SOLUTIONS FOR CONTEST 0

SOLUTION 1

LANGUAGE: C++

```
/*
    Time complexity: O( N )
    Space complexity: O( N )

    where 'N' is the length of the array A.
*/

int minCostFlip (int n, vector <int> &a, vector <int> &b) {

   // Find prefix suffix '0's.
   vector <int> prefix(n), suffix(n);
   for (int i = 0; i < n; i ++) {
      if (b[i] == 0) {
         prefix[i] = 1 + (i - 1 >= 0 ? prefix[i - 1] : 0);
      }
   }
   for (int i = n - 1; i >= 0; i --) {
      if (b[i] == 0) {
         suffix[i] = 1 + (i + 1 < n ? suffix[i + 1] : 0);
      }
   }

   int mxLen = 0, cost = 0;
   for (int i = 0; i < n; i ++) {
    // Find the max length you can make by flipping the current '1' if avaiable.
    int curLen = (i - 1 >= 0 ? prefix[i - 1] : 0) + (i + 1 < n ? suffix[i + 1] : 0);
    if (b[i] == 1) {
      curLen += 1;
      if (mxLen < curLen) {
        mxLen = curLen;
        cost = a[i];
      }
      else if (curLen == mxLen) {
        cost = min(cost, a[i]);
      }
    }
  }
  return cost;
}
```

LANGUAGE: JAVA

```
/*
    Time complexity: O(N)
    Space complexity: O(N)

    where 'N' is the length of the array A.
*/

import java.util.ArrayList;
import java.util.Collections;

public class Solution {

    static int minCostFlip(int n, ArrayList<Integer> a, ArrayList<Integer> b) {
        // Find prefix and suffix '0's.
        ArrayList<Integer> prefix = new ArrayList<>(Collections.nCopies(n, 0));
        ArrayList<Integer> suffix = new ArrayList<>(Collections.nCopies(n, 0));

        for (int i = 0; i < n; i++) {
            if (b.get(i) == 0) {
                prefix.set(i, 1 + (i - 1 >= 0 ? prefix.get(i - 1) : 0));
            }
        }
        for (int i = n - 1; i >= 0; i--) {
            if (b.get(i) == 0) {
                suffix.set(i, 1 + (i + 1 < n ? suffix.get(i + 1) : 0));
            }
        }

        int mxLen = 0, cost = 0;
        for (int i = 0; i < n; i++) {
            // Find the max length you can make by flipping the current '1' if available.
            int curLen = (i - 1 >= 0 ? prefix.get(i - 1) : 0) + (i + 1 < n ? suffix.get(i + 1) : 0);
            if (b.get(i) == 1) {
                curLen += 1;
                if (curLen > mxLen) {
                    mxLen = curLen;
                    cost = a.get(i);
                } else if (curLen == mxLen) {
                    cost = Math.min(cost, a.get(i));
                }
            }
        }
        return cost;
    }
}
```

LANGUAGE: PYTHON

```
"""
    Time complexity: O( N )
    Space complexity: O( N )

    where 'N' is the length of the array A.
"""
```

```
from typing import *

def min_cost_flip(n: int, a: List[int], b: List[int]):

    # Find prefix suffix '0's.
    prefix = [0] * n
    suffix = [0] * n
    for i in range(n):
        if b[i] == 0:
            prefix[i] = 1 + (prefix[i - 1] if i - 1 >= 0 else 0)
    for i in range(n - 1, -1, -1):
        if b[i] == 0:
            suffix[i] = 1 + (suffix[i + 1] if i + 1 < n else 0)

    mx_len = 0
    cost = 0
    for i in range(n):
        # Find the max length you can make by flipping the current '1' if available.
        cur_len = (prefix[i - 1] if i - 1 >= 0 else 0) + (suffix[i + 1] if i + 1 < n else 0)
        if b[i] == 1:
            cur_len += 1
            if mx_len < cur_len:
                mx_len = cur_len
                cost = a[i]
            elif cur_len == mx_len:
                cost = min(cost, a[i])
    return cost
```

SOLUTION 2:

LANGUAGE : C++

```
/*
    Time Complexity: O(N log N)
    Space Complexity: O(1)
    where 'N' is the number of elements in the array.
*/

int minPrice(int n, int k, vector<int> &a){

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

    // 'l' denotes the leftover elements after selecting 'k' elements.
    int l=n-k;
    int ans=a[l-1]-a[0];

    // Selecting 'k' elements from the array and
    // finding the minimum difference between the maximum and minimum element.
    for(int i=l;i<n;i++){
        ans=min(ans,a[i]-a[i-l+1]);
    }
    return ans;
}
```

LANGUAGE : JAVA

```
/*
    Time Complexity: O(N log N)
    Space Complexity: O(1)

    where 'N' is the number of elements in the array.
*/

import java.util.Arrays;

public class Solution {


    static int minPrice(int n, int k, int[] a) {
        Arrays.sort(a);

        // 'l' denotes the leftover elements after selecting 'k' elements.
        int l = n - k;
        int ans = a[l - 1] - a[0];

        // Selecting 'k' elements from the array and
        // finding the minimum difference between the maximum and minimum element.
        for (int i = l; i < n; i++) {
            ans = Math.min(ans, a[i] - a[i - l + 1]);
        }
        return ans;
    }
}
```

LANGUAGE: PYTHON

```
"""
    Time Complexity: O(N log N)
    Space Complexity: O(1)

    where 'N' is the number of elements in the array.
"""

from typing import *

def min_price(n: int, k: int, a: List[int]) -> int:

    # Sorting the array
    a.sort()

    # 'l' denotes the leftover elements after selecting 'k' elements.
    l = n - k

    # Initialize the answer with the difference between the maximum and minimum
element.
    ans = a[l - 1] - a[0]

    # Selecting 'k' elements from the array and
    # finding the minimum difference between the maximum and minimum element.
    for i in range(l, n):
        ans = min(ans, a[i] - a[i - l + 1])

    return ans
```

SOLUTION 3:

LANGUAGE: C++

```
/*
    Time Complexity: O(n)
    Space Complexity: O(1)

    Where 'n' denotes the length of the vector 'v'.
*/

int maximumAlternateSum(int n, vector<int> &v) {

    // Initialize an integer 'suffix' equal to '0'.
    int suffix = 0;

    // Calculate the alternate sum of the vector 'v' and store into 'suffix'.
    for (int i = 0; i < n; i++) {
        if (i % 2 == 0) {
            suffix += v[i];
        }
        else {
            suffix -= v[i];
        }
    }

    // Initialize the integers 'prefix' and 'answer' equal to '0' and '-Infinity', respectively.
    int prefix = 0, answer = -1e9;

    // Iterate through every element of the vector 'v'.
    for (int i = 0; i < n; i++) {

        // Update the 'suffix' by removing the contribution of 'v[i]'.
        if (i % 2 == 0) {
            suffix -= v[i];
        }
        else {
            suffix += v[i];
        }

        // Update the 'answer' based on the alternate sum obtained by removing the
element 'v[i]'.
        int currentSum = prefix - suffix;
        answer = max(answer, currentSum);

        // Update 'prefix' by adding the contribution of 'v[i]'.
        if (i % 2 == 0) {
            prefix += v[i];
        }
        else {
            prefix -= v[i];
        }
    }

    // Return 'answer'.
    return answer;
}
```

LANGUAGE: JAVA

```
/*
    Time Complexity: O(n)
    Space Complexity: O(1)

    Where 'n' denotes the length of the array 'v'.
*/

public class Solution {

    public static int maximumAlternateSum(int n, int[] v) {
        // Initialize an integer 'suffix' equal to '0'.
        int suffix = 0;

        // Calculate the alternate sum of the array 'v' and store into 'suffix'.
        for (int i = 0; i < n; i++) {
            if (i % 2 == 0) {
                suffix += v[i];
            } else {
                suffix -= v[i];
            }
        }

        // Initialize the integers 'prefix' and 'answer' equal to '0' and '-Infinity',
respectively.
        int prefix = 0, answer = Integer.MIN_VALUE;

        // Iterate through every element of the array 'v'.
        for (int i = 0; i < n; i++) {

            // Update the 'suffix' by removing the contribution of 'v[i]'.
            if (i % 2 == 0) {
                suffix -= v[i];
            } else {
                suffix += v[i];
            }

            // Update the 'answer' based on the alternate sum obtained by removing the
element 'v[i]'.
            int currentSum = prefix - suffix;
            answer = Math.max(answer, currentSum);

            // Update 'prefix' by adding the contribution of 'v[i]'.
            if (i % 2 == 0) {
                prefix += v[i];
            } else {
                prefix -= v[i];
            }
        }

        // Return 'answer'.
        return answer;
    }
}
```

LANGUAGE: PYTHON

```
"""
    Time Complexity: O(n)
    Space Complexity: O(1)

    Where 'n' denotes the length of the vector 'v'.
"""

from typing import List

def maximumAlternateSum(n: int, v: List[int]) -> int:

    # Initialize an integer 'suffix' equal to '0'.
    suffix = 0

    # Calculate the alternate sum of the vector 'v' and store into 'suffix'.
    for i in range(n):
        if i % 2 == 0:
            suffix += v[i]
        else:
            suffix -= v[i]

    # Initialize the integers 'prefix' and 'answer' equal to '0' and '-Infinity', respectively.
    prefix = 0
    answer = float('-inf')

    # Iterate through every element of the vector 'v'.
    for i in range(n):
        # Update the 'suffix' by removing the contribution of 'v[i]'.
        if i % 2 == 0:
            suffix -= v[i]
        else:
            suffix += v[i]

        # Update the 'answer' based on the alternate sum obtained by removing the
element 'v[i]'.
        currentSum = prefix - suffix
        answer = max(answer, currentSum)

        # Update 'prefix' by adding the contribution of 'v[i]'.
        if i % 2 == 0:
            prefix += v[i]
        else:
            prefix -= v[i]

    # Return 'answer'.
    return answer
```

SOLUTION 4:

Will be discussed during the meet.

SOLUTION 5:

LANGUAGE: C++

```
/*
    Time Complexity: O(n ^ 2)
    Space Complexity: O(n)

    Where 'n' is the length of the array 'a'.
*/

vector<int> nextGreaterElementII(vector<int>& a) {
    int n = a.size();

    // Declare an array 'answer' of size 'n',
    // to store the Next Greater Element for each element.
    vector <int> answer(n);

    // Run a for loop form i = 0 to 'n' - 1.
    for (int i = 0; i < n; i++) {

        // Initialise an integer variable 'currentAnswer' to -1.
        int currentAnswer = -1;
        for (int j = 1; j < n; j++) {
            // If a[(i + j) % n] > a[i] then update
            // 'currentAnswer' to a[(i + j) % n] and break.
            if (a[(i + j) % n] > a[i]) {
                currentAnswer = a[(i + j) % n];
                break;
            }
        }

        // Update answer[i] to 'currentAnswer'.
        answer[i] = currentAnswer;
    }

    // Return 'answer' as the answer to the problem.
    return answer;
}
```

LANGUAGE: JAVA

```
/*
    Time Complexity: O(N^2)
    Space Complexity: O(N)

    Where 'N' is the length of the array 'A'.
*/

public class Solution {
    public static int[] nextGreaterElementII(int []a) {
        int n = a.length;

        // Initialize an array 'answer' of size 'N',
        // to store the Next Greater Element for each element.
        int []answer = new int[n];

        // Run a for loop form i=0 to N-1.
        for (int i = 0; i < n; i++) {
            // Initialise an integer variable 'currentAnswer' to -1.
            int currentAnswer = -1;
            for (int j = 1; j < n; j++) {

                // If A[ (i+j)%N ] > A[ i ] then update
                // 'currentAnswer' to A[ (i+j)%N ] and break.
                if (a[(i + j) % n] > a[i]) {
                    currentAnswer = a[(i + j) % n];
                    break;
                }
            }

            // Update answer[ i ] to 'currentAnswer.
            answer[i] = currentAnswer;
        }

        // Return 'answer' as the answer to the problem.
        return answer;
    }
}
```

LANGUAGE: PYTHON

```
'''

    Time Complexity: O(N^2)
    Space Complexity: O(N)

    Where 'N' is the length of the array 'A'.

'''


def nextGreaterElementII(a: int) -> int:

    n = len(a)
```

```
# Initialize an array â€˜answerâ€™ of size â€˜Nâ€™,
# to store the Next Greater Element for each element.

answer = [0 for i in range(n)]

# Run a for loop form i=0 to N-1.

for i in range(n):

    # Initialise an integer variable â€˜currentAnswerâ€™ to -1.

    currentAnswer = -1

    for j in range(n):

        # If A[ (i+j)%N ] > A[ i ] then update
        # â€˜currentAnswerâ€™ to A[ (i+j)%N ] and break.

        ind = int((i + j) % n)

        if (a[ind] > a[i]):

            currentAnswer = a[int((i + j) % n)]

            break

    # Update answer[ i ] to â€˜currentAnswer.

    answer[i] = currentAnswer

# Return â€˜answerâ€™ as the answer to the problem.
return answer
```

SOLUTION 6:

LANGUAGE: C++

```
/*
```

```
    Time Complexity: O(n * m)
    Space Complexity: O(n * m)

    Where 'n' and 'm' denote the number of vectors and the number of elements in each vector, respectively.
*/

vector<int> lexicographicallyMinimum(int n, int m, vector<vector<int>> &v) {

    // Initialize a two-dimensional vector 'valid' to store the dp states.
    vector<vector<bool>> valid(n, vector<bool>(m, false));

    // Initialize an integer 'nextMax' to infinity.
    int nextMax = 1e9;
    for (int j = m - 1; j >= 0; j--) {

        // Initialize an integer 'currentMax' equal to '0'.
        int currentMax = 0;
        for (int i = 0; i < n; i++) {

            // Calculate the value of 'valid[i][j]'.
            if (v[i][j] <= nextMax) {
                valid[i][j] = true;
                currentMax = max(currentMax, v[i][j]);
            }
        }

        // Update 'nextMax' to 'currentMax'.
        nextMax = currentMax;
    }

    // Initialize a vector 'answer'.
    vector<int> answer(m);

    // Initialize an integer 'previous' equal to '0'.
    int previous = 0;
    for (int j = 0; j < m; j++) {
        for (int i = 0; i < n; i++) {

            // Find the minimum value of 'j' to perform the replacement operation.
            if (valid[i][j] && v[i][j] >= previous) {
                previous = v[i][j];
                answer[j] = i;
                break;
            }

            // Return '-1' if there is no such 'j'.
            if (i == n - 1) {
                return {-1};
            }
        }
    }

    // Return the 'answer'.
    return answer;
}
```

LANGUAGE: JAVA

```
/*
    Time Complexity: O(n * m)
    Space Complexity: O(n * m)

    Where 'n' and 'm' denote the number of vectors and the number of elements in each vector, respectively.
*/

import java.util.*;

public class Solution {
    public static int[] lexicographicallyMinimum(int n, int m, int[][] v) {
        // Initialize a two-dimensional array 'valid' to store the dp states.
        boolean[][] valid = new boolean[n][m];

        // Initialize an integer 'nextMax' to infinity.
        int nextMax = (int) 1e9;
        for (int j = m - 1; j >= 0; j--) {
            // Initialize an integer 'currentMax' equal to '0'.
            int currentMax = 0;
            for (int i = 0; i < n; i++) {
                // Calculate the value of 'valid[i][j]'.
                if (v[i][j] <= nextMax) {
                    valid[i][j] = true;
                    currentMax = Math.max(currentMax, v[i][j]);
                }
            }
            // Update 'nextMax' to 'currentMax'.
            nextMax = currentMax;
        }

        // Initialize an array 'answer'.
        int[] answer = new int[m];

        // Initialize an integer 'previous' equal to '0'.
        int previous = 0;
        for (int j = 0; j < m; j++) {
            for (int i = 0; i < n; i++) {
                // Find the minimum value of 'j' to perform the replacement operation.
                if (valid[i][j] && v[i][j] >= previous) {
                    previous = v[i][j];
                    answer[j] = i;
                    break;
                }
                // Return '-1' if there is no such 'j'.
                if (i == n - 1) {
                    return new int[]{-1};
                }
            }
        }
        // Return the 'answer'.
        return answer;
    }
}
```

LANGUAGE: PYTHON

```
"""
    Time Complexity: O(n * m)
    Space Complexity: O(n * m)
    Where 'n' and 'm' denote the number of vectors and the number of elements in each vector, respectively.
"""

from typing import List

def lexicographicallyMinimum(n: int, m: int, v: List[List[int]]) -> List[int]:
    # Initialize a two-dimensional list 'valid' to store the dp states.
    valid = [[False] * m for _ in range(n)]

    # Initialize an integer 'nextMax' to infinity.
    nextMax = float('inf')
    for j in range(m - 1, -1, -1):
        # Initialize an integer 'currentMax' equal to '0'.
        currentMax = 0
        for i in range(n):
            # Calculate the value of 'valid[i][j]'.
            if v[i][j] <= nextMax:
                valid[i][j] = True
                currentMax = max(currentMax, v[i][j])

        # Update 'nextMax' to 'currentMax'.
        nextMax = currentMax

    # Initialize a list 'answer'.
    answer = [0] * m

    # Initialize an integer 'previous' equal to '0'.
    previous = 0
    for j in range(m):
        for i in range(n):
            # Find the minimum value of 'j' to perform the replacement operation.
            if valid[i][j] and v[i][j] >= previous:
                previous = v[i][j]
                answer[j] = i
                break

            # Return '-1' if there is no such 'j'.
            if i == n - 1:
                return [-1]
```

```
# Return the 'answer'.
return answer
```