是出现在 0 之前。

#### 3.7.2概要设计

设计结构体**struct** Fight，内有成员sum和index，设计一个**bool** cmp(Fight a1,Fight a2)函数比较行间战斗力的强弱即军人数量的多少，最后用sort(fight,fight+m,cmp)函数排序，输出sum值最小的结构体的行数index。

#### 3.7.3算法分析:

**算法说明:**

统计每一排的战斗力强弱，进行排序，按顺序输出即可。

本题核心代码如下:

**bool** cmp(Fight a1,Fight a2)

{

**if**(a1.sum==a2.sum){**return** a1.index<a2.index;}

**else**

{

**return** a1.sum<a2.sum;

}

}

**时间复杂度分析:**

遍历整个表复杂度*O(mn)*,快速排序sort耗时*O(nlog n)*，由于这个算法的复杂度由m和n决定,我们无法知道m和n谁大,故无法根据加法法则省略掉其中的一个,故总复杂度为*O*(*mn*+*n*log*n*)﻿。

#### 3.7.4使用说明:

输入说明 :

输入若干行：

第一行输入两个整数m和n，表示矩阵的行数和列数。

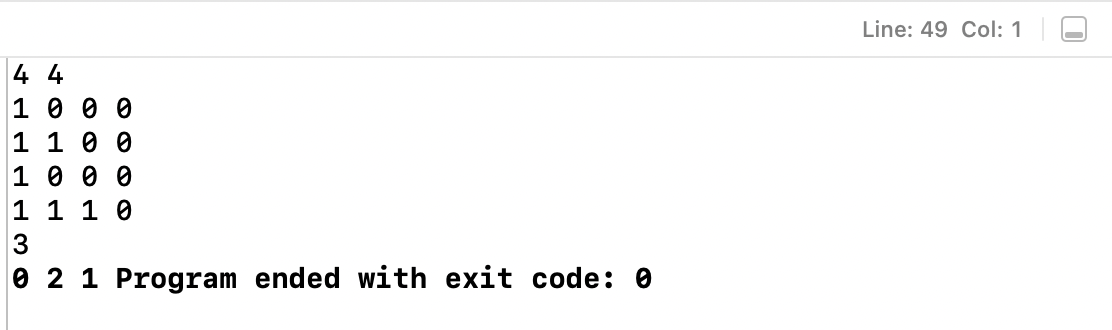
之后m行，每行输入n个整数(0或1)表示矩阵的元素。

最后一行输入一个整数.

提示：



    矩阵的元素 不是 0 就是 1

**运行界面如下:在xcode编译器的环境下可以正常运行** 

#### 3.7.5测试结果与分析

(1)特殊数据全是0,运行正确

IN:

4 4

0 0 0 0

0 0 0 0

0 0 0 0

0 0 0 0

4

OUT:

0 1 2 3

(2)特殊数据全是1,运行正确

IN:

4 4

1 1 1 1

1 1 1 1

1 1 1 1

1 1 1 1

2

OUT:

0 1

(3)特殊数据,各排1和0的数量都不同的,运行正确

IN:

5 5

1 0 0 0 0

1 1 0 0 0

1 1 1 0 0

1 1 1 1 0

1 1 1 1 1

4

OUT:

0 1 2 3

(4)普通数据, 0和1分布随机,运行正确

IN:

5 5

1 1 1 0 0

1 0 0 0 0

1 0 0 0 0

1 1 0 0 0

1 1 1 1 0

5

OUT:

1 2 3 0 4

(5)普通数据,但不为方阵, 且某些排0和1的数量相同,且分布随机,测试通过

IN：

6 4

1 0 0 0

1 1 0 0

1 1 1 0

1 0 0 0

1 1 1 1

1 1 0 0

4

OUT:

0 3 1 5

(6)大数据测试，测试通过

### 3.8找到和最大的长度为 K 的子序列

#### 3.8.1题目以及要求

给你一个整数数组 *nums* 和一个整数 *k* 。你需要找到 *nums* 中长度为 *k* 的子序列 ，且这个子序列的和最大 。

请你返回 任意 一个长度为 *k* 的整数子序列。

子序列定义为从一个数组里删除一些元素后，不改变剩下元素的顺序得到的数组。

#### 3.8.2算法设计

设计一个main函数，不断遍历删除每次的最小元素，输出剩余元素。

#### 3.8.3算法分析

**算法说明:**

用一个vector数组存放所有元素，设计一个for循环，从第一个开始遍历，遍历都找到一个最小值，进行n-k次遍历并删除每次的最小元素，输出剩余元素。

本题的核心代码如下:

**int** main()

{

**int** n;

cin>>n;

**int** nums;

vector<**int**>xulie;

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

{

cin>>nums;

xulie.push\_back(nums);

}

**int** k;

cin>>k;

**int** min=INT\_MAX;

**int** in\_dex;

**int** tmp=n-k;

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

{

min=INT\_MAX;

**for**(**int** j=0;j<xulie.size();j++)

{

**if** (xulie[j]<= min)

{

min = xulie[j];

in\_dex = j;

}

}

xulie.erase(xulie.begin()+in\_dex);

}

**for**(**int** i=0;i<xulie.size();i++)

{

cout<<xulie[i]<<" ";

}

**return** 0;

}

**时间复杂度分析:**

先分析最内层,内层的for 循环+erase 第一次查找删除操作耗时*O*(2*n*),第二次耗时O(2(n-1)),...根据外层函数tmp=n-k是最开始固定的常数,故总时间复杂度为 的。

#### 3.8.4使用说明

输入说明 :

输入三行：

第一行为一个整数*n*代表数组*nums*的长度。

第二行输入*n*个整数代表数组*nums*的元素。

第三行为一个整数*k*。

提示：



**运行界面如下:在xcode编译器的环境下可以正常运行**

图形用户界面, 图表

中度可信度描述已自动生成

#### 测试结果与分析

(1)特殊情况，输入数据没有负数，运行正确

IN:

8

1 7 6 2 9 13 5 8

5

OUT:

7 6 9 13 8

(2)特殊情况，输入数据没有正数，运行正确

IN:

10

-3 -12 -6 -8 -2 -24 -16 -5 -9 -13

6

OUT:

-3 -6 -8 -2 -5 -9

(3)特殊情况，即无需调用删除操作，测运行正确

IN:

10

-5 -4 -3 -1 -9 1 2 5 6 7

10

OUT:

-5 -4 -3 -1 -9 1 2 5 6 7

(4)特殊情况，所有元素全部相同，运行正确

IN:

20

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

5

OUT:

1 1 1 1 1

(5)普通数据，元素顺序随机，运行正确

IN:

10

8 12 -7 3 -19 26 3 -8 5 21

8

OUT:

8 12 -7 3 26 3 5 21

(6)大数据测试，测试通过

### 3.9无人机方阵

#### 3.9.1题目以及要求

在 「挑战赛」 开幕式的压轴节目 「无人机方阵」中，每一架无人机展示一种灯光颜色。 无人机方阵通过两种操作进行颜色图案变换：

  (1)  调整无人机的位置布局

  (2)  切换无人机展示的灯光颜色

给定两个大小均为 *N\*M* 的二维数组 *source* 和 *target* 表示无人机方阵表演的两种颜色图案，由于无人机切换灯光颜色的耗能很大，请返回从 *source* 到 *target* 最少需要多少架无人机切换灯光颜色。

注意： 调整无人机的位置布局时无人机的位置可以随意变动。

#### 3.9.2概要设计

用一个nums数组储存每行每列不用颜色灯光出现的次数，然后在main函数里执行。

#### 3.9.3算法分析

算法说明:

先开一个值全部为0的nums数组，source方阵每行每列输入一个数字，其相应的序列号对应的数值加一， target 方阵每行每列输入一个数字，其相应的序列号对应的数值减一，最后进行遍历，输出数值不为0的下标值。

核心代码部分：

**int** nums[10002];

**int** main()

{

**int** m,n;

cin>>m>>n;

**int** num;

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

{

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

{

cin>>num;

nums[num]++;

}

}

**int** sum=0;

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

{

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

{

cin>>num;

**if**(nums[num]!=0)

{

nums[num]--;

}

**else**

{

sum++;

}

}

}

cout<<sum;

**return** 0;

}

**时间复杂度分析:**

两个循环都是*O(mn)*级别的，外层m次，内层n次，故时间复杂度就是*O(mn)*的。

#### 3.9.4使用说明

输入说明 :

输入若干行：

第一行为两个整数*n*和*m，n*代表二维数组的行数，*m*代表二维数组的列数。

而后*n*行，每行输入*m*个整数，代表*source*数组的元素。

再*n*行，每行输入*m*个整数，代表*target*数组的元素。

提示：



**运行界面如下:在xcode编译器的环境下可以正常运行**



#### 3.9.5测试结果与分析

(1)特殊情况，第一排方阵与第二排方阵相同，测试通过

IN:

5 5

1 1 1 1 1

1 1 1 1 1

1 1 1 1 1

1 1 1 1 1

1 1 1 1 1

1 1 1 1 1

1 1 1 1 1

1 1 1 1 1

1 1 1 1 1

1 1 1 1 1

OUT:

0

(2)特殊情况，第一个方阵和第二个方阵的元素均不相等，测试通过

IN：

4 4

3 7 1 4

2 8 9 5

6 12 33 11

10 14 15 88

13 17 21 29

35 37 24 59

66 39 28 94

62 64 67 55

OUT：

16

(3)特殊用例，不是方阵，测试通过

IN：

3 4

13 6 8 7

13 16 10 9

79 52 59 98

75 2 3 4

9 10 17 18

23 24 25 26

OUT：

10

(4)普通用例，方阵中存在相同的元素,测试通过

IN：

5 5

1 2 7 4 5

9 2 3 6 5

3 2 4 3 10

1 1 1 1 7

2 6 7 6 8

4 3 2 1 9

7 6 3 1 8

9 8 7 7 10

4 6 8 7 2

3 2 1 2 6

OUT:

4

(5)特殊用例，方阵中行列均为1,测试通过

IN：

1 1

1

2

OUT:

1

(6)大数据测试，测试通过

IN:

OUT:

### 3.10完成一半题目

#### 3.10.1题目描述

有 *N* 位扣友参加了微软与力扣举办的「以扣会友」线下活动。主办方提供了道题目，整型数组 *questions* 中每个数字对应了每道题目所涉及的知识点类型。

若每位扣友选择不同的一题，请返回被选的 N 道题目至少包含多少种知识点类型。

#### 3.10.2概要设计

用一个vector数组存储不同题目，用一个nums数组存储不同题目的数量，设计一个**bool** cmp(**int** a,**int** b)比较函数对题目数量进行sort(nums,nums+10001,cmp)排序，最后在main函数中执行。

#### 3.10.3算法分析

**算法说明:**

储存不同题目的数量并从大到小排序，然后进行循环，从头开始选出n个题目，每增加一种题目则sum加一，最后输出sum。

核心代码如下:

**int** nums[10001];

**bool** cmp(**int** a,**int** b)

{

**return** a>b;

}

**int** main()

{

**int** n;

cin>>n;

**int** num;

vector<**int**>questions;

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

{

cin>>num;

questions.push\_back(num);

nums[num]++;

}

sort(nums,nums+10001,cmp);

**int** k=n/2;

**int** sum=0;

**int** start=0;

**while**(k>0)

{

sum++;

k=k-nums[start];

start++;

}

cout<<sum;

**return** 0;

}

**时间复杂度说明：**

sort排序的时间复杂度是固定常数故不予考虑，下面的while循环最多执行次，故总的时间复杂度为*O*()﻿。

#### 3.10.4使用说明

输入说明 :

输入两行：

第一行为一个整数*n*代表数组*questions*的长度，，*N*为扣友的数量。

第二行输入*n*个整数代表数组*questions*的元素。

提示：



**运行界面如下:在xcode编译器的环境下可以正常运行**

图形用户界面, 文本

中度可信度描述已自动生成

#### 3.10.5测试结果与分析

(1)特殊数据，数组有序且没有重复出现的元素,测试通过

IN：

8

8 7 6 5 4 3 2 1

OUT:

4

(2)特殊数据，数组所有元素均相同,测试通过

IN：

12

1 1 1 1 1 1 1 1 1 1 1 1

OUT:

1

(3)普通数据，数组无序且不存在重复出现的元素,测试通过

IN：

10

12 39 31 47 29 28 83 91 52 88

OUT:

5

(4)普通数据，数组无序且存在重复出现的元素,测试通过

IN：

14

3 2 3 8 9 4 7 6 13 22 98 66 13 27

OUT：

5

(5)普通数据，测试通过

IN:

10

1 8 27 37 3 8 1 2 6 9

OUT：

3

(6)大数据测试,考察时间复杂度,测试通过

IN:

200

35 18 17 58 4 23 64 85 95 66 47 48 48 20 62 44 1 33 93 74 31 34 52 52 37 70 61 36 50 68 20 46 58 12 9 97 52 80 76 54 29 20 89 71 77 17 50 68 55 63 16 68 6 73 87 84 35 90 86 47 42 37 26 3 60 88 8 86 77 50 81 41 53 42 16 21 73 32 69 12 17 0 42 76 45 86 18 67 18 11 58 56 60 4 97 92 44 92 21 35 93 91 4 65 98 27 92 53 30 50 81 63 91 56 47 82 27 88 45 99 88 9 69 71 76 26 33 72 7 55 87 95 2 81 47 97 99 47 54 65 10 74 61 32 53 99 35 97 55 74 94 10 59 12 52 67 30 36 90 42 22 58 65 85 39 3 4 44 35 95 1 7 10 52 44 70 25 65 5 13 86 74 93 42 100 90 17 92 1 71 39 2 41 88 84 84 93 47 69 80

OUT：

27

### 3.11 错误的集合

#### 3.11.1题目以及要求

集合 *s* 包含从 1 到 *n* 的整数，并且按升序排列。不幸的是，因为数据错误，导致集合里面某一个数字复制了成了集合里面的另外一个数字的值，导致集合 丢失了一个数字 并且有一个数字重复 。

给定一个数组 *nums* 代表集合 *S* 发生错误后的结果。

请你找出重复出现的整数，再找到丢失的整数。

#### 3.11.2概要设计

设计一个简单的for循环，在main函数中执行。

#### 3.11.3算法分析

**算法说明:**

遍历整个nums数组，将输入的num与i进行比较，发现不同输出num和i即可。

核心代码如下:

**int** main()

{

**int** n;

cin>>n;

**int** num;

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

{

cin>>num;

**if**(num!=i)

{

cout<<num<<" "<<i<<" ";

}

}

**return** 0;

}

**时间复杂度分析:**

最多执行n次，故时间复杂度为。

#### 3.11.4使用说明

输入说明 :

输入两行：

第一行为一个整数*n*代表数组*nums*的长度。

第二行输入*n*个整数代表数组*nums*的元素。

提示：





**运行界面如下:在xcode编译器的环境下可以正常运行**



#### 3.11.5测试结果与分析

(1)特殊情况，缺失的数字为1

IN:

8

4 2 3 4 5 6 7 8

OUT：

4 1

(2)特殊情况,缺失的数字在末端

IN:

10

1 2 3 4 5 6 7 8 9 9

OUT:

9 10

(3)普通情况，缺失的数字和重复的数字相邻

IN：

10

1 2 3 4 5 6 6 8 9 10

OUT:

6 7

(4)普通情况,缺失的数字和重复的数字不相邻

IN:

12

1 2 3 9 5 6 7 8 9 10 11 12

OUT:

9 4

(5)特殊情况，只有两个数字且相等

IN：

2

1 1

OUT:

1 2

(6) 大数据测试，测试通过

IN:

100

0 1 2 3 4 5 6 7 8 9 10 11 48 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99

OUT:

0 1 1 2 2 3 3 4 4 5 5 6 6 7 7 8 8 9 9 10 10 11 11 12 48 13 13 14 14 15 15 16 16 17 17 18 18 19 19 20 20 21 21 22 22 23 23 24 24 25 25 26 26 27 27 28 28 29 29 30 30 31 31 32 32 33 33 34 34 35 35 36 36 37 37 38 38 39 39 40 40 41 41 42 42 43 43 44 44 45 45 46 46 47 47 48 48 49 49 50 50 51 51 52 52 53 53 54 54 55 55 56 56 57 57 58 58 59 59 60 60 61 61 62 62 63 63 64 64 65 65 66 66 67 67 68 68 69 69 70 70 71 71 72 72 73 73 74 74 75 75 76 76 77 77 78 78 79 79 80 80 81 81 82 82 83 83 84 84 85 85 86 86 87 87 88 88 89 89 90 90 91 91 92 92 93 93 94 94 95 95 96 96 97 97 98 98 99 99 100

### 3.12至少是其他数字两倍的最大数

#### 3.12.1题目以及要求

给你一个整数数组 *nums* ，其中总是存在 唯一的 一个最大整数 。

请你找出数组中的最大元素并检查它是否 至少是数组中每个其他数字的两倍 。如果是，则返回 最大元素的下标 ，否则返回 -1 。

#### 3.12.2概要设计

用一个nums数组存储数字，设计一个for循环进行比较判断，在main函数中执行。

#### 3.12.3算法分析

**算法说明：**

先用for循环遍历所有数字找出最大值，然后再进行一次遍历，判断该最大值是否大于等于遍历到的数字的两倍，如果是则继续循环，直至结束返回最大元素的下标；如果不是则输出-1，退出循环。

核心代码部分:

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

{

**if**(max>=nums[i]\*2&&i!=index)

{

**continue**;

}

**if**(max<nums[i]\*2&&i!=index)

{

cout << "-1";

**return** 0;

}

}

**时间复杂度分析:**

两个循环都是*O(n)*的,故总的时间复杂度为*O(n)。*

#### 3.12.4使用说明

输入说明 :

输入两行：

第一行输入一个整数*n*表示数组*nums*的长度。

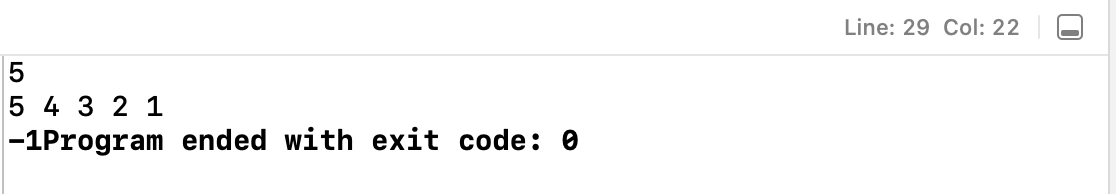
第二行输入*n*个整数表示数组的元素。

提示：



*nums* 中的最大元素是唯一的

**运行界面如下:在xcode编译器的环境下可以正常运行**



#### 3.12.5测试结果与分析

(1)特殊情况,仅一个元素且为1,测试正确

IN:

1

1

OUT:

0

(2)特殊情况，数组中的元素有序，测试正确

IN:

12

1 3 4 5 8 9 10 11 13 14 16 18

OUT:

-1

(3)普通情况，数组元素无序，有一个元素是其余元素的至少两倍，测试正确

IN:

10

13 22 4 3 17 36 32 8 19 100

OUT:

9

(4)普通情况，数组元素无序，没有一个元素是其余所有元素的两倍，测试正确

IN:

10

27 53 28 46 13 10 26 43 48 4

OUT:

-1

(5)普通数据，测试正确

IN:

20

34 50 57 66 27 76 35 28 57 58 15 22 14 34 93 11 48 23 57 87

OUT:

-1

(6)大数据测试，测试正确

IN:

198

12 100 5 95 89 32 70 10 97 74 72 11 26 46 56 18 44 42 99 23 94 41 22 68 52 53 11 74 56 27 30 36 69 96 17 42 2 44 47 71 97 16 58 40 91 15 22 78 64 94 18 54 41 63 3 60 61 91 15 43 6 49 72 61 60 22 95 56 75 51 83 45 99 32 91 25 55 57 47 34 46 8 81 81 15 21 14 72 65 32 40 70 79 18 39 93 94 61 80 82 16 17 36 42 33 82 99 98 61 49 69 58 40 91 34 31 83 73 25 0 22 85 51 96 4 26 59 66 85 36 16 42 20 58 79 11 90 48 0 36 21 50 91 45 80 72 19 98 93 20 19 9 75 84 63 20 49 57 78 79 7 37 44 68 86 9 1 90 90 33 44 35 2 55 13 93 8 76 45 5 41 75 5 29 49 81 78 24 51 28 62 36 50 0 56 98 64 28

OUT:

-1

### 3.13 K 次取反后最大化的数组和

#### 3.13.1题目以及要求

给你一个整数数组 *nums* 和一个整数 k ，按以下方法修改该数组：

选择某个下标 *i* 并将 *nums*[*i*] 替换为 *-nums*[*i*] 。

重复这个过程恰好 *k* 次,可以多次选择同一个下标 *i* 。

以这种方式修改数组后，返回数组 可能的最大和 。

#### 3.13.2概要设计

设计一个vector<**int**>nums存储数据，主代码在main函数中执行。

#### 3.13.3算法分析

**算法说明:**

对原数据从小到大进行sort(nums.begin(),nums.end())排序，在for循环中先对负数进行反转，然后再从最小的正数开始进行反转，再进行sort排序，返回数组的可能最大和。

核心代码部分:

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

{

**if**(nums[i]<0)

{

nums[i]=-nums[i];

k--;

}

**if**(k<=0)

{

**break**;

}

}

sort(nums.begin(),nums.end());

**if**(k==0)

{

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

{

sum=sum+nums[i];

}

cout<<sum;

}

**else** **if**(k>0)

{

**if**(k%2==0)

{

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

{

sum=sum+nums[i];

}

cout<<sum;

}

**else**

{

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

{

sum=sum+nums[i];

}

sum=sum-2\*nums[0];

cout<<sum;

}

}

**时间复杂度说明:**

第一个for循环时间复杂度为,第二个、第三个sort用时为，最后三个for循环根据条件执行一个，用时，取最高的时间复杂度。

#### 3.13.4使用说明

输入说明 :

输入三行：

第一行输入一个整数*n*表示数组*nums*的长度。

第二行输入*n*个整数表示数组的元素。

第三行输入一个整数表示*k*.

提示：

****

**运行界面如下:在xcode编译器的环境下可以正常运行**

图形用户界面, 文本

描述已自动生成

#### 3.13.5测试结果与分析

(1)简单数据，数组中有正数和负数，k为偶数，测试通过

IN:

2

1 -1

50

OUT:

0

(2)普通数据，数组中都是正数，k为奇数，测试通过

IN:

10

12 43 32 8 29 74 31 23 43 45

5

OUT:

324

(3)普通数据,数组中含有负数与正数，负数的个数小于*k*，*k*为偶数

IN:

12

-9 12 -3 -5 8 7 16 23 45 12 5 8

8

OUT:

147

(4)普通数据，数组中含有负数与正数，负数的个数小于*k*，*k*为奇数

IN:

20

-19 -24 8 27 39 -14 23 -71 52 66 8 -33 -16 20 29 37 88 54 20 26

15

OUT:

658

(5)普通数据，数组中含有负数与正数,负数的个数大于*k*，*k*为偶数

IN:

20

-99 -20 34 13 -17 8 -13 66 -17 -13 5 35 27 -18 -36 92 1 -8 10 -19

6

OUT:

44

(6)大数据测试，测试正确

IN:

400

4660 -1446 -5888 1953 3012 -2732 -4167 6607 -5078 2606 -1667 6547 -6277 4884 -3461 -6226 9734 6162 -4617 -4330 -5698 -9182 -1474 -5207 -7391 -7937 5134 8734 -9284 -4434 -3700 -7434 -5604 5095 919 7308 -2101 5822 1572 -1534 -1995 8999 2915 -9593 6960 597 3638 -6406 1739 -9690 -4215 -1114 -990 3392 -791 -9704 111 -2059 -7346 -140 7082 -9110 4291 9983 -4555 -4219 -7319 4649 7112 4930 -6980 1936 7872 -6820 -5319 -4000 4277 -8342 -819 1282 9637 8116 -5378 2439 -2037 -5312 -1601 -1356 8213 -3149 -3605 -5205 -9049 6999 2936 741 -5389 -9100 -1038 -8603 6405 8892 -1319 -4261 5445 1134 8970 -1835 -3997 1553 5403 -2547 3232 -4896 815 3973 8487 -6269 7301 -580 -4901 -429 -7014 -9994 -7827 -3 3947 8657 -6993 -8611 4430 -1411 -9774 7714 -9950 -2394 -3186 -8817 -5519 -9426 8760 -957 8056 -8820 -4004 -3268 -2581 7160 -1595 4835 3680 7303 -9761 -9059 -843 9283 -4204 -8578 5095 2307 745 -1380 8152 -6219 4277 -453 1877 3965 -9862 -7624 589 8033 -6588 1735 -8297 -5250 433 -5019 -7805 -1171 -8781 5141 -8647 -1102 6685 -1097 9432 3181 3513 9369 8029 -7055 6059 9358 -9669 -1954 -7101 -4784 -9446 -7245 1599 -3057 -4088 2663 -6141 -1855 -3090 7155 -8930 1671 -1246 -8659 280 991 -600 7890 -8801 -2338 -7075 7015 -1414 4099 -299 -8738 -1384 9585 1906 -7457 -6887 1205 -4531 -231 -6902 9638 -7409 9953 -2628 -9258 8714 -2162 -4319 -8176 3345 -4575 -9070 582 -1670 553 -8509 -8483 -9441 -375 -721 -5299 -8858 2094 7726 -7583 -3588 7887 8144 -258 -6932 -1332 311 7440 2860 -4341 8332 9183 8675 -3723 -5950 5363 4417 -1674 7247 -4924 -4637 -5114 1281 -4167 8900 7567 5250 -4754 1472 1087 2709 -7609 -3193 -5667 -2949 9357 3735 -7527 7823 -5109 9321 -4712 1751 -7949 -8972 -977 -5757 -8156 -661 5900 -7078 -9236 836 5233 2863 -3909 3629 6655 -6895 7427 -6972 -7159 6475 -6478 3689 -7606 5069 -8683 -546 -5067 5081 5801 -5679 2541 -1722 5969 9929 4365 4210 -4569 -4097 5311 -3077 -4230 -9757 7189 5056 -9695 5091 7531 -5379 -4984 1 9150 1135 -8699 -267 5405 2270 2756 4047 1352 -8641 -6131 4803 9773 -463 -6808 -2411 -6424 -1792 -7974 8223 5348 -9322 -5324 -9846 2281 394 726 -646 -8856 9904 -8755 8885 -8554 7880 5414 138 -9635 -1782 5780 5129 456 3594 -8286 -6572 8755 9446 6447 -1558 6811

200

OUT:

2035604

### 3.14检查整数及其两倍数是否存在

#### 3.14.1题目以及要求

给你一个整数数组 *arr*，请你检查是否存在两个整数 N 和 M，满足 N 是 M 的两倍(即，)。

更正式地，检查是否存在两个下标 *i* 和 *j* 满足：



#### 3.14.2概要设计

设计一个nums[10001]存储数据，主代码在main函数中执行。

#### 3.14.3算法分析

**算法说明:**

先用for循环遍历所有num，将nums[num+1000]赋值为1，然后再次进行循环遍历，不断进行判断，如果nums[i+1000]==1&&nums[2\*i+1000]==1，则输出true，退出循环；如果遍历完都没有找到则输出false。

核心代码部分：

**for**(**int** i=0;i<4000;i++)

{

**if**(nums[i+1000]==1&&nums[2\*i+1000]==1)

{

cout<<"true"<<endl;

**return** 0;

}

}

cout<<"false";

**时间复杂度分析:**

nums哈希的时间复杂度是，后面查找的时间复杂度是*O(4000)*，常数忽略不计，故时间复杂度为

#### 3.14.4使用说明

输入说明 :

输入两行：

第一行输入一个整数*n*表示数组*arr*的长度。

第二行输入*n*个整数表示数组的元素。

提示：



**运行界面如下:在Xcode编译器的环境下可以正常运行**

图表

描述已自动生成

#### 3.14.5测试结果与分析

(1)只有两个元素，测试通过

IN:

2

1 2

OUT:

true

(2)数据含有负数,元素有序排列,测试通过

IN:

10

-22 -15 -13 -7 4 8 12 15 37 60

OUT:

true

(3)数据集含有负数,随机排列，测试通过

IN:

10

-4 6 10 7 8 -8 -12 7 16 9

OUT:

True

(4)所有数据都为0

IN:

20

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

OUT:

True

(5)随机数据

IN:

8

-9 3 -2 1 10 19 23 55

OUT:

false

(6)大数据测试，测试正确

IN:

400

5240 -8329 -6011 -2296 -5396 -6500 3349 300 -7900 2430 -3236 1024 7934 -148 -4881 -1770 -3409 1070 2355 -5914 4470 293 2369 -8941 -4063 -1093 2741 -2380 2567 -2187 -4250 -2545 -2189 6986 -453 1722 -4032 -3047 -1201 -230 -134 6566 4432 -1904 8534 -9636 1789 -7556 -5727 3365 1515 -9645 4507 -8748 -934 4876 3600 5985 8389 -2309 -7104 629 -6587 1512 4838 -8260 7836 -1034 -6032 -3906 7113 -6162 -9581 6246 5732 8697 -2967 8994 5901 -1016 -5869 -4932 -1796 -357 -6002 5839 994 -4058 -2408 -5304 1791 1040 -3432 9498 -6600 -7583 47 -3790 3848 9834 3266 4428 -3032 1143 4599 1330 9493 -9509 -9196 8660 4840 121 5891 -993 -5612 9626 383 8545 56 6821 -3828 -5343 -9831 3862 7504 9289 9621 -2385 9738 -8218 9355 8598 -6542 9628 8750 -8860 -2648 4801 5887 1023 4827 -3718 -208 -4782 4745 -5909 -6504 -4456 -4827 -455 9654 -8517 -5745 4022 3240 2996 6853 2088 6468 -4785 571 953 -3319 -5753 -8209 325 7868 -1739 910 -4383 -4003 7979 -8807 -6509 2398 7613 -7881 -9607 -43 2205 6874 6493 5217 -3679 6599 3024 -6817 -966 -5758 -7014 282 -1323 -2092 -1641 -8324 -1452 8575 920 1350 7072 -5953 -283 1398 2514 3846 9600 7804 -1590 1224 -8452 -3637 2450 6483 7994 1725 5628 6643 9887 -117 4602 3597 7829 8851 5172 3882 3270 -5461 3828 5726 -3871 5847 7495 9377 -1954 -5887 6231 5303 6911 8446 5596 1733 790 1601 2606 6361 1129 -3681 -2081 -5568 -444 -9964 -7513 9105 -1935 2888 -2800 3874 6104 -7378 9916 -4428 9087 -9288 -3260 1240 -4202 1895 -7577 -9628 6382 -4433 -7952 -4451 -3412 1872 -6282 4429 -1797 -4685 -8587 -1825 -3880 -5276 6903 8857 -8957 5814 4149 9962 1168 -8266 8799 -7924 4796 6915 -437 2427 -1090 -7343 -2681 -1385 5726 -2414 -7274 -5696 4936 -8531 -618 6349 -5286 -3927 -8942 4624 -5599 5108 -9345 2367 6390 5216 -5635 -9525 -4450 1313 -9197 -5012 2744 -6541 -8344 -362 5512 -5552 -2849 229 1167 7939 -1991 -5440 -4721 9344 -3617 8824 3709 -1265 -594 -8040 -3899 -5678 9327 4197 4249 -5897 5293 9645 -9548 1109 -9845 7336 -3787 812 3394 4494 7110 9195 -4613 -3285 1789 7831 1599 -2970 -8372 1327 -8223 7799 5837 -5823 5090 -2214 -3748 1227 -4686 4488 -8565 -1227 3566 6167 -1143 -4413 -8656 2006 8987 -4079 -8583 -7090 -1392 -4936 -9979 7496 7375 -3236 600

OUT:

true

### 3.15数据流的第 K 大数值

#### 3.15.1题目以及要求

设计一个找到数据流中第 *k* 大元素的类(*class*)。注意是排序后的第 *k* 大元素，不是第 *k* 个不同的元素。

请实现 *KthLargest* 类：

*KthLargest(int k, int*[] *nums)* 使用整数 *k* 和整数流 *nums* 初始化对象。

*int add(int val)* 将 *val* 插入数据流 *nums* 后，返回当前数据流中第 *k* 大的元素。

#### 3.15.2概要设计

用priority\_queue<**int**,vector<**int**>,greater<**int**>>nums储存数据并自动排序，在main函数中执行。

#### 3.15.3算法分析

**算法说明:**

用优先队列存储前k大的元素，并保证最小的元素即第k大的元素在队头，在输入 add 指令时先判断输入的整数val是否比队头元素大，如果大于队头元素则先将队头元素弹出再将新元素进行排序，输出现在的队头元素即第k大元素。

priority\_queue<**int**,vector<**int**>,greater<**int**>>nums;

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

{

cin>>num;

nums.push(num);

**if**(nums.size()>k)

{

nums.pop();

}

}

**while**(cin>>s)

{

cin>>add;

nums.push(add);

**if**(nums.size()>k)

{

nums.pop();

}

cout<<nums.top()<<endl;

}

**时间复杂度说明**:

使用堆这种数据结构初始化，时间复杂度，总的时间复杂度为*O(logk)*。

#### 3.15.4使用说明

输入说明 :

输入若干行：

首行输入两个整数*k*和*n*，*k*代表题中的*k*，*n*代表用来初始化对象的*nums*数组的长度。

第二行输入*n*个整数代表用来初始化对象的数组*nums*的元素。

后面若干行输入 *add* 指令，每个*add*后面跟一个整数表示*val*。

提示：



    最多调用 add 方法  次

    题目数据保证，在查找第 *k* 大元素时，数组中至少有 *k* 个元素

**运行界面如下:在xcode编译器的环境下可以正常运行**

图形用户界面, 应用程序

描述已自动生成

#### 3.15.5测试结果与分析

(1)特殊情况，k=1，n=0，测试通过

IN:

1 0

add 1

add 2

add 3

add 4

add 2

add 1

OUT:

1

2

3

4

4

4

(2)特殊情况，k=1，n测试通过

IN:

1 6

4 6 13 7 1 5

add 7

add 10

add -9

add 10

add 39

add 77

OUT:

13

13

13

13

39

77

(3)普通情况，输入的元素有序，测试通过

IN:

3 4

7 9 13 21

add 17

add 29

add 44

add 56

add 68

add 72

add 99

OUT:

13

17

21

29

44

56

68

(4)普通情况，元素无序且含有负数，测试通过

IN:

6 10

-13 -35 -22 197 23 542 24 29 -97 24

add 7

add 1

add 9

add -14

add -83

add 68

add -13

add 24

OUT:

23

23

23

23

23

24

24

24

(5)普通情况,元素无序且含有负数,测试通过

IN:

9 20

-8 69 33 43 85 98 -29 -41 -34 39 -63 -5 78 -19 61 87 391 393 90 12

add 10

add 23

add 12

add -45

add -23

add -78

add 14

add 10

add 234

add 481

OUT：

61

61

61

61

61

61

61

61

69

78