Q1. Disguised Scroll in the Archive :

In the royal archives of an ancient kingdom, thousands of scrolls are stored on dusty shelves. Each scroll carries a unique number representing the year it was written. A scholar is trying to locate a particular scroll containing forgotten war strategies. However, the scrolls were damaged during a recent earthquake and are no longer in chronological order. To complicate matters, some scrolls look nearly identical, and only their numbers can identify them correctly. The scholar cannot rely on sorting because moving fragile scrolls risks tearing them apart. Instead, they must inspect each scroll one by one until they find the one that matches the required number. The search may fail entirely if the scroll is missing, which means the scholar wasted hours looking through dust and cobwebs. This exact challenge reflects the **linear search technique**, where every item must be checked.

Input Format

- First line: Integer n (number of scrolls)
- Second line: n space-separated integers (scroll IDs)
- Third line: Integer x (desired scroll ID)

Output Format

- Position (1-based index) if found, otherwise Not Found

Sample Input 1:

```
6

45 12 89 23 56 78

23
```

Sample Output 1:

```
4
```

Sample Input 2 (False Case):

```
5

11 22 33 44 55

99
```

Sample Output 2:

```
Not Found
```

Q2. Spy’s Secret Code :

In an underground intelligence agency, enemy codes are stored in a secure vault. The vault system contains a **sorted sequence of encrypted IDs**, each representing a known intercepted message. When a spy intercepts a suspicious new signal, they must check instantly whether the code exists in the vault. However, entering the wrong code repeatedly triggers alarms. A brute-force sequential check would take too long and increase risk. The spy instead relies on a split-and-check method: they compare the signal with the middle entry, then eliminate half the vault in a single step. This process continues until the code is either found or proven absent. But beware — the sorted list contains traps: some codes are fake entries meant to confuse intruders.

Input Format

- First line: Integer n (number of codes in the vault)
- Second line: n space-separated integers (sorted codes)
- Third line: Integer x (suspicious signal code)

Output Format

- Code Found if present, otherwise Intruder Alert

Sample Input 1:

7

101 105 110 120 135 150 175

120

Sample Output 1:

Code Found

Sample Input 2 (False Case):

6

200 400 600 800 1000 1200

750

Sample Output 2:

Intruder Alert

Q3. Broken Robots' Competition :

In a robotics competition, a group of old robots must complete a sorting challenge. Each robot carries a **weight number** that represents its battery charge. They are required to stand in a line arranged from the weakest to the strongest. But the robots are clumsy: they can only compare themselves with their immediate neighbor and swap places if they are out of order. With every pass through the line, the heaviest robot slowly drags itself to the end, while lighter ones move toward the front. Some robots try to cheat by pretending to have higher charge, but repeated passes eventually reveal the truth.

Input Format

- First line: Integer n (number of robots)
- Second line: n space-separated integers (battery charges)

Output Format

- Sorted list in ascending order

Sample Input 1:

6

45 20 10 60 30 50

Sample Output 1:

10 20 30 45 50 60

Sample Input 2 (False Case – already sorted):

5

5 10 15 20 25

Sample Output 2:

5 10 15 20 25

Q4. Mysterious Jewel Auction :

At a grand royal auction, priceless jewels are being displayed on velvet cushions. Each jewel is marked with a number representing its appraised value. The king wants them to be arranged from the least valuable to the most valuable before buyers arrive. The auctioneer follows a simple rule: from the remaining unsorted jewels, always pick the one with the smallest value and place it in the correct position. The audience grows impatient because this process seems slow, but it ensures that no mistakes are made. Interestingly, if multiple jewels share the same value, the auctioneer sometimes has to pick arbitrarily, adding tension to the event.

Input Format

- First line: Integer n (number of jewels)
- Second line: n space-separated integers (jewel values)

Output Format

- Sorted values in ascending order

Sample Input 1:

5

90 20 50 10 70

Sample Output 1:

10 20 50 70 90

Sample Input 2 (False Case – duplicate values):

6

100 50 50 200 150 300

Sample Output 2:

50 50 100 150 200 300

Q5. Student Placement Desk :

During college placements, students arrive at the interview desk one by one. Each has a **score card** showing their test marks. The desk administrator must arrange the students in increasing order of scores to help companies pick candidates easily. Since students arrive randomly, the administrator inserts each new student into the correct position among those already seated, pushing others as needed. The tricky part is that some students arrive with identical scores, and the administrator must maintain fairness by inserting them carefully without disturbing the already seated ones.

Input Format

- First line: Integer n (number of students)
- Second line: n space-separated integers (student scores)

Output Format

- Sorted scores in ascending order

Sample Input 1:

6

85 70 90 60 75 80

Sample Output 1:

60 70 75 80 85 90

Sample Input 2 (False Case – all same scores):

5

50 50 50 50 50

Sample Output 2:

50 50 50 50 50

Q6. Candy Jar Mystery :

Inside a famous candy shop, jars of candies are scattered on the counter. Each jar contains a random number of candies, and children want them arranged from the least filled jar to the most filled one. The shopkeeper cannot simply pour candies out, so the jars themselves must be swapped in place. The children notice that the heaviest jars keep sliding to the end after every swap, as if they sink naturally. Though inefficient for many jars, this technique works well for their small collection. Eventually, the display looks perfect and fair where heavier elements move toward the end in passes.

Input Format

- First line: Integer n (number of jars)
- Second line: n space-separated integers (candies per jar)

Output Format

- Sorted counts in ascending order

Sample Input 1:

6

30 10 20 50 40 60

Sample Output 1:

10 20 30 40 50 60

Sample Input 2 (False Case – reverse order):

5

50 40 30 20 10

Sample Output 2:

10 20 30 40 50

Q7. Classroom Height Line-up :

In preparation for the annual photo, a teacher asks students to line up by height. She measures them quickly but writes down the list in random order. To save time, she looks for the shortest student, places them at the front, and then repeats the process with the remaining group. Each round ensures the line becomes more orderly. Though not the fastest method, it works when the class size is small and discipline is maintained where the smallest is repeatedly selected from the unsorted portion.

Input Format

- First line: Integer n (number of students)
- Second line: n space-separated integers (heights in cm)

Output Format

- Sorted heights in ascending order

Sample Input 1:

5

150 140 160 155 145

Sample Output 1:

140 145 150 155 160

Sample Input 2 (False Case – single student):

1

170

Sample Output 2:

170

Q8. Exam Score Ranking :

At a coaching center, students’ exam scores are entered randomly into the system. To prepare a merit list, the system reorders them in increasing order. For fairness, it repeatedly scans for the lowest score left and appends it to the result. Duplicate scores remain intact but affect how ties are resolved. where the list is divided into sorted and unsorted regions.

Input Format

- First line: Integer n (number of students)
- Second line: n space-separated integers (exam scores)

Output Format

- Sorted scores in ascending order

Sample Input 1:

6

55 72 48 90 68 81

Sample Output 1:

48 55 68 72 81 90

Sample Input 2 (False Case – duplicate scores):

5

70 70 80 60 90

Sample Output 2:

60 70 70 80 90

Q9. Cinema Ticket Authentication :

A cinema hall uses sequential IDs for issued tickets, stored in sorted form. When an inspector suspects a counterfeit ticket, he runs a quick search on the digital list. Instead of scanning one by one, the system jumps straight to the middle ticket ID and narrows the search range instantly. This ensures fake tickets are detected before the show begins. reducing checks drastically compared to linear search.

Input Format

- First line: Integer n (tickets sold)
- Second line: n space-separated integers (ticket IDs)
- Third line: Integer x (ticket ID to verify)

Output Format

- Valid if found, otherwise Invalid

Sample Input 1:

6

101 102 103 104 105 106

104

Sample Output 1:

Valid

Sample Input 2 (False Case):

5

11 12 13 14 15

20

Sample Output 2:

Invalid

Q10. Hospital Bed Allocation :

In a busy hospital, patients are assigned bed numbers as they arrive. A nurse, checking for available beds, must determine if a particular bed is already occupied. Since the record is simply a list of current bed numbers, not ordered in any way, she has no choice but to check each entry until she either finds the bed or confirms it free. This process is straightforward but can be time-consuming for large wards. where each element is inspected in sequence.

Input Format

- First line: Integer n (number of occupied beds)
- Second line: n space-separated integers (occupied bed numbers)
- Third line: Integer x (bed number to check)

Output Format

- Occupied if found, otherwise Free

Sample Input 1:

```
6

1 3 5 7 9 11

5
```

Sample Output 1:

```
Occupied
```

Sample Input 2 (False Case):

```
5

2 4 6 8 10

7
```

Sample Output 2:

```
Free
```

Q11. Ancient Library Scroll Arrangement :

In a forgotten monastery, monks preserve ancient scrolls filled with wisdom. One day, a novice monk is asked to arrange scrolls in order of their lengths so that elders can quickly pick the smallest ones for daily study. The novice cannot rearrange them all at once. Instead, as each scroll is handed to him, he carefully inserts it into the correct place among the already arranged ones, shifting others if needed. Over time, the scrolls form a neat sequence, each correctly placed in order.where new elements are inserted into the sorted sequence.

Input Format

- First line: Integer n (number of scrolls)
- Second line: n space-separated integers (lengths of scrolls)

Output Format

- Sorted lengths in ascending order

Sample Input 1:

6

30 10 40 20 50 25

Sample Output 1:

10 20 25 30 40 50

Sample Input 2 (False Case – all equal lengths):

4

15 15 15 15

Sample Output 2:

15 15 15 15

Q12. Festival Stalls Arrangement :

During a village festival, small stalls are set up randomly along the main street. Each stall has a token number. The organizing committee wants them arranged in ascending order of their tokens for proper navigation. Since stalls come in one by one, they are inserted carefully into the correct position among those already placed. This ensures a systematic arrangement without having to move all stalls at once.

Input Format

- First line: Integer n (number of stalls)
- Second line: n space-separated integers (stall tokens)

Output Format

- Sorted tokens in ascending order

Sample Input 1:

5

32 18 25 40 12

Sample Output 1:

12 18 25 32 40

Sample Input 2 (False Case – descending order):

5

50 40 30 20 10

Sample Output 2:

10 20 30 40 50

Q13. Data Packets in a Router :

In a network router, packets arrive at random intervals. To process them efficiently, the system must order them by their sequence numbers. However, packets often come in chunks, each already partially sorted. The router decides to split packets into halves, recursively sort them, and then merge the sorted halves. This ensures that no packet is left out of sequence. dividing and combining until order is achieved.

Input Format

- First line: Integer n (number of packets)
- Second line: n space-separated integers (sequence numbers)

Output Format

- Sorted packets in ascending order

Sample Input 1:

6

23 10 15 5 30 20

Sample Output 1:

5 10 15 20 23 30

Sample Input 2 (False Case – single packet):

1

99

Sample Output 2:

99

Q14. Music Playlist Organizer :

A popular streaming app keeps track of songs played by users. For better recommendations, the app wants to sort songs by play frequency. Since the playlist can be very large, a divide-and-merge approach is chosen. The playlist is split into smaller parts, sorted individually, and merged back together to form the complete ordered list. making it suitable for linked list-based data.

Input Format

- First line: Integer n (number of songs)
- Second line: n space-separated integers (song play counts)

Output Format

- Sorted counts in ascending order

Sample Input 1:

```
5

12 45 23 8 16
```

Sample Output 1:

```
8 12 16 23 45
```

Sample Input 2 (False Case – already sorted):

```
4

1 2 3 4
```

Sample Output 2:

```
1 2 3 4
```

Q15. Library Database Search :

A digital library stores IDs of rare manuscripts in sorted order. A researcher wants to check whether a particular manuscript exists. Manually going through the list would be inefficient. Instead, the database system checks the middle entry first, eliminating half of the search space in each step. This guarantees a quick result even for thousands of manuscripts.

Input Format

- First line: Integer n (number of manuscripts)
- Second line: n space-separated integers (sorted IDs)
- Third line: Integer x (desired manuscript ID)

Output Format

- Found if the ID exists, otherwise Not Found

Sample Input 1:

6

11 22 33 44 55 66

33

Sample Output 1:

Found

Sample Input 2 (False Case):

5

5 10 15 20 25

13

Sample Output 2:

Not Found

Q16. Security Gate Pass Verification :

At a secure tech company, every employee is given a unique ID stored in ascending order. A visitor claiming to be an employee must have their ID verified instantly at the security gate. The guard enters the ID into the system, which uses binary search to check. If the ID doesn't exist, the person is flagged as an intruder.

Input Format

- First line: Integer n (number of employees)
- Second line: n space-separated integers (employee IDs)
- Third line: Integer x (visitor's ID)

Output Format

- Access Granted if found, otherwise Access Denied

Sample Input 1:

```
7

100 200 300 400 500 600 700

400
```

Sample Output 1:

```
Access Granted
```

Sample Input 2 (False Case):

```
5

10 20 30 40 50

35
```

Sample Output 2:

```
Access Denied
```

Q17. Sorting Exam Papers :

A professor receives exam papers graded out of 100, but they are stacked randomly. To organize them, she picks one paper at a time and inserts it into the correct position among the already sorted pile. Slowly, the entire stack is ordered.

Input Format

- First line: Integer n (number of papers)
- Second line: n space-separated integers (marks)

Output Format

- Sorted marks in ascending order

Sample Input 1:

```
6

75 50 90 40 60 85
```

Sample Output 1:

```
40 50 60 75 85 90
```

Sample Input 2 (False Case – same scores):

```
3

70 70 70
```

Sample Output 2:

```
70 70 70
```

Q18. City Bus Arrival Order :

A city transport office tracks bus arrivals but finds them jumbled in their log system. To make scheduling efficient, they split the log, sort smaller parts, and merge them back into the correct order, suitable for handling large input.

Input Format

- First line: Integer n (number of buses)
- Second line: n space-separated integers (arrival times)

Output Format

- Sorted arrival times

Sample Input 1:

5

12 5 18 7 15

Sample Output 1:

5 7 12 15 18

Sample Input 2 (False Case – one bus only):

1

99

Sample Output 2:

99

Q19. Space Mission Telemetry :

During a space mission, telemetry data is received in two halves from separate satellites. Each half is unsorted, and the control team needs a single ordered sequence to analyze. They use a recursive strategy to sort smaller sets and merge them into one sorted data log. This ensures no critical information is misplaced.

Input Format

- First line: Integer n (data points)
- Second line: n space-separated integers (telemetry values)

Output Format

- Sorted telemetry data

Sample Input 1:

7

30 10 20 60 50 40 70

Sample Output 1:

10 20 30 40 50 60 70

Sample Input 2 (False Case – already sorted):

4

1 2 3 4

Sample Output 2:

1 2 3 4

Q20. Library Seat Allocation :

A modern library uses digital seat booking. Seats are numbered in ascending order, and a student requests a particular seat. The system quickly checks if the seat is already booked by performing a binary search on the sorted list of occupied seats. If not found, the student gets the seat.

Input Format

- First line: Integer n (number of booked seats)
- Second line: n space-separated integers (booked seat numbers)
- Third line: Integer x (requested seat)

Output Format

- Booked if found, otherwise Available

Sample Input 1:

5

2 4 6 8 10

6

Sample Output 1:

Booked

Sample Input 2 (False Case):

6

1 3 5 7 9 11

4

Sample Output 2:

Available

Q21. Secret Agent’s Dossier Search

In a top-secret intelligence bureau, classified dossiers are stored in a strange way. The shelves are rotated after every week to confuse intruders. As a result, the list of dossier IDs looks ordered but begins at an unknown point in the middle. An undercover agent must find whether a particular dossier ID exists in this rotated sequence. The agent cannot afford to scan one by one as alarms may trigger. Instead, they must use the properties of the arrangement to quickly pinpoint whether the dossier is present or not. If the ID is not there, the agent must report “Missing” to headquarters.

Input Format

- First line: Integer n (number of dossiers)
- Second line: n space-separated integers (rotated, sorted dossier IDs)
- Third line: Integer x (dossier ID to find)

Output Format

- Found if the ID exists, otherwise Missing

Sample Input 1:

7

40 50 60 10 20 30 35

20

Sample Output 1:

Found

Sample Input 2 (False Case):

6

15 18 20 2 5 10

25

Sample Output 2:

Missing

Q22. Archaeologist’s Artifact Ranking

A team of archaeologists has uncovered ancient coins with unique weights. They want to rank the coins in order of increasing weight. The problem is that the coins are scattered randomly in the catalog, and moving them one by one wastes precious time during excavation. The researchers must devise a way to rearrange them into a neat order for museum display. If some coins have identical weights, they must still be placed next to each other. The final record will help in authenticating the timeline of the civilization.

Input Format

- First line: Integer n (number of coins)
- Second line: n space-separated integers (coin weights)

Output Format

- Sorted weights in ascending order

Sample Input 1:

6

45 12 33 12 50 40

Sample Output 1:

12 12 33 40 45 50

Sample Input 2 (False Case – already sorted):

5

5 10 15 20 25

Sample Output 2:

5 10 15 20 25

Q23. Astronaut’s Telemetry Signal

During a space mission, telemetry signals are sent in two separate sorted streams from different satellites. The mission control must quickly find the central tendency (median) of all received values combined, without actually merging the two streams into one list. The accuracy of this median determines critical space navigation decisions. If no value exists due to empty feeds, the system reports an error. This tricky calculation ensures astronauts survive in space without communication breakdown.

Input Format

- First line: Integer n (size of first stream)
- Second line: n space-separated integers (sorted)
- Third line: Integer m (size of second stream)
- Fourth line: m space-separated integers (sorted)

Output Format

- Median value (float if needed)

Sample Input 1:

2

1 3

2

2 4

Sample Output 1:

2.5

Sample Input 2 (False Case – one empty stream):

0

3

10 20 30

Sample Output 2:

20

Q24. Tournament Threshold Score

In a global coding tournament, participants' scores are collected in random order. The organizers need to know the **k-th smallest score** to set the qualification cutoff. Sorting all scores fully would waste time, so they rely on a clever selection technique to locate exactly the needed rank. If k is larger than the number of participants, the system should report "Invalid." This ensures fairness when distributing prizes and qualifications.

Input Format

- First line: Integer n (number of participants)
- Second line: n space-separated integers (scores)
- Third line: Integer k (position to find)

Output Format

- The k-th smallest score, or Invalid if not possible

Sample Input 1:

6

7 10 4 3 20 15

3

Sample Output 1:

7

Sample Input 2 (False Case – k too large):

4

10 20 30 40

6

Sample Output 2:

Invalid

Q25. Detective's Case File Index

A detective is examining an index of case files in a crime database. The index is sorted, but he wants to know not just if a file ID exists, but the **first occurrence** of that ID because duplicates might appear due to copying errors. This helps him trace the oldest entry of a case. If the ID is absent, the system must return Not Found. The accuracy of locating the earliest entry ensures justice isn't delayed.

Input Format

- First line: Integer n (number of files)
- Second line: n space-separated integers (sorted file IDs, may contain duplicates)
- Third line: Integer x (file ID to find)

Output Format

- Index of first occurrence (1-based) or Not Found

Sample Input 1:

8

10 20 20 20 30 40 50 60

20

Sample Output 1:

2

Sample Input 2 (False Case):

5

5 10 15 20 25

12

Sample Output 2:

Not Found
