TCS CodeVita 2025: Problem Statements and Solutions

Question 1: Election Outcome

Problem Statement:

There are voters in a queue where some are supporters of Candidate A, some of Candidate B, and some neutral. The goal is to determine the final result of the election based on how neutral voters are influenced.

Input:

Length of queue n.

Characters representing voters (A, B, -).

Output:

Print the winner: 'A', 'B', or 'Coalition government'.

Constraints:

1 <= n <= 1000

Approach:

Traverse the queue from both sides.

Supporters of A influence voters on the left, and B influences voters on the right.

Simulate the movement and determine the final vote counts.

Solution (Python):

```
def election_result(voter_queue):
    a_supporters = b_supporters = 0
    for voter in voter_queue:
        if voter == 'A':
```

```
            a_supporters += 1
        elif voter == 'B':
            b_supporters += 1
    if a_supporters > b_supporters:
        return 'A'
    elif b_supporters > a_supporters:
        return 'B'
    else:
        return 'Coalition government'

n = int(input())
voter_queue = input().strip()
print(election_result(voter_queue))
```

Explanation:

We count supporters of A and B.

Whichever count is higher determines the winner.

If both are equal, it's a coalition.

Time Complexity:

O(n)

Question 2: Minimum Platforms

Problem Statement:

Given the arrival and departure times of trains, find the minimum number of platforms required at the station.

Input:

Number of trains N.

Each train's arrival and departure time.

Output:

Minimum number of platforms required.

Constraints:

1 <= N <= 10^5

Approach:

Sort the arrival and departure times.

Use a two-pointer technique to track overlapping trains.

Solution (Python):

```
def find_platforms(arrivals, departures):
    arrivals.sort()
    departures.sort()
    platforms_needed = 1
    result = 1
    i, j = 1, 0


    while i < len(arrivals) and j < len(departures):
        if arrivals[i] <= departures[j]:
            platforms_needed += 1
            i += 1
        else:
            platforms_needed -= 1
            j += 1
```

```
        result = max(result, platforms_needed)


    return result


n = int(input())
arrivals = list(map(int, input().split()))
departures = list(map(int, input().split()))
print(find_platforms(arrivals, departures))
```

Explanation:

By sorting the times and using two pointers, we can track how many platforms are needed at any given time.

Time Complexity:

O(n log n)

Question 3: Election Outcome

Problem Statement:

There are voters in a queue where some are supporters of Candidate A, some of Candidate B, and some neutral. The goal is to determine the final result of the election based on how neutral voters are influenced.

Input:

Length of queue n.

Characters representing voters (A, B, -).

Output:

Print the winner: 'A', 'B', or 'Coalition government'.

Constraints:

1 <= n <= 1000

Approach:

Traverse the queue from both sides.

Supporters of A influence voters on the left, and B influences voters on the right.

Simulate the movement and determine the final vote counts.

Solution (Python):

```
def election_result(voter_queue):
    a_supporters = b_supporters = 0
    for voter in voter_queue:
        if voter == 'A':
            a_supporters += 1
        elif voter == 'B':
            b_supporters += 1
    if a_supporters > b_supporters:
        return 'A'
    elif b_supporters > a_supporters:
        return 'B'
    else:
        return 'Coalition government'


n = int(input())
voter_queue = input().strip()
print(election_result(voter_queue))
```

Explanation:

We count supporters of A and B.

Whichever count is higher determines the winner.

If both are equal, it's a coalition.

Time Complexity:

O(n)

Question 4: Minimum Platforms

Problem Statement:

Given the arrival and departure times of trains, find the minimum number of platforms required at the station.

Input:

Number of trains N.

Each train's arrival and departure time.

Output:

Minimum number of platforms required.

Constraints:

1 <= N <= 10^5

Approach:

Sort the arrival and departure times.

Use a two-pointer technique to track overlapping trains.

Solution (Python):

```
def find_platforms(arrivals, departures):
    arrivals.sort()
```

```
    departures.sort()
    platforms_needed = 1
    result = 1
    i, j = 1, 0

    while i < len(arrivals) and j < len(departures):
        if arrivals[i] <= departures[j]:
            platforms_needed += 1
            i += 1
        else:
            platforms_needed -= 1
            j += 1
        result = max(result, platforms_needed)

    return result

n = int(input())
arrivals = list(map(int, input().split()))
departures = list(map(int, input().split()))
print(find_platforms(arrivals, departures))
```

Explanation:

By sorting the times and using two pointers, we can track how many platforms are needed at any given time.

Time Complexity:

O(n log n)

Question 5: Election Outcome

Problem Statement:

There are voters in a queue where some are supporters of Candidate A, some of Candidate B, and some neutral. The goal is to determine the final result of the election based on how neutral voters are influenced.

Input:

Length of queue n.

Characters representing voters (A, B, -).

Output:

Print the winner: 'A', 'B', or 'Coalition government'.

Constraints:

1 <= n <= 1000

Approach:

Traverse the queue from both sides.

Supporters of A influence voters on the left, and B influences voters on the right.

Simulate the movement and determine the final vote counts.

Solution (Python):

```
def election_result(voter_queue):
    a_supporters = b_supporters = 0
    for voter in voter_queue:
        if voter == 'A':
            a_supporters += 1
        elif voter == 'B':
```

```
            b_supporters += 1
    if a_supporters > b_supporters:
        return 'A'
    elif b_supporters > a_supporters:
        return 'B'
    else:
        return 'Coalition government'

n = int(input())
voter_queue = input().strip()
print(election_result(voter_queue))
```

Explanation:

We count supporters of A and B.

Whichever count is higher determines the winner.

If both are equal, it's a coalition.

Time Complexity:

O(n)

Question 6: Minimum Platforms

Problem Statement:

Given the arrival and departure times of trains, find the minimum number of platforms required at the station.

Input:

Number of trains N.

Each train's arrival and departure time.

Output:

Minimum number of platforms required.

Constraints:

1 <= N <= 10^5

Approach:

Sort the arrival and departure times.

Use a two-pointer technique to track overlapping trains.

Solution (Python):

```
def find_platforms(arrivals, departures):
    arrivals.sort()
    departures.sort()
    platforms_needed = 1
    result = 1
    i, j = 1, 0

    while i < len(arrivals) and j < len(departures):
        if arrivals[i] <= departures[j]:
            platforms_needed += 1
            i += 1
        else:
            platforms_needed -= 1
            j += 1
        result = max(result, platforms_needed)

    return result
```

```
n = int(input())

arrivals = list(map(int, input().split()))

departures = list(map(int, input().split()))

print(find_platforms(arrivals, departures))
```

Explanation:

By sorting the times and using two pointers, we can track how many platforms are needed at any given time.

Time Complexity:

O(n log n)

Question 7: Election Outcome

Problem Statement:

There are voters in a queue where some are supporters of Candidate A, some of Candidate B, and some neutral. The goal is to determine the final result of the election based on how neutral voters are influenced.

Input:

Length of queue n.

Characters representing voters (A, B, -).

Output:

Print the winner: 'A', 'B', or 'Coalition government'.

Constraints:

1 <= n <= 1000

Approach:

Traverse the queue from both sides.

Supporters of A influence voters on the left, and B influences voters on the right.

Simulate the movement and determine the final vote counts.

Solution (Python):

```
def election_result(voter_queue):
    a_supporters = b_supporters = 0
    for voter in voter_queue:
        if voter == 'A':
            a_supporters += 1
        elif voter == 'B':
            b_supporters += 1
    if a_supporters > b_supporters:
        return 'A'
    elif b_supporters > a_supporters:
        return 'B'
    else:
        return 'Coalition government'


n = int(input())
voter_queue = input().strip()
print(election_result(voter_queue))
```

Explanation:

We count supporters of A and B.

Whichever count is higher determines the winner.

If both are equal, it's a coalition.

Time Complexity:

O(n)

Question 8: Minimum Platforms

Problem Statement:

Given the arrival and departure times of trains, find the minimum number of platforms required at the station.

Input:

Number of trains N.

Each train's arrival and departure time.

Output:

Minimum number of platforms required.

Constraints:

1 <= N <= 10^5

Approach:

Sort the arrival and departure times.

Use a two-pointer technique to track overlapping trains.

Solution (Python):

```
def find_platforms(arrivals, departures):
    arrivals.sort()
    departures.sort()
    platforms_needed = 1
```

```
    result = 1
    i, j = 1, 0

    while i < len(arrivals) and j < len(departures):
        if arrivals[i] <= departures[j]:
            platforms_needed += 1
            i += 1
        else:
            platforms_needed -= 1
            j += 1
        result = max(result, platforms_needed)

    return result

n = int(input())
arrivals = list(map(int, input().split()))
departures = list(map(int, input().split()))
print(find_platforms(arrivals, departures))
```

Explanation:

By sorting the times and using two pointers, we can track how many platforms are needed at any given time.

Time Complexity:

O(n log n)

Question 9: Election Outcome

Problem Statement:

There are voters in a queue where some are supporters of Candidate A, some of Candidate B, and some neutral. The goal is to determine the final result of the election based on how neutral voters are influenced.

Input:

Length of queue n.

Characters representing voters (A, B, -).

Output:

Print the winner: 'A', 'B', or 'Coalition government'.

Constraints:

1 <= n <= 1000

Approach:

Traverse the queue from both sides.

Supporters of A influence voters on the left, and B influences voters on the right.

Simulate the movement and determine the final vote counts.

Solution (Python):

```
def election_result(voter_queue):
    a_supporters = b_supporters = 0
    for voter in voter_queue:
        if voter == 'A':
            a_supporters += 1
        elif voter == 'B':
            b_supporters += 1
    if a_supporters > b_supporters:
```

```
        return 'A'
    elif b_supporters > a_supporters:
        return 'B'
    else:
        return 'Coalition government'


n = int(input())
voter_queue = input().strip()
print(election_result(voter_queue))
```

Explanation:

We count supporters of A and B.

Whichever count is higher determines the winner.

If both are equal, it's a coalition.

Time Complexity:

O(n)

Question 10: Minimum Platforms

Problem Statement:

Given the arrival and departure times of trains, find the minimum number of platforms required at the station.

Input:

Number of trains N.

Each train's arrival and departure time.

Output:

Minimum number of platforms required.

Constraints:

1 <= N <= 10^5

Approach:

Sort the arrival and departure times.

Use a two-pointer technique to track overlapping trains.

Solution (Python):

```
def find_platforms(arrivals, departures):
    arrivals.sort()
    departures.sort()
    platforms_needed = 1
    result = 1
    i, j = 1, 0

    while i < len(arrivals) and j < len(departures):
        if arrivals[i] <= departures[j]:
            platforms_needed += 1
            i += 1
        else:
            platforms_needed -= 1
            j += 1
        result = max(result, platforms_needed)

    return result
```

```
n = int(input())

arrivals = list(map(int, input().split()))

departures = list(map(int, input().split()))

print(find_platforms(arrivals, departures))
```

Explanation:

By sorting the times and using two pointers, we can track how many platforms are needed at any given time.

Time Complexity:

O(n log n)

Question 11: Election Outcome

Problem Statement:

There are voters in a queue where some are supporters of Candidate A, some of Candidate B, and some neutral. The goal is to determine the final result of the election based on how neutral voters are influenced.

Input:

Length of queue n.

Characters representing voters (A, B, -).

Output:

Print the winner: 'A', 'B', or 'Coalition government'.

Constraints:

1 <= n <= 1000

Approach:

Traverse the queue from both sides.

Supporters of A influence voters on the left, and B influences voters on the right.

Simulate the movement and determine the final vote counts.

Solution (Python):

```
def election_result(voter_queue):

    a_supporters = b_supporters = 0

    for voter in voter_queue:

        if voter == 'A':

            a_supporters += 1

        elif voter == 'B':

            b_supporters += 1

    if a_supporters > b_supporters:

        return 'A'

    elif b_supporters > a_supporters:

        return 'B'

    else:

        return 'Coalition government'



n = int(input())

voter_queue = input().strip()

print(election_result(voter_queue))
```

Explanation:

We count supporters of A and B.

Whichever count is higher determines the winner.

If both are equal, it's a coalition.

Time Complexity:

O(n)

Question 12: Minimum Platforms

Problem Statement:

Given the arrival and departure times of trains, find the minimum number of platforms required at the station.

Input:

Number of trains N.

Each train's arrival and departure time.

Output:

Minimum number of platforms required.

Constraints:

1 <= N <= 10^5

Approach:

Sort the arrival and departure times.

Use a two-pointer technique to track overlapping trains.

Solution (Python):

```
def find_platforms(arrivals, departures):
    arrivals.sort()
    departures.sort()
    platforms_needed = 1
    result = 1
```

```
    i, j = 1, 0

    while i < len(arrivals) and j < len(departures):
        if arrivals[i] <= departures[j]:
            platforms_needed += 1
            i += 1
        else:
            platforms_needed -= 1
            j += 1
        result = max(result, platforms_needed)

    return result

n = int(input())
arrivals = list(map(int, input().split()))
departures = list(map(int, input().split()))
print(find_platforms(arrivals, departures))
```

Explanation:

By sorting the times and using two pointers, we can track how many platforms are needed at any given time.

Time Complexity:

O(n log n)

Question 13: Election Outcome

Problem Statement:

There are voters in a queue where some are supporters of Candidate A, some of Candidate B, and some neutral. The goal is to determine the final result of the election based on how neutral voters are influenced.

Input:

Length of queue n.

Characters representing voters (A, B, -).

Output:

Print the winner: 'A', 'B', or 'Coalition government'.

Constraints:

1 <= n <= 1000

Approach:

Traverse the queue from both sides.

Supporters of A influence voters on the left, and B influences voters on the right.

Simulate the movement and determine the final vote counts.

Solution (Python):

```
def election_result(voter_queue):
    a_supporters = b_supporters = 0
    for voter in voter_queue:
        if voter == 'A':
            a_supporters += 1
        elif voter == 'B':
            b_supporters += 1
    if a_supporters > b_supporters:
        return 'A'
```

```
    elif b_supporters > a_supporters:
        return 'B'
    else:
        return 'Coalition government'


n = int(input())
voter_queue = input().strip()
print(election_result(voter_queue))
```

Explanation:

We count supporters of A and B.

Whichever count is higher determines the winner.

If both are equal, it's a coalition.

Time Complexity:

O(n)

Question 14: Minimum Platforms

Problem Statement:

Given the arrival and departure times of trains, find the minimum number of platforms required at the station.

Input:

Number of trains N.

Each train's arrival and departure time.

Output:

Minimum number of platforms required.

Constraints:

1 <= N <= 10^5

Approach:

Sort the arrival and departure times.

Use a two-pointer technique to track overlapping trains.

Solution (Python):

```
def find_platforms(arrivals, departures):

    arrivals.sort()

    departures.sort()

    platforms_needed = 1

    result = 1

    i, j = 1, 0



    while i < len(arrivals) and j < len(departures):

        if arrivals[i] <= departures[j]:

            platforms_needed += 1

            i += 1

        else:

            platforms_needed -= 1

            j += 1

        result = max(result, platforms_needed)



    return result



n = int(input())
```

```
arrivals = list(map(int, input().split()))

departures = list(map(int, input().split()))

print(find_platforms(arrivals, departures))
```

Explanation:

By sorting the times and using two pointers, we can track how many platforms are needed at any given time.

Time Complexity:

O(n log n)

Question 15: Election Outcome

Problem Statement:

There are voters in a queue where some are supporters of Candidate A, some of Candidate B, and some neutral. The goal is to determine the final result of the election based on how neutral voters are influenced.

Input:

Length of queue n.

Characters representing voters (A, B, -).

Output:

Print the winner: 'A', 'B', or 'Coalition government'.

Constraints:

1 <= n <= 1000

Approach:

Traverse the queue from both sides.

Supporters of A influence voters on the left, and B influences voters on the right.

Simulate the movement and determine the final vote counts.

Solution (Python):

```
def election_result(voter_queue):
    a_supporters = b_supporters = 0
    for voter in voter_queue:
        if voter == 'A':
            a_supporters += 1
        elif voter == 'B':
            b_supporters += 1
    if a_supporters > b_supporters:
        return 'A'
    elif b_supporters > a_supporters:
        return 'B'
    else:
        return 'Coalition government'


n = int(input())
voter_queue = input().strip()
print(election_result(voter_queue))
```

Explanation:

We count supporters of A and B.

Whichever count is higher determines the winner.

If both are equal, it's a coalition.

Time Complexity:

O(n)

Question 16: Minimum Platforms

Problem Statement:

Given the arrival and departure times of trains, find the minimum number of platforms required at the station.

Input:

Number of trains N.

Each train's arrival and departure time.

Output:

Minimum number of platforms required.

Constraints:

1 <= N <= 10^5

Approach:

Sort the arrival and departure times.

Use a two-pointer technique to track overlapping trains.

Solution (Python):

```
def find_platforms(arrivals, departures):
    arrivals.sort()
    departures.sort()
    platforms_needed = 1
    result = 1
    i, j = 1, 0
```

```
    while i < len(arrivals) and j < len(departures):
        if arrivals[i] <= departures[j]:
            platforms_needed += 1
            i += 1
        else:
            platforms_needed -= 1
            j += 1
        result = max(result, platforms_needed)

    return result

n = int(input())
arrivals = list(map(int, input().split()))
departures = list(map(int, input().split()))
print(find_platforms(arrivals, departures))
```

Explanation:

By sorting the times and using two pointers, we can track how many platforms are needed at any given time.

Time Complexity:

O(n log n)

Question 17: Election Outcome

Problem Statement:

There are voters in a queue where some are supporters of Candidate A, some of Candidate B, and

some neutral. The goal is to determine the final result of the election based on how neutral voters are influenced.

Input:

Length of queue n.

Characters representing voters (A, B, -).

Output:

Print the winner: 'A', 'B', or 'Coalition government'.

Constraints:

1 <= n <= 1000

Approach:

Traverse the queue from both sides.

Supporters of A influence voters on the left, and B influences voters on the right.

Simulate the movement and determine the final vote counts.

Solution (Python):

```
def election_result(voter_queue):
    a_supporters = b_supporters = 0
    for voter in voter_queue:
        if voter == 'A':
            a_supporters += 1
        elif voter == 'B':
            b_supporters += 1
    if a_supporters > b_supporters:
        return 'A'
    elif b_supporters > a_supporters:
```

```
        return 'B'
    else:
        return 'Coalition government'

n = int(input())
voter_queue = input().strip()
print(election_result(voter_queue))
```

Explanation:

We count supporters of A and B.

Whichever count is higher determines the winner.

If both are equal, it's a coalition.

Time Complexity:

O(n)

Question 18: Minimum Platforms

Problem Statement:

Given the arrival and departure times of trains, find the minimum number of platforms required at the station.

Input:

Number of trains N.

Each train's arrival and departure time.

Output:

Minimum number of platforms required.

Constraints:

1 <= N <= 10^5

Approach:

Sort the arrival and departure times.

Use a two-pointer technique to track overlapping trains.

Solution (Python):

```
def find_platforms(arrivals, departures):
    arrivals.sort()
    departures.sort()
    platforms_needed = 1
    result = 1
    i, j = 1, 0

    while i < len(arrivals) and j < len(departures):
        if arrivals[i] <= departures[j]:
            platforms_needed += 1
            i += 1
        else:
            platforms_needed -= 1
            j += 1
        result = max(result, platforms_needed)

    return result

n = int(input())
arrivals = list(map(int, input().split()))
```

```
departures = list(map(int, input().split()))

print(find_platforms(arrivals, departures))
```

Explanation:

By sorting the times and using two pointers, we can track how many platforms are needed at any given time.

Time Complexity:

O(n log n)

Question 19: Election Outcome

Problem Statement:

There are voters in a queue where some are supporters of Candidate A, some of Candidate B, and some neutral. The goal is to determine the final result of the election based on how neutral voters are influenced.

Input:

Length of queue n.

Characters representing voters (A, B, -).

Output:

Print the winner: 'A', 'B', or 'Coalition government'.

Constraints:

1 <= n <= 1000

Approach:

Traverse the queue from both sides.

Supporters of A influence voters on the left, and B influences voters on the right.

Simulate the movement and determine the final vote counts.

Solution (Python):

```
def election_result(voter_queue):
    a_supporters = b_supporters = 0
    for voter in voter_queue:
        if voter == 'A':
            a_supporters += 1
        elif voter == 'B':
            b_supporters += 1
    if a_supporters > b_supporters:
        return 'A'
    elif b_supporters > a_supporters:
        return 'B'
    else:
        return 'Coalition government'


n = int(input())
voter_queue = input().strip()
print(election_result(voter_queue))
```

Explanation:

We count supporters of A and B.

Whichever count is higher determines the winner.

If both are equal, it's a coalition.

Time Complexity:

O(n)

Question 20: Minimum Platforms

Problem Statement:

Given the arrival and departure times of trains, find the minimum number of platforms required at the station.

Input:

Number of trains N.

Each train's arrival and departure time.

Output:

Minimum number of platforms required.

Constraints:

1 <= N <= 10^5

Approach:

Sort the arrival and departure times.

Use a two-pointer technique to track overlapping trains.

Solution (Python):

```
def find_platforms(arrivals, departures):
    arrivals.sort()
    departures.sort()
    platforms_needed = 1
    result = 1
    i, j = 1, 0

    while i < len(arrivals) and j < len(departures):
```

```
        if arrivals[i] <= departures[j]:
            platforms_needed += 1
            i += 1
        else:
            platforms_needed -= 1
            j += 1
        result = max(result, platforms_needed)

    return result

n = int(input())
arrivals = list(map(int, input().split()))
departures = list(map(int, input().split()))
print(find_platforms(arrivals, departures))
```

Explanation:

By sorting the times and using two pointers, we can track how many platforms are needed at any given time.

Time Complexity:

O(n log n)