初赛 DAY 3

1. 排序算法

排序算法的作用是快速的让一个序列从小到大排列。

1.1 应用

1. 查看第 k 大值
2. 去重
3. 查看是否是一个 1-n 的排列
4. 求逆序对（区间和大于 0、小于 0）

1.2 冒泡排序

冒泡排序（Bubble sorting）每次逐渐的，将未排序的数据中最大的数据，交换到最后位置上。

算法的具体步骤是：

1. **比较相邻数据**：从序列的第一个数据开始，比较相邻的两个数据。
2. **交换位置**：如果前一个数据比后一个数据大，则交换它们的位置。
3. **重复遍历**：对序列中的每一对相邻数据重复上述步骤，直到列表的末尾。这样，最大的数据会被“冒泡” 到列表的最后。
4. **缩小范围**：忽略已经排序好的最后一个数据，重复上述步骤，直到整个列表排序完成。

【示例程序】以数字从小到大排序为例。

代码块

```
for(int i = 1; i <= n - 1; i++) {  //第 i 遍让前 n - i + 1 个数据归位
    for(int j = 1; j <= n - i; j++) {
        if(a[j + 1] < a[j]) {
            swap(a[j],a[j + 1]);
        }
    }
}
```

时间复杂度： $O(n^2)$。

空间复杂度： $O(1)$。

1.3 选择排序

选择排序（Select Sorting）每次选择出未排序的数据中最小的数据，交换到第一个位置上。

算法的具体步骤是：

1. **初始化**：将列表分为已排序部分和未排序部分。初始时，已排序部分为空，未排序部分为整个列表。
2. **查找最小值**：在未排序部分中查找最小的数据。
3. **交换位置**：将找到的最小数据与未排序部分的第一个数据交换位置。
4. **更新范围**：将未排序部分的起始位置向后移动一位，扩大已排序部分的范围。
5. **重复步骤**：重复上述步骤，直到未排序部分为空，列表完全有序。

【示例程序】以数字从小到大排序为例。

代码块

```
for(int i = 1; i <= n - 1; i++) { //重复 n - 1 次
    int id = i;
    for(int j = i; j <= n; j++) {//第 i 次从 i ~ n 选出最大或者最小的数字
        if(a[id] > a[j]) {
            id = j;
        }
    }
    swap(a[i],a[id]);
}
```

时间复杂度： $O(n^2)$。

空间复杂度： $O(1)$。

1.4 插入排序

插入排序（Select Sorting）每次将一个数据插入到已排序的序列中。

算法的具体步骤是：

1. **初始化**：将列表分为已排序部分和未排序部分。初始时，已排序部分只包含第一个元素，未排序部分包含剩余元素。
2. **选择元素**：从未排序部分中取出第一个元素。
3. **插入到已排序部分**：将该元素与已排序部分的元素从后向前依次比较，找到合适的位置插入。
4. **重复步骤**：重复上述步骤，直到未排序部分为空，列表完全有序。

【示例程序】以数字从小到大排序为例。

```
for(int i = 2; i <= n; i++) {
    for(int j = i - 1; j >= 1; j--) {
        if(a[j + 1] < a[j]) {
            swap(a[j],a[j + 1]);
        } else break;
    }
}
```

时间复杂度：

当序列是从小到大排列的情况，排序将达到最优复杂度： $O(n)$ 。

当序列是从大到小排列的情况，排序将达到最坏复杂度： $O(n^2)$ 。

平均复杂： $O(n^2)$ 。

1.5 计数排序

当需要排列的数据分布在一个小范围内时，直观上可以通过计数来进行有效排序。因为我们知道 “数据小” 的元素会自然地排在 “数据大” 的元素前面，所以我们只需要知道这个范围内的每个数据出现了多少次。例如，要给整数序列 `1 2 1 2 2 3 1 3 3` 从小到大排列，此处的数字小的自然的就要排在数字大的前面。所以，排序的结果一定是 1 在最前，其次是 2 ，最后是 3。于是，我们只需要知道 1、2、3 的个数便可完成排序。这被称为**计数排序（count sorting）**。

【示例程序】以数字从小到大排序为例。

代码块

```cpp
#include<iostream>
using namespace std;
const int N = 1e7 + 10;
int a[N],b[N],c[N]; //b[i] 是 i 这个数字出现的个数
int main() {
    int n;
    cin >> n;
    for(int i = 1; i <= n; i++){
        scanf("%d",&a[i]);
        b[a[i]]++;
    }
    int sum = 0;
    for(int i = 1; i <= 1e7; i++){
        for(int j = 1; j <= b[i]; j++){
            c[++sum] = i; // 记录结果
        }
    }
    return 0;
}
```

时间复杂度： $O(n+w)$ //w是值域

空间复杂度： $O(w)$

1.6 归并排序

分治思想。

将 1 - n 的排序分为 1 - n/2 和 n/2 - n 的排序，最后合并结果。

代码块

```cpp
#include <bits/stdc++.h>
using namespace std;

#define int long long
const int N = 550000;
int a[N],c[N];

void merge_sort(int l,int r){
    // l ~ r 排序
    if(l == r){
        return;
    }
    int sum = 0;
    int mid = (l + r) / 2;
    sum += merge_sort(l,mid);
```

```
16        sum += merge_sort(mid + 1,r);
17        int s = l,t =  mid + 1;
18        int p = l;
19        while(s <= mid && t <= r){
20            if(a[s] <= a[t]){
21                c[p] = a[s];
22                p++;
23                s++;
24            }else{
25                c[p] = a[t];
26                p++;
27                t++;
28            }
29        }
30
31        while(s <= mid){
32            c[p] = a[s];
33            p++;
34            s++;
35        }
36
37        while(t <= r){
38            c[p] = a[t];
39            p++;
40            t++;
41        }
42
43        for(int i = l; i <= r; i++){
44            a[i] = c[i];
45        }
46    }
47
48
```

1.7 稳定性

稳定性是指相同的数据经过排序之后相对顺序是否发生了改变。例如，数字序列 `4(1) 4(2) 2` ，排序后变为 `2 4(1) 4(2)` ，那么此排序就是稳定排序。若排序后变为 `2 4(2) 4(1)` ，那么此排序就是不稳定排序。

目前学习的，计数排序、插入排序、冒泡排序是稳定排序。

1.8 逆序对

在一个排列中，如果某一个较大的数据排在某一个较小的数据前面，就说这两个数据构成一个 **逆序（inversion）**或反序。

在一个排列里出现的逆序的总个数，叫做**逆序数**。排列的逆序数是它恢复成正序序列所需要做相邻对换的最少次数。因为，交换相邻的一对逆序，会减少一个逆序数。而冒泡排序、插入排序都是通过不断的交换相邻数据来完成排序，于是都可以用于求出逆序数。

代码块

```
for(int i = 1; i <= n - 1; i++) {  //第 i 遍让前 n - i + 1 个数据归位
    for(int j = 1; j <= n - i; j++) {
        if(a[j + 1] < a[j]) {
            swap(a[j],a[j + 1]);
            sum++;
        }
    }
}
```

在归并排序的过程中也可以顺便求出逆序对数量。

序列 1 - n 的逆序对可以分为，序列 1 - n / 2 的逆序对数量 + n / 2 - n 的逆序对数量 + 第一个数在 1 - n / 2，第二个数在 n / 2 - n 的逆序对数量。

代码块

```
#include <bits/stdc++.h>
using namespace std;

#define int long long
const int N = 550000;
int a[N],c[N];

int merge_sort(int l,int r){
    // l ~ r 排序
    if(l == r){
        return 0;
    }
    int sum = 0;
    int mid = (l + r) / 2;
    sum += merge_sort(l,mid);
    sum += merge_sort(mid + 1,r);
    int s = l,t =  mid + 1;
    int p = l;
    while(s <= mid && t <= r){
        if(a[s] <= a[t]){
            c[p] = a[s];
            p++;
            s++;
        }else{
            c[p] = a[t];
```

```
26                 p++;
27                 t++;
28                 sum += (mid - s + 1);
29             }
30         }
31 
32         while(s <= mid){
33             c[p] = a[s];
34             p++;
35             s++;
36         }
37 
38         while(t <= r){
39             c[p] = a[t];
40             p++;
41             t++;
42         }
43 
44         for(int i = l; i <= r; i++){
45             a[i] = c[i];
46         }
47         return sum;
48     }
```

1.9 选择题

1. 以下哪个排序是稳定的？（ ）

A. 计数排序 B.选择排序 C.希尔排序 D.快速排序

2. 以下哪个排序在运行 1/3 时间之后无法获得最大值？（ ）

A. 选择 B.冒泡 C.都错误 D.插入

3. 以下哪个排序无需进行比较？（ ）

A. 选择 B.冒泡 C.计数 D.插入

4. 以下哪个选项是错误的（ ）

A. 插入排序最好的时间复杂度为 $O(n)$。

B. 插入排序最坏的时间复杂度为 $O(n^2)$。

C. 插入排序也可以用于求解逆序对的数量

D. 插入排序的原理是将最大值插入到有序序列中

5. 序列 [2,5,6,1,7,3] 进行一趟排序后，以下哪个说法错误（ ）

A. 经过一趟选择排序 [1,5,6,2,7,3]

B. 经过一趟插入排序 [2,7,5,6,1,3]

C. 经过一趟冒泡排序 [2,5,1,6,3,7]

D. 经过一趟选择排序 [2,5,6,1,3,7]

1.10 阅读程序

代码块

```
#include<bits/stdc++.h>
using namespace std;
const int maxn = 500000, INF = 0x3f3f3f3f;
int L[maxn / 2 + 2], R[maxn / 2 + 2];
void unknown(int a[], int n, int left, int mid, int right) {
    int n1 = mid - left, n2 = right - mid;
    for(int i = 0; i < n1; i++)
        L[i] = a[left + i];
    for(int i = 0; i < n2; i++)
        R[i] = a[mid + i];
    L[n1] = R[n2] = INF;
    int i = 0, j = 0;
    for(int k = left; k < right; k++) {
        if(L[i] <= R[j])
            a[k] = L[i++];
        else
            a[k] = R[j++];
    }
}
void unknownsort(int a[], int n, int left, int right) {
    if(left + 1 < right) {
        int mid = (left + right) / 2;
        unknownsort(a, n, left, mid);
        unknownsort(a, n, mid, right);
        unknown(a, n, left, mid, right);
    }
}
int main() {
    int a[maxn], n;
    cin >> n;
    for(int i = 0; i < n; i++) cin >> a[i];
    unknownsort(a, n, 0, n);
    for(int i = 0; i < n; i++) {
        if(i) cout << " ";
        cout << a[i];
    }
    cout << endl;
    return 0;
```

```
39   }
```

判断题

（1）将第13行的“<”改为“<=”不会改变运行结果。

（2）将第21行的“<”改为“<=”不会改变运行结果。

（3）此类排序方法是高效的，但是不稳定。

（4）将第4行的2个“+2”都去掉不会改变运行结果。

选择题

（5）此题是哪种排序?

A. 选择排序

B. 桶排序

C. 归并排序

D. 堆排序

（6）（4分）此题用到了（　）思想。

A. 动态规划

B. 分治

C. 冒泡

D. 贪心

2.二分查找算法

二分查找（Binary Search）算法是一种优化枚举查找的方法。它让我们重复选取枚举范围中心的对象进行判断，根据判断反馈，排除掉一半的积累对象，直到枚举范围内没有积累对象，仅需判断约 $\log n$ 次。

2.1 单调性

枚举能被二分优化的核心前提是**枚举序列必须具有单调性**。它指的是：在枚举范围内，存在一个明确的临界点，使得在该点之前的所有对象对于某个判断条件 O 均给出相同的结果（例如，全为“真”），而该点之后的所有对象对于条件 O 均给出相反的结果（例如全为 “假”）。

简而言之，整个序列对于条件 O 的判断结果，呈现出清晰的 `[假，假，...，假，真，真，...，真]` 或 `[真，真，...，真，假，假，...，假]` 模式。正是这种单调性，使得我们可以根据反馈向结果方向不断靠拢。

例如，在一个从小到大排列的 100 个数字的序列中，查找第一个大于或等于数字 x 的位置。

判断条件 O 是 “大于或等于数字 x ”。因为，这个序列是从小到大排列的，所以呈现 `[假，假，...，假，真，真，...，真]` 的模式，满足单调性。于是，可以二分优化。

2.2 实现

假设序列对于条件 `O` 的判断结果呈现为 `[假，假，...，假，真，真，...，真]` 的单调特性，要查找的是第一个真所在的位置。二分查找的实现主要有三种思路：

2.2.1 闭区间

左边界为未查看的元素的左端点，右边界为未查看的元素的右端点。

【示例程序】

代码块

```
int l = minn,r = maxn,ans = -1; // l 是枚举的最小范围，r 是枚举的最大范围 ans 记录答案
while(l <= r){ // l > r 则没有数字
    int mid = (l + r) / 2;
    if(条件为真) ans = mid,r = mid - 1;
    else l = mid + 1;
}
```

2.2.2 单闭区间

左边界为未查看的元素的左端点，右边界为未查看的元素的右端点的后一个。

此方法不单独记录答案，而是始终将满足条件的对象保留在枚举范围内，通过不断收缩二分范围 `[l，r]`，直到 `l` 与 `r` 相遇，最终的相遇点即为答案。

【示例程序】

代码块

```
int l = minn,r = maxn; // l 是枚举的最小范围，r 是枚举的最大范围
while(l < r){ // 保留一个对象
    int mid = (l + r) / 2;
    if(条件为真) r = mid; // 把 mid 保留在内
    else l = mid + 1;
}
```

【查找最后一个为假的示例程序】

代码块

```
int l = minn,r = maxn; // l 是枚举的最小范围，r 是枚举的最大范围
while(l < r){ // 保留一个对象
    int mid = (l + r + 1) / 2;
    if(条件为真) r = mid - 1;
    else l = mid;
}
```

关键细节：中点取整方式的选择

当边界更新策略包含 `l = mid` 时，必须使用 `mid = (l + r + 1) / 2`（向上取整）来计算中点。这是为了防止在 `l` 和 `r` 相邻时，`mid` 因向下取整等于 `l`，进而导致 `l = mid` 无法缩小二分范围，形成死循环。向上取整能确保在此情况下 `mid` 等于 `r`，从而使边界得以有效移动。

2.2.3 开区间

左边界为未查看的元素的左端点的上一个，右边界为未查看的元素的右端点的后一个。

代码块

```
while(l < r - 1){
    mid = (l + r) / 2;
    if(last[mid] <= ch){
        l = mid;
    }else r = mid;
}
```

2.3 解决单调性

- 寻找 2 ~ n 中第一个大于 `x` 的质数。

判断条件 O 是 “大于 x && 是质数”，该条件显然不满足单调性，无法直接进行二分。

缘由在于 “是质数” 不满足单调性。所以，我们可以先把所有 2 - n 的质数筛选出来，这样是质数的条件一定满足，这样条件 O 的结果只与 “大于 x” 相关，满足单调性，可以二分。

2.4 阅读程序

一、

代码块

```
#include <iostream>
#include <cstdio>
#include <algorithm>

using namespace std;

```

```cpp
const int N = 1e5 + 10;
int n, s, cnt, ans, res;
int a[N], b[N];

bool check(int mid) {
    for (int i = 1; i <= n; i++)
        b[i] = a[i] + i * mid;
    sort(b + 1, b + 1 + n);
    res = 0;
    for (int i = 1; i <= mid && res <= s; i++)
        res += b[i];
    return res <= s;
}

int main() {
    scanf("%d%d", &n, &s);
    for (int i = 1; i <= n; i++)
        scanf("%d", &a[i]);
    int l = 0, r = n;
    while (l <= r) {
        int mid = (l + r) >> 1;
        if (check(mid)) cnt = mid, ans = res, l = mid + 1;
        else r = mid - 1;
    }
    printf("%d %d\n", cnt, ans);
    return 0;
}
```

判断题

1. 若输入 `4 100 1 2 5 6`，则程序的输出为 `4 54`。 （ ）

【考点分析】√ 代值法。直接代入 mid = 4 的情况。

2. 对于任意的输入， `cnt` 的一个必定合法的取值为 `n`。 （ ）

【考点分析】× 只能说有可能。

3. 这个程序的时间复杂度为 `O(nlogn)`。 （ ）

【考点分析】× $O(n \log^2 n)$

选择题

4. 当输入为 `3 11 2 3 5` 时，程序的输出为（）。
   - A. `1 11`
   - B. `2 11`
   - C. `3 8`

- D. `0 0`

【考点分析】B 代值法。从最大的情况开始代入。

5. 代码中 `check` 函数的作用是什么？（）
   - A. 判断当前数组是否有序
   - B. 检查是否能从数组中选出 `mid` 个数，使得它们的总和小于或等于 `s`
   - C. 判断数组的所有元素是否大于某个值
   - D. 计算数组元素的平均值

【考点分析】B

6. （4分）变量 `cnt` 和 `ans` 的作用分别是什么？（）
   - A. `cnt` 记录满足条件的最大 `mid` 值， `ans` 记录对应的总和
   - B. `cnt` 记录数组的长度， `ans` 记录数组中的最大值
   - C. `cnt` 表示排序后的最小值索引， `ans` 记录当前结果的最小值
   - D. `cnt` 表示满足条件的元素个数， `ans` 记录最终的目标值

【考点分析】A

二、

代码块

```
#include<bits/stdc++.h>
using namespace std;
int n, m,k,l,r,mid;
int check(int g) {
    int st=1,ed=m,cnt=0;
    while(st<=n && ed>=1) {
        if(st*ed>g)
            ed--;
        else {
            cnt+=ed;
            st++;
        }
    }
    return cnt>=k;
}
int main() {
    scanf("%d%d%d",&n,&m,&k);
    l=1,r=n*m;
    while(l<r) {
        mid=(l+r)/2;
        if(check(mid))r=mid;
```

```
22            else l=mid+1;
23        }
24        cout<<l<<endl;
25        return 0;
26    }
```

已知 `k <= n*m`，保证 `n`， `m` 同阶，完成以下问题。

- 判断题

21. 每次运行 `check` 时，第7行必定运行 `n` 次。（）

【考点分析】 ×。有可能运行 n 次，也有可能运行 m 次。

22. 如果保证 `m=1`，则输出一定为 `k`。（）

【考点分析】 ×。 需要注意，题目保证了 `k <= n * m`。所以， `l` ~ `r` 的范围内包含了 `k`。

23. 若将第21行中的 `r=mid` 改成 `r=mid-1`，程序输出一定不变。（）

【考点分析】 ×。 答案未被包含在内。

24. 第24行可以改成 `cout<<r<<endl;`。（）

【考点分析】 √。此二分方法最后 `l`、 `r` 相等。

- 选择题

25. 该程序的时间复杂度为（）

  - A. O(n)
  - B. O(nlogn)
  - C. O(n^2)
  - D. O(nk)

【考点分析】 B。

26. 若输入为 `2 3 4`，则输出为（）

  - A. 1
  - B. 2
  - C. 3
  - D. 6

【考点分析】 C。 代值法。

三、

代码块

```cpp
#include<bits/stdc++.h>

using namespace std;

char last[101],ch;
int main(){
    int n,length = 0;
    scanf("%d",&n);
    for(int i = 0; i <= n; i++){
        cin >> ch;
        if(ch >= last[length]){
            last[++length] = ch;
        }else if(ch < last[1]){
            last[1] = ch;
        }else{
            int l = 1,r = length,mid;
            while(l < r - 1){
                mid = (l + r) / 2;
                if(last[mid] <= ch){
                    l = mid;
                }else r = mid;
            }
            last[r] = ch;
        }
    }
    printf("%d\n",length);
    return 0;
}
```

输入的只可能是小写字母。

判断题

1) 要使输出等于 n，输入的所有字符必须满足第 i 个字符ASCII整数值小于第 $i+1$ 个字符的 ASCII 整数值 $1 \leq i \leq n-1$ (　　)

【考点分析】 ×。 最后输出的是 length，而能让 length 增加的只有第 12 行。此时，只有当当前字符比上一个字符要**大于等于**，才会增加。

2) 若输入字符个数超过 101 个，一定会发生数组下标溢出。

【考点分析】√。当输入的字符超过 101 个，n 最小是 100。此时，第 12 行最大会访问 last[n + 1]。

3) 若将第 18 行替换为 $mid = l + ((r-l) >> 1)$，程序运行结果不会改变。( )

【考点分析】√。这样写是为了防止溢出。

4) 程序运行过程中，变量 r 的值可能会等于 n。

【考点分析】√。前 n 个字符满足不递减时，长度变为 n。最后一个字符，小于上一个字符，且大于等于第一个字符时出现 `r = n` 的情况。

5) 如果 11 行修改为 `ch > last[length]` 最终输出的结果可能会有 30 (　)

【考点分析】×。没有可能，小写字母只有 26 个。

单选题

1. 第 23 行 `last[r] = ch` 最多会运行 (　) 次。

A. 1　B. n-2　C. n-1　D. n

【考点分析】C。当 length 等于 2 的时候才有可能出现第三种情况。当 n = 2，输入三个字符。此时达成运行 n-1 次。

2. 若输入 $n = 100$，且第 17 行的 while 循环运行了至少一次，那么输出结果不可能是 (　)

A. 2　B. 3　C. 98　D. 100

【考点分析】D。显然。

3. 若输入数据为 4 b c d a e，则输出是 (　)

A. 4　B. 2　C. 5　D. 3

【考点分析】A。显然。

2.5 完善程序

一、

（选择数对）有 n 个小朋友分别拿着一个数字，现在要求你将小朋友两两配对，要求每对小朋友手上的数字之差大于给定值 k。问最多可以从数组中选出多少对符合条件小朋友。（每个小朋友最多只能和另外一个小朋友配对，不能脚踏两条船）

输入格式：第一行给出 n，k 分别表示小朋友的个数，以及参数 k。$(1 \leq n, k \leq 10^5)$，接下来给出 n 个数字，表示每个小朋友手上的数字。

提示：二分最终的结果，对于二分的结果 mid，在 check 函数中 $O(n)$ 判断能否选出 mid 对小朋友符合要求，最终保留二分的结果。

代码块

```
#include <iostream>
#include <algorithm>
using namespace std;
const int maxn = 1e5 + 5;
int a[maxn], n, k;
bool check(int mid) { // 要配对 mid 队
    bool flag = true;
    for(int i = 1, **1**; j <= n; j++, i++) {
        if(**2**)
```

```
10              flag = false; //不满足要求
11      }
12      return flag;
13  }
14  int main() {
15      cin >> n >> k;
16      for(int i = 1; i <= n; i++)
17          cin >> a[i];
18      int l = **3**, r = n, ans;
19      sort(a+1, a+1+n);
20      while(l <= r) {
21          int mid = (l + r) / 2;
22          if(check(mid)) {
23              **4**;
24              ans = mid;
25          } else {
26              **5**;
27          }
28      }
29      cout << ans << endl;
30  }
```

1. 1 处应该填的代码是:

A. j = 1　　B. j = n - mid　　C. j = n - mid + 1　　D. j = mid

2. 2 处应该填的代码是:

A.a[i] - a[j] > k　　B. a[i] - a[j] <= k　　C. a[j] - a[i] > k　　D. a[j] - a[i] <= k

3. 3 处应该填的代码是:

A. 1　　B. 0　　C. n/2　　D. a[1]

4. 4 处应该填的代码是:

A. l = mid　　B. r = mid　　C. l = mid + 1　　D. r = mid - 1

5. 5 处应该填的代码是:

A. l = mid　　B. r = mid　　C. l = mid + 1　　D. r = mid - 1

【要点分析】

答案为：CDBCD

通过 24 行可知，当判断为真，记录了答案。也就是说，当判断为真，当前数量的小朋友满足情况。那么第四空能够选出 C，第五空能选出 D。第二空就能选出来选 D。同时，这应该是个贪心的过程，拿最小的小朋友和尽量大的小朋友比较。于是第一空选 C。

由于，ans 没有初始值，如果一对小朋友都不满足，那么输出即为随机，所以要将 l 设为 0，包含一对都不满足的情况。

二、

（插入排序）对一段长度为 n 的数组使用插入排序的算法进行从小到大排序。

提示：插入排序共有 n 轮，第 i 轮时使用二分查找找到第 i 个数字应该插入的位置，然后将 pos 到 i 位置向右移动，再将数字 i 放入 pos 位置。保持数组下标 $1 \sim i$ 的元素有序，继续进行下一轮插入排序。

代码块

```
#include <iostream>
using namespace std;
const int maxn = 1000 + 5;
int a[maxn];
int main() {
    int n;
    cin >> n;
    for(int i = 1; i <= n; i++) {
        cin >> a[i];
    }
    for(int i = 1; i <= n; i++) {
        int temp = a[i];
        int l = 1, r = **1**, pos; // 确定二分法的上界和下界
        while(l <= r) {
            int mid = (l + r) / 2;
            if(**2**) {
                l = mid + 1;
                pos = mid;
            } else {
                r = mid - 1;
            }
        }
        **3** { // 向右移动，腾出位置
            a[j] = a[j-1];
        }
        a[pos] = **4**;
    }
    for(int i = 1; i <= n; i++)
        cout << a[i] << endl;
}
```

1. 1 处应填

A. n　　B. i　　C. i-1　　D. i+1

2. 2 处应填

A. mid <= a[i]　　B. mid >= a[i]　　C. a[mid] <= a[i]　　D. a[mid] >= temp

3. 3 处应填

A. for(int j = pos; j <= i-1; j++)

B. for(int j = i-1; j >= pos; j--)

C. for(int j = pos; j <= i; j++)

D. for(int j = i; j >= pos; j--)

4. 4 处应填

A. a[i]　　B. temp　　C. a[l]　　D. a[r]

5. 该算法的平均时间复杂度和最坏时间复杂度分别为

A. $O(n^2), O(n^2)$　　B. $O(nlogn), O(nlogn)$　　C. $O(nlogn), O(n^2)$　　D. $O(n\sqrt{n}), O(n^2)$

【要点分析】

CCDBA

根据题目所说，保证 1 ~ i 有序，于是要将第 i 个数字插入至前 $i-1$ 个数字中。

第二空，记录了答案，并且继续往右找，说明当前的数字小了或者等于。

第三空，数字全都往右移动给 temp 腾出位置放进去。放进去后 1 ~ i 有序，所以最大的数字移动到位置 i 。

第五空，具有欺骗性，很多人认为时间复杂度是 $O(\log n)$ ，但是需要注意，每个数字往后移动，需要 $O(n)$ 的时间，所以平均、最坏的时间复杂度都为 $O(n^2)$ 。

三、

（切割）将一个单调递增的数组从某一位置切成两半，然后交换左半边和右半边。现在给出交换后的数组，多次询问，每次给出一个元素 k，问数组中是否存在该元素。

提示：先使用二分查找，找到切割点，则切割点左边的数组单调递增，右边的数组单调递增，再对半边数组进行二分查找，寻找元素 k。

代码块

```
#include <iostream>
using namespace std;
const int maxn = 1e5 + 5;
int a[maxn];
bool binary_search(int l, int r, int k) { // 用于实现二分查找，判断 [l,r] 中是否有 k 出现
    while(**1**) {
        int mid = (l + r) >> 1;
```

```
        if(a[mid] == k) {
            **2**;
        } else if(a[mid] > k) {
            r = mid;
        } else {
            l = mid;
        }
    }
    return a[l] == k || a[r] == k;
}
int main() {
    int n, m, k;
    cin >> n;
    for(int i = 1; i <= n; i++) {
        cin >> a[i];
    }
    int L = 1, R = n, pos;
    while(L <= R) {
        int mid = (L + R) / 2;
        if(**3**) {
            L = mid + 1;
            pos = mid;
        } else {
            R = mid - 1;
        }
    }
    cin >> m; // 表示共有 m 次询问
    while(m--) {
        cin >> k;
        if(k >= a[1]) {
            if(**4**)    cout << "YES" << endl;
            else    cout << "NO" << endl;
        } else {
            if(**5**)    cout << "YES" << endl;
            else    cout << "NO" << endl;
        }
    }
}
```

1. 1 处应填

A. `l <= r`　　B. `l < r`　　C. `l + 1 < r`　　D. `l >= r`

2. 2 处应填

A. `break`　　B. `return true`　　C. `l = mid + 1`　　D. `r = mid - 1`

3. 3 处应填

A. `a[mid] > k`　　B. `mid > k`　　C. `a[mid] >= a[1]`　　D. `a[mid] > a[1]`

4. 4 处应填

A. `binary_search(L, R, k)`

B. `binary_search(1, pos+1, k)`

C. `binary_search(L, pos, k)`

D. `binary_search(1, pos, k)`

5. 5 处应填

A. `binary_search(pos, R, k)`

B. `binary_search(pos+1, R, k)`

C. `binary_search(pos+1, n, k)`

D. `binary_search(L, R, k)`

【要点分析】

ABCDC。

根据题意，第一次二分是为了找到切割点。切割点为 pos。由于，切割点与 k 毫无关系，所以只能选择 C 或者 D。由于，切割点左半边的数字一定比右边大，所以答案选 D。

第 37 行判断数字是在左半边还是右半边。如果 k 大于等于 a[1] 说明在左半边，那么就要从 1，pos 去查找。剩余情况从 pos + 1 ~ n 去查找。

第一空如果选择 A、B 会死循环，则选择 C。第二空找到答案 return true。