Plus Minus

```
#include <assert.h>
#include <ctype.h>
#include <limits.h>
#include <math.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* readline();
char* ltrim(char*);
char* rtrim(char*);
char** split_string(char*);

int parse_int(char*);

/*
 * Complete the 'plusMinus' function below.
 * The function accepts INTEGER_ARRAY arr as parameter.
 */

void plusMinus(int arr_count, int* arr) {
    int positive_count = 0;
    int negative_count = 0;
    int zero_count = 0;
    for (int i = 0; i < arr_count; i++) {
        if (arr[i] > 0) {
            positive_count++;
        } else if (arr[i] < 0) {
            negative_count++;
        } else {
            zero_count++;
        }
    }

    double total_elements = (double)arr_count;

    printf("%.6f\n", (double)positive_count / total_elements);
    printf("%.6f\n", (double)negative_count / total_elements);
    printf("%.6f\n", (double)zero_count / total_elements);
}

int main()
{
    int n = parse_int(ltrim(rtrim(readline())));

    char** arr_temp = split_string(rtrim(readline()));

    int* arr = malloc(n * sizeof(int));

    for (int i = 0; i < n; i++) {
        int arr_item = parse_int(*(arr_temp + i));

        *(arr + i) = arr_item;
    }
    plusMinus(n, arr);

    return 0;
}

char* readline() {
    size_t alloc_length = 1024;
    size_t data_length = 0;

    char* data = malloc(alloc_length);

    while (true) {
        char* cursor = data + data_length;
        char* line = fgets(cursor, alloc_length - data_length, stdin);

        if (!line) {
            break;
        }

        data_length += strlen(cursor);

        if (data_length < alloc_length - 1 || data[data_length - 1] == '\n') {
            break;
        }

        alloc_length <<= 1;

        data = realloc(data, alloc_length);

        if (!data) {
            data = '\0';
            break;
        }
    }

    if (data[data_length - 1] == '\n') {
        data[data_length - 1] = '\0';

        data = realloc(data, data_length);

        if (!data) {
            data = '\0';
        }
    } else {
        data = realloc(data, data_length + 1);

        if (!data) {
            data = '\0';
        } else {
            data[data_length] = '\0';
        }
    }

    return data;
}

char* ltrim(char* str) {
    if (!str) {
        return '\0';
    }
    if (!*str) {
        return str;
    }

    while (*str != '\0' && isspace(*str)) {
        str++;
    }

    return str;
}

char* rtrim(char* str) {
    if (!str) {
        return '\0';
    }

    if (!*str) {
        return str;
    }

    char* end = str + strlen(str) - 1;

    while (end >= str && isspace(*end)) {
        end--;
    }

    *(end + 1) = '\0';

    return str;
}
```

```
char** split_string(char* str) {
    char** splits = NULL;
    char* token = strtok(str, " ");

    int spaces = 0;

    while (token) {
        splits = realloc(splits, sizeof(char*) * ++spaces);

        if (!splits) {
            return splits;
        }

        splits[spaces - 1] = token;

        token = strtok(NULL, " ");
    }

    return splits;
}

int parse_int(char* str) {
    char* endptr;
    int value = strtol(str, &endptr, 10);

    if (endptr == str || *endptr != '\0') {
        exit(EXIT_FAILURE);
    }

    return value;
}
```

Mini- Max Sum

```
#include <assert.h>

#include <ctype.h>

#include <limits.h>

#include <math.h>

#include <stdbool.h>

#include <stddef.h>

#include <stdint.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>


char* readline();

char* ltrim(char*);

char* rtrim(char*);

char** split_string(char*);


int parse_int(char*);


/*

 * Complete the 'miniMaxSum' function below.

 *

 * The function accepts INTEGER_ARRAY arr as parameter.

 */


void miniMaxSum(int arr_count, int* arr) {

    // Using long long int to avoid integer overflow

    long long int total_sum = 0;
    long long int min_val = arr[0];
    long long int max_val = arr[0];

    // Loop through the array to find sum, min, and max
    for (int i = 0; i < arr_count; i++) {
        total_sum += arr[i];

        if (arr[i] < min_val) {
            min_val = arr[i];
        }

        if (arr[i] > max_val) {
            max_val = arr[i];
        }
    }

    // Calculate min and max sums
    long long int min_sum = total_sum - max_val;
    long long int max_sum = total_sum - min_val;

    // Print the results
    printf("%lld %lld\n", min_sum, max_sum);
}

int main()
{
    char** arr_temp = split_string(rtrim(readline()));

    int* arr = malloc(5 * sizeof(int));

    for (int i = 0; i < 5; i++) {
        int arr_item = parse_int(*(arr_temp + i));

        *(arr + i) = arr_item;
    }

    miniMaxSum(5, arr);

    return 0;
}

char* readline() {
    size_t alloc_length = 1024;
    size_t data_length = 0;

    char* data = malloc(alloc_length);

    while (true) {
        char* cursor = data + data_length;
        char* line = fgets(cursor, alloc_length - data_length, stdin);

        if (!line) {
            break;
        }

        data_length += strlen(cursor);

        if (data_length < alloc_length - 1 || data[data_length - 1] == '\n') {
            break;
        }

        alloc_length <<= 1;
        data = realloc(data, alloc_length);

        if (!data) {
            data = '\0';

            break;
        }
    }

    if (data[data_length - 1] == '\n') {
        data[data_length - 1] = '\0';

        data = realloc(data, data_length);

        if (!data) {
            data = '\0';
        }
    } else {
        data = realloc(data, data_length + 1);

        if (!data) {
            data = '\0';
        } else {
            data[data_length] = '\0';
        }
    }

    return data;
}
```

```
char* ltrim(char* str) {
    if (!str) {
        return '\0';
    }

    if (!*str) {
        return str;
    }

    while (*str != '\0' && isspace(*str)) {
        str++;
    }

    return str;
}

char* rtrim(char* str) {
    if (!str) {
        return '\0';
    }

    if (!*str) {
        return str;
    }

    char* end = str + strlen(str) - 1;

    while (end >= str && isspace(*end)) {
        end--;
    }
    *(end + 1) = '\0';

    return str;
}

char** split_string(char* str) {
    char** splits = NULL;
    char* token = strtok(str, " ");

    int spaces = 0;

    while (token) {
        splits = realloc(splits, sizeof(char*) * ++spaces);

        if (!splits) {
            return splits;
        }

        splits[spaces - 1] = token;

        token = strtok(NULL, " ");
    }

    return splits;
}

int parse_int(char* str) {
    char* endptr;
    int value = strtol(str, &endptr, 10);

    if (endptr == str || *endptr != '\0') {
        exit(EXIT_FAILURE);
    }

    return value;
}
```

Time Conversion

```
#include <assert.h>
#include <ctype.h>
#include <limits.h>
#include <math.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* readline();

/*
 * Complete the 'timeConversion' function below.
 *
 * The function is expected to return a STRING.
 * The function accepts STRING s as parameter.
 */
char* timeConversion(char* s) {
    // Dynamically allocate memory for the new string
    // It needs 8 characters for "HH:mm:ss" + 1 for the null terminator.
    char* result = malloc(sizeof(char) * 9);

    // Extract hours, minutes, and seconds from the input string
    int hh, mm, ss;
    sscanf(s, "%d:%d:%d", &hh, &mm, &ss);

    // Check the AM/PM part of the string at index 8 and convert hours
    if (s[8] == 'P' && hh != 12) {
        hh += 12; // Add 12 hours for PM times, except for 12 PM
    } else if (s[8] == 'A' && hh == 12) {
        hh = 0;   // 12 AM becomes 00 in 24-hour format
    }

    // Format the new time string into the allocated memory
    // %02d ensures that single-digit numbers are padded with a leading zero.
    snprintf(result, 9, "%02d:%02d:%02d", hh, mm, ss);

    return result;
}

int main()
{
    FILE* fptr = fopen(getenv("OUTPUT_PATH"), "w");

    char* s = readline();
    char* result = timeConversion(s);

    fprintf(fptr, "%s\n", result);

    fclose(fptr);

    return 0;
}

char* readline() {
    size_t alloc_length = 1024;
    size_t data_length = 0;

    char* data = malloc(alloc_length);

    while (true) {
        char* cursor = data + data_length;
        char* line = fgets(cursor, alloc_length - data_length, stdin);

        if (!line) {
            break;
        }

        data_length += strlen(cursor);

        if (data_length < alloc_length - 1 || data[data_length - 1] == '\n') {
            break;
        }

        alloc_length <<= 1;
        data = realloc(data, alloc_length);

        if (!data) {
            data = '\0';

            break;
        }
    }

    if (data[data_length - 1] == '\n') {
        data[data_length - 1] = '\0';

        data = realloc(data, data_length);

        if (!data) {
            data = '\0';
        }
    } else {
        data = realloc(data, data_length + 1);

        if (!data) {
            data = '\0';
        } else {
            data[data_length] = '\0';
        }
    }

    return data;
}
```

Sparse Arrays

```
#include <assert.h>
#include <ctype.h>
#include <limits.h>
#include <math.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* readline();
char* ltrim(char*);
char* rtrim(char*);

int parse_int(char*);

/*
 * Complete the 'matchingStrings' function below.
 *
 * The function is expected to return an INTEGER_ARRAY.
 * The function accepts following parameters:
 * 1. STRING_ARRAY strings
 * 2. STRING_ARRAY queries
 */
int* matchingStrings(int strings_count, char** strings, int queries_count, char** queries, int* result_count) {
    // Dynamically allocate memory for the result array.
    // The size of the array is equal to the number of queries.
    int* result = malloc(sizeof(int) * queries_count);
    *result_count = queries_count;

    // Loop through each query.
    for (int i = 0; i < queries_count; i++) {
        int count = 0; // Initialize a counter for the current query.

        // Compare the current query with every string in the strings array.
        for (int j = 0; j < strings_count; j++) {
            // strcmp returns 0 if the strings are identical.
            if (strcmp(queries[i], strings[j]) == 0) {
                count++;
            }
        }

        // Store the final count for the current query.
        result[i] = count;
    }

    return result;
}

int main()
{
    FILE* fptr = fopen(getenv("OUTPUT_PATH"), "w");

    int strings_count = parse_int(ltrim(rtrim(readline())));

    char** strings = malloc(strings_count * sizeof(char*));
```

```
for (int i = 0; i < strings_count; i++) {
    char* strings_item = readline();

    *(strings + i) = strings_item;
}

int queries_count = parse_int(ltrim(rtrim(readline())));

char** queries = malloc(queries_count * sizeof(char*));

for (int i = 0; i < queries_count; i++) {
    char* queries_item = readline();

    *(queries + i) = queries_item;
}

int res_count;
int* res = matchingStrings(strings_count, strings, queries_count, queries, &res_count);

for (int i = 0; i < res_count; i++) {
    fprintf(fptr, "%d", *(res + i));

    if (i != res_count - 1) {
        fprintf(fptr, "\n");
    }
}

fprintf(fptr, "\n");

fclose(fptr);
```

```
    return 0;
}

char* readline() {
    size_t alloc_length = 1024;
    size_t data_length = 0;

    char* data = malloc(alloc_length);

    while (true) {
        char* cursor = data + data_length;
        char* line = fgets(cursor, alloc_length - data_length, stdin);

        if (!line) {
            break;
        }

        data_length += strlen(cursor);

        if (data_length < alloc_length - 1 || data[data_length - 1] == '\n') {
            break;
        }

        alloc_length <<= 1;

        data = realloc(data, alloc_length);

        if (!data) {
            data = '\0';

            break;
        }
    }

    if (data[data_length - 1] == '\n') {
        data[data_length - 1] = '\0';

        data = realloc(data, data_length);

        if (!data) {
            data = '\0';
        }
    } else {
        data = realloc(data, data_length + 1);

        if (!data) {
            data = '\0';
        } else {
            data[data_length] = '\0';
        }
    }

    return data;
}

char* ltrim(char* str) {
    if (!str) {
        return '\0';
    }

    if (!*str) {
        return str;
    }

    while (*str != '\0' && isspace(*str)) {
        str++;
    }

    return str;
}

char* rtrim(char* str) {
    if (!str) {
        return '\0';
    }

    if (!*str) {
        return str;
    }

    char* end = str + strlen(str) - 1;

    while (end >= str && isspace(*end)) {
        end--;
    }

    *(end + 1) = '\0';

    return str;
}

int parse_int(char* str) {
    char* endptr;
    int value = strtol(str, &endptr, 10);

    if (endptr == str || *endptr != '\0') {
        exit(EXIT_FAILURE);
    }

    return value;
}
```

Lonely Integer

```
#include <assert.h>
#include <ctype.h>
#include <limits.h>
#include <math.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* readline();
char* ltrim(char*);
char* rtrim(char*);
char** split_string(char*);

int parse_int(char*);

/*
 * Complete the 'lonelyinteger' function below.
 *
 * The function is expected to return an INTEGER.
 * The function accepts INTEGER_ARRAY a as parameter.
 */

int lonelyinteger(int a_count, int* a) {
    int unique_element = 0;
    // The key is to use the bitwise XOR operator.
    // XORing all elements together will cancel out the paired elements,
    // leaving only the unique one.
    for (int i = 0; i < a_count; i++) {
        unique_element ^= a[i];
    }

    return unique_element;
}

int main()
{
    FILE* fptr = fopen(getenv("OUTPUT_PATH"), "w");

    int n = parse_int(ltrim(rtrim(readline())));

    char** a_temp = split_string(rtrim(readline()));

    int* a = malloc(n * sizeof(int));

    for (int i = 0; i < n; i++) {
        int a_item = parse_int(*(a_temp + i));

        *(a + i) = a_item;
    }

    int result = lonelyinteger(n, a);

    fprintf(fptr, "%d\n", result);
    fclose(fptr);

    return 0;
}

char* readline() {
    size_t alloc_length = 1024;
    size_t data_length = 0;

    char* data = malloc(alloc_length);

    while (true) {
        char* cursor = data + data_length;
        char* line = fgets(cursor, alloc_length - data_length, stdin);

        if (!line) {
            break;
        }

        data_length += strlen(cursor);

        if (data_length < alloc_length - 1 || data[data_length - 1] == '\n') {
            break;
        }

        alloc_length <<= 1;

        data = realloc(data, alloc_length);

        if (!data) {
            data = '\0';
            break;
        }
    }

    if (data[data_length - 1] == '\n') {
        data[data_length - 1] = '\0';

        data = realloc(data, data_length);

        if (!data) {
            data = '\0';
        }
    } else {
        data = realloc(data, data_length + 1);

        if (!data) {
            data = '\0';
        } else {
            data[data_length] = '\0';
        }
    }

    return data;
}

char* ltrim(char* str) {
    if (!str) {
        return '\0';
    }
    if (!*str) {
        return str;
    }

    while (*str != '\0' && isspace(*str)) {
        str++;
    }

    return str;
}

char* rtrim(char* str) {
    if (!str) {
        return '\0';
    }

    if (!*str) {
        return str;
    }

    char* end = str + strlen(str) - 1;

    while (end >= str && isspace(*end)) {
        end--;
    }

    *(end + 1) = '\0';

    return str;
}
```

```
char** split_string(char* str) {
    char** splits = NULL;
    char* token = strtok(str, " ");

    int spaces = 0;

    while (token) {
        splits = realloc(splits, sizeof(char*) * ++spaces);

        if (!splits) {
            return splits;
        }

        splits[spaces - 1] = token;

        token = strtok(NULL, " ");
    }

    return splits;
}

int parse_int(char* str) {
    char* endptr;
    int value = strtol(str, &endptr, 10);

    if (endptr == str || *endptr != '\0') {
        exit(EXIT_FAILURE);
    }

    return value;
}
```

Flipping Bits

```
#include <assert.h>
#include <ctype.h>
#include <limits.h>
#include <math.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* readline();
char* ltrim(char*);
char* rtrim(char*);

int parse_int(char*);
long parse_long(char*);

/*
 * Complete the 'flippingBits' function below.
 *
 * The function is expected to return a LONG_INTEGER.
 * The function accepts LONG_INTEGER n as parameter.
 */
long flippingBits(long n) {
    // A 32-bit unsigned integer has a maximum value of 2^32 - 1.
    // This value is 4294967295.
    // Flipping all bits of a 32-bit number `n` is equivalent to calculating `(2^32 - 1) - n`.
    unsigned long mask = 4294967295;
    return mask - n;
}

int main()
{
    FILE* fptr = fopen(getenv("OUTPUT_PATH"), "w");

    int q = parse_int(ltrim(rtrim(readline())));

    for (int q_itr = 0; q_itr < q; q_itr++) {
        long n = parse_long(ltrim(rtrim(readline())));

        long result = flippingBits(n);

        fprintf(fptr, "%ld\n", result);
    }

    fclose(fptr);

    return 0;
}

char* readline() {
    size_t alloc_length = 1024;
    size_t data_length = 0;

    char* data = malloc(alloc_length);
```

```
while (true) {
   char* cursor = data + data_length;
   char* line = fgets(cursor, alloc_length - data_length, stdin);


   if (!line) {
      break;
   }


   data_length += strlen(cursor);


   if (data_length < alloc_length - 1 || data[data_length - 1] == '\n') {
      break;
   }


   alloc_length <<= 1;


   data = realloc(data, alloc_length);


   if (!data) {
      data = '\0';


      break;
   }
}


if (data[data_length - 1] == '\n') {
   data[data_length - 1] = '\0';


   data = realloc(data, data_length);
```

```
        if (!data) {
            data = '\0';
        }
    } else {
        data = realloc(data, data_length + 1);

        if (!data) {
            data = '\0';
        } else {
            data[data_length] = '\0';
        }
    }

    return data;
}

char* ltrim(char* str) {
    if (!str) {
        return '\0';
    }

    if (!*str) {
        return str;
    }

    while (*str != '\0' && isspace(*str)) {
        str++;
    }

    return str;
}
```

```
char* rtrim(char* str) {
    if (!str) {
        return '\0';
    }

    if (!*str) {
        return str;
    }

    char* end = str + strlen(str) - 1;

    while (end >= str && isspace(*end)) {
        end--;
    }

    *(end + 1) = '\0';

    return str;
}

int parse_int(char* str) {
    char* endptr;
    int value = strtol(str, &endptr, 10);

    if (endptr == str || *endptr != '\0') {
        exit(EXIT_FAILURE);
    }

    return value;
}
```

```
long parse_long(char* str) {
    char* endptr;
    long value = strtol(str, &endptr, 10);

    if (endptr == str || *endptr != '\0') {
        exit(EXIT_FAILURE);
    }

    return value;
}
```

Diagonal Difference

```
#include <assert.h>
#include <ctype.h>
#include <limits.h>
#include <math.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* readline();
char* ltrim(char*);
char* rtrim(char*);
char** split_string(char*);

int parse_int(char*);

/*
 * Complete the 'diagonalDifference' function below.
 *
 * The function is expected to return an INTEGER.
 * The function accepts 2D_INTEGER_ARRAY arr as parameter.
 */
int diagonalDifference(int arr_rows, int arr_columns, int** arr) {
    int primary_diagonal_sum = 0;
    int secondary_diagonal_sum = 0;
    // Use a single loop to iterate through the matrix diagonals.
    // Since it's a square matrix, arr_rows is equal to arr_columns.
    for (int i = 0; i < arr_rows; i++) {
        // The primary diagonal consists of elements where the row and column indices are the same (arr[i][i]).
        primary_diagonal_sum += arr[i][i];

        // The secondary diagonal consists of elements where the sum of the row and column indices equals n-1.
        // For a given row `i`, the column index is `arr_rows - 1 - i`.
        secondary_diagonal_sum += arr[i][arr_rows - 1 - i];
    }

    // Return the absolute difference between the two sums.
    return abs(primary_diagonal_sum - secondary_diagonal_sum);
}

int main()
{
    FILE* fptr = fopen(getenv("OUTPUT_PATH"), "w");

    int n = parse_int(ltrim(rtrim(readline())));

    int** arr = malloc(n * sizeof(int*));

    for (int i = 0; i < n; i++) {
        *(arr + i) = malloc(n * (sizeof(int)));

        char** arr_item_temp = split_string(rtrim(readline()));

        for (int j = 0; j < n; j++) {
            int arr_item = parse_int(*(arr_item_temp + j));

            *(*(arr + i) + j) = arr_item;
        }
    }

    int result = diagonalDifference(n, n, arr);

    fprintf(fptr, "%d\n", result);

    fclose(fptr);

    return 0;
}

char* readline() {
    size_t alloc_length = 1024;
    size_t data_length = 0;

    char* data = malloc(alloc_length);

    while (true) {
        char* cursor = data + data_length;
        char* line = fgets(cursor, alloc_length - data_length, stdin);

        if (!line) {
            break;
        }

        data_length += strlen(cursor);
        if (data_length < alloc_length - 1 || data[data_length - 1] == '\n') {
            break;
        }

        alloc_length <<= 1;

        data = realloc(data, alloc_length);

        if (!data) {
            data = '\0';

            break;
        }
    }

    if (data[data_length - 1] == '\n') {
        data[data_length - 1] = '\0';

        data = realloc(data, data_length);

        if (!data) {
            data = '\0';
        }
    } else {
        data = realloc(data, data_length + 1);

        if (!data) {
            data = '\0';
        } else {
            data[data_length] = '\0';
        }
    }

    return data;
}

char* ltrim(char* str) {
    if (!str) {
        return '\0';
    }

    if (!*str) {
        return str;
    }

    while (*str != '\0' && isspace(*str)) {
        str++;
    }

    return str;
}

char* rtrim(char* str) {
    if (!str) {
        return '\0';
    }

    if (!*str) {
        return str;
    }

    char* end = str + strlen(str) - 1;
    while (end >= str && isspace(*end)) {
        end--;
    }

    *(end + 1) = '\0';

    return str;
}

char** split_string(char* str) {
    char** splits = NULL;
    char* token = strtok(str, " ");

    int spaces = 0;

    while (token) {
        splits = realloc(splits, sizeof(char*) * ++spaces);

        if (!splits) {
            return splits;
        }

        splits[spaces - 1] = token;

        token = strtok(NULL, " ");
    }

    return splits;
}
```

```
int parse_int(char* str) {
    char* endptr;
    int value = strtol(str, &endptr, 10);

    if (endptr == str || *endptr != '\0') {
        exit(EXIT_FAILURE);
    }

    return value;
}
```

Counting Sort 1

```
#include <assert.h>
#include <ctype.h>
#include <limits.h>
#include <math.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* readline();
char* ltrim(char*);
char* rtrim(char*);
char** split_string(char*);

int parse_int(char*);

/*
 * Complete the 'countingSort' function below.
 *
 * The function is expected to return an INTEGER_ARRAY.
 * The function accepts INTEGER_ARRAY arr as parameter.
 */
int* countingSort(int arr_count, int* arr, int* result_count) {
   // The problem states that the values in `arr` are in the range 0 <= x < 100.
```

```
    // This means a frequency array of size 100 is needed to store the counts for each number from 0 to 99.
    int* frequency = (int*)calloc(100, sizeof(int));
    if (frequency == NULL) {
        // In a production environment, you would handle memory allocation failure.
        exit(EXIT_FAILURE);
    }

    // The size of the returned array is 100, as it will contain the frequency of numbers from 0 to 99.
    *result_count = 100;

    // Iterate through the input array to count the frequency of each number.
    for (int i = 0; i < arr_count; i++) {
        // Use the value of the array element `arr[i]` as an index to increment the count in the `frequency` array.
        // For example, if arr[i] is 5, frequency[5] is incremented.
        frequency[arr[i]]++;
    }

    return frequency;
}

int main()
{
    FILE* fptr = fopen(getenv("OUTPUT_PATH"), "w");

    int n = parse_int(ltrim(rtrim(readline())));

    char** arr_temp = split_string(rtrim(readline()));

    int* arr = malloc(n * sizeof(int));
    for (int i = 0; i < n; i++) {
        int arr_item = parse_int(*(arr_temp + i));

        *(arr + i) = arr_item;
    }

    int result_count;
    int* result = countingSort(n, arr, &result_count);

    for (int i = 0; i < result_count; i++) {
        fprintf(fptr, "%d", *(result + i));

        if (i != result_count - 1) {
            fprintf(fptr, " ");
        }
    }

    fprintf(fptr, "\n");

    fclose(fptr);

    return 0;
}

char* readline() {
    size_t alloc_length = 1024;
    size_t data_length = 0;

    char* data = malloc(alloc_length);
```

```
while (true) {
    char* cursor = data + data_length;
    char* line = fgets(cursor, alloc_length - data_length, stdin);

    if (!line) {
        break;
    }

    data_length += strlen(cursor);

    if (data_length < alloc_length - 1 || data[data_length - 1] == '\n') {
        break;
    }

    alloc_length <<= 1;

    data = realloc(data, alloc_length);

    if (!data) {
        data = '\0';

        break;
    }
}

if (data[data_length - 1] == '\n') {
    data[data_length - 1] = '\0';

    data = realloc(data, data_length);

    if (!data) {
            data = '\0';
        }
    } else {
        data = realloc(data, data_length + 1);

        if (!data) {
            data = '\0';
        } else {
            data[data_length] = '\0';
        }
    }

    return data;
}

char* ltrim(char* str) {
    if (!str) {
        return '\0';
    }

    if (!*str) {
        return str;
    }

    while (*str != '\0' && isspace(*str)) {
        str++;
    }

    return str;
}
```

```
char* rtrim(char* str) {
    if (!str) {
        return '\0';
    }

    if (!*str) {
        return str;
    }

    char* end = str + strlen(str) - 1;

    while (end >= str && isspace(*end)) {
        end--;
    }

    *(end + 1) = '\0';

    return str;
}

char** split_string(char* str) {
    char** splits = NULL;
    char* token = strtok(str, " ");

    int spaces = 0;

    while (token) {
        splits = realloc(splits, sizeof(char*) * ++spaces);

        if (!splits) {
            return splits;
        }

        splits[spaces - 1] = token;

        token = strtok(NULL, " ");
    }

    return splits;
}

int parse_int(char* str) {
    char* endptr;
    int value = strtol(str, &endptr, 10);

    if (endptr == str || *endptr != '\0') {
        exit(EXIT_FAILURE);
    }

    return value;
}
```

Pangrams

```
#include <assert.h>

#include <ctype.h>

#include <limits.h>

#include <math.h>

#include <stdbool.h>

#include <stddef.h>

#include <stdint.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>


char* readline();


/*

 * Complete the 'pangrams' function below.

 *

 * The function is expected to return a STRING.

 * The function accepts STRING s as parameter.

 */

char* pangrams(char* s) {

    // A boolean array to track the presence of each letter of the alphabet.

    bool alphabet_present[26] = {false};

    int letters_found_count = 0;


    // Iterate through the input string.

    for (int i = 0; s[i] != '\0'; i++) {
        char c = s[i];

        // Check if the character is an alphabet character.
        if (isalpha(c)) {
            // Convert the character to lowercase to handle both upper and lower case letters.
            char lower_c = tolower(c);

            // Map the character to an index from 0 to 25.
            int index = lower_c - 'a';

            // If the letter has not been marked as found yet, mark it and increment the counter.
            if (!alphabet_present[index]) {
                alphabet_present[index] = true;
                letters_found_count++;
            }
        }
    }

    // If the count of unique letters found is 26, it is a pangram.
    if (letters_found_count == 26) {
        // The problem expects a string to be returned.
        return "pangram";
    } else {
        return "not pangram";
    }
}

int main()
{
    FILE* fptr = fopen(getenv("OUTPUT_PATH"), "w");
    char* s = readline();

    char* result = pangrams(s);

    fprintf(fptr, "%s\n", result);

    fclose(fptr);

    return 0;
}

char* readline() {
    size_t alloc_length = 1024;
    size_t data_length = 0;

    char* data = malloc(alloc_length);

    while (true) {
        char* cursor = data + data_length;
        char* line = fgets(cursor, alloc_length - data_length, stdin);

        if (!line) {
            break;
        }

        data_length += strlen(cursor);

        if (data_length < alloc_length - 1 || data[data_length - 1] == '\n') {
            break;
        }
        alloc_length <<= 1;

        data = realloc(data, alloc_length);

        if (!data) {
            data = '\0';

            break;
        }
    }

    if (data[data_length - 1] == '\n') {
        data[data_length - 1] = '\0';

        data = realloc(data, data_length);

        if (!data) {
            data = '\0';
        }
    } else {
        data = realloc(data, data_length + 1);

        if (!data) {
            data = '\0';
        } else {
            data[data_length] = '\0';
        }
    }

    return data;
}
```

Permuting Two arrays

```
#include <assert.h>

#include <ctype.h>

#include <limits.h>

#include <math.h>

#include <stdbool.h>

#include <stddef.h>

#include <stdint.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>


char* readline();

char* ltrim(char*);

char* rtrim(char*);

char** split_string(char*);


int parse_int(char*);


// Comparison function for ascending sort

int compare_ascending(const void* a, const void* b) {

   return (*(int*)a - *(int*)b);

}


// Comparison function for descending sort

int compare_descending(const void* a, const void* b) {
    return (*(int*)b - *(int*)a);
}


/*
 * Complete the 'twoArrays' function below.
 *
 * The function is expected to return a STRING.
 * The function accepts following parameters:
 * 1. INTEGER k
 * 2. INTEGER_ARRAY A
 * 3. INTEGER_ARRAY B
 */
char* twoArrays(int k, int A_count, int* A, int B_count, int* B) {
    // Sort array A in ascending order.
    qsort(A, A_count, sizeof(int), compare_ascending);

    // Sort array B in descending order.
    qsort(B, B_count, sizeof(int), compare_descending);

    // Check if the condition A[i] + B[i] >= k is satisfied for all pairs.
    for (int i = 0; i < A_count; i++) {
        if (A[i] + B[i] < k) {
            // If the condition fails for any pair, it's not possible to achieve.
            // We can return "NO" immediately.
            return "NO";
        }
    }

    // If the loop completes, it means all pairs satisfy the condition.
    return "YES";
}
```

```
int main()
{
    FILE* fptr = fopen(getenv("OUTPUT_PATH"), "w");

    int q = parse_int(ltrim(rtrim(readline())));

    for (int q_itr = 0; q_itr < q; q_itr++) {
        char** first_multiple_input = split_string(rtrim(readline()));

        int n = parse_int(*(first_multiple_input + 0));

        int k = parse_int(*(first_multiple_input + 1));

        char** A_temp = split_string(rtrim(readline()));

        int* A = malloc(n * sizeof(int));

        for (int i = 0; i < n; i++) {
            int A_item = parse_int(*(A_temp + i));

            *(A + i) = A_item;
        }

        char** B_temp = split_string(rtrim(readline()));

        int* B = malloc(n * sizeof(int));

        for (int i = 0; i < n; i++) {
            int B_item = parse_int(*(B_temp + i));
            *(B + i) = B_item;
        }

        char* result = twoArrays(k, n, A, n, B);

        fprintf(fptr, "%s\n", result);
    }

    fclose(fptr);

    return 0;
}

char* readline() {
    size_t alloc_length = 1024;
    size_t data_length = 0;

    char* data = malloc(alloc_length);

    while (true) {
        char* cursor = data + data_length;
        char* line = fgets(cursor, alloc_length - data_length, stdin);

        if (!line) {
            break;
        }

        data_length += strlen(cursor);

        if (data_length < alloc_length - 1 || data[data_length - 1] == '\n') {
            break;
        }

        alloc_length <<= 1;

        data = realloc(data, alloc_length);

        if (!data) {
            data = '\0';

            break;
        }
    }

    if (data[data_length - 1] == '\n') {
        data[data_length - 1] = '\0';

        data = realloc(data, data_length);

        if (!data) {
            data = '\0';
        }
    } else {
        data = realloc(data, data_length + 1);

        if (!data) {
            data = '\0';
        } else {
            data[data_length] = '\0';
        }
    }
```

```
    return data;
}

char* ltrim(char* str) {
    if (!str) {
        return '\0';
    }

    if (!*str) {
        return str;
    }

    while (*str != '\0' && isspace(*str)) {
        str++;
    }

    return str;
}

char* rtrim(char* str) {
    if (!str) {
        return '\0';
    }

    if (!*str) {
        return str;
    }

    char* end = str + strlen(str) - 1;

    while (end >= str && isspace(*end)) {
        end--;
    }

    *(end + 1) = '\0';

    return str;
}

char** split_string(char* str) {
    char** splits = NULL;
    char* token = strtok(str, " ");

    int spaces = 0;

    while (token) {
        splits = realloc(splits, sizeof(char*) * ++spaces);

        if (!splits) {
            return splits;
        }

        splits[spaces - 1] = token;

        token = strtok(NULL, " ");
    }

    return splits;
}

int parse_int(char* str) {
    char* endptr;
    int value = strtol(str, &endptr, 10);


    if (endptr == str || *endptr != '\0') {
        exit(EXIT_FAILURE);
    }


    return value;
}
```

Subarray Division 1

```
#include <assert.h>
#include <ctype.h>
#include <limits.h>
#include <math.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char* readline();
char* ltrim(char*);
char* rtrim(char*);
char** split_string(char*);

int parse_int(char*);

/*
 * Complete the 'birthday' function below.
 *
 * The function is expected to return an INTEGER.
 * The function accepts following parameters:
 * 1. INTEGER_ARRAY s
 * 2. INTEGER d
 * 3. INTEGER m
 */
int birthday(int s_count, int* s, int d, int m) {
    if (m > s_count) {
        return 0;
    }

    int ways = 0;
    int current_sum = 0;

    // Calculate the sum of the first segment of length m.
    for (int i = 0; i < m; i++) {
        current_sum += s[i];
    }

    // Check the first segment.
    if (current_sum == d) {
        ways++;
    }

    // Use a sliding window to check the remaining segments.
    for (int i = m; i < s_count; i++) {
        // Update the sum by subtracting the element that is no longer in the window
        // and adding the new element that is now in the window.
        current_sum = current_sum - s[i - m] + s[i];

        if (current_sum == d) {
            ways++;
        }
    }
    return ways;
}

int main()
{
    FILE* fptr = fopen(getenv("OUTPUT_PATH"), "w");

    int n = parse_int(ltrim(rtrim(readline())));

    char** s_temp = split_string(rtrim(readline()));

    int* s = malloc(n * sizeof(int));

    for (int i = 0; i < n; i++) {
        int s_item = parse_int(*(s_temp + i));

        *(s + i) = s_item;
    }

    char** first_multiple_input = split_string(rtrim(readline()));

    int d = parse_int(*(first_multiple_input + 0));

    int m = parse_int(*(first_multiple_input + 1));

    int result = birthday(n, s, d, m);

    fprintf(fptr, "%d\n", result);

    fclose(fptr);
    return 0;
}

char* readline() {
    size_t alloc_length = 1024;
    size_t data_length = 0;

    char* data = malloc(alloc_length);

    while (true) {
        char* cursor = data + data_length;
        char* line = fgets(cursor, alloc_length - data_length, stdin);

        if (!line) {
            break;
        }

        data_length += strlen(cursor);

        if (data_length < alloc_length - 1 || data[data_length - 1] == '\n') {
            break;
        }

        alloc_length <<= 1;

        data = realloc(data, alloc_length);

        if (!data) {
            data = '\0';

            break;
        }
    }

    if (data[data_length - 1] == '\n') {
        data[data_length - 1] = '\0';

        data = realloc(data, data_length);

        if (!data) {
            data = '\0';
        }
    } else {
        data = realloc(data, data_length + 1);

        if (!data) {
            data = '\0';
        } else {
            data[data_length] = '\0';
        }
    }

    return data;
}

char* ltrim(char* str) {
    if (!str) {
        return '\0';
    }

    if (!*str) {
        return str;
    }

    while (*str != '\0' && isspace(*str)) {
        str++;
    }

    return str;
}

char* rtrim(char* str) {
    if (!str) {
        return '\0';
    }

    if (!*str) {
        return str;
    }

    char* end = str + strlen(str) - 1;

    while (end >= str && isspace(*end)) {
        end--;
    }

    *(end + 1) = '\0';

    return str;
}

char** split_string(char* str) {
    char** splits = NULL;
    char* token = strtok(str, " ");

    int spaces = 0;

    while (token) {
        splits = realloc(splits, sizeof(char*) * ++spaces);

        if (!splits) {
            return splits;
        }

        splits[spaces - 1] = token;

        token = strtok(NULL, " ");
    }

    return splits;
}

int parse_int(char* str) {
    char* endptr;
    int value = strtol(str, &endptr, 10);

    if (endptr == str || *endptr != '\0') {
        exit(EXIT_FAILURE);
    }

    return value;
}
```

XOR String 2

```
#include <cmath>
#include <cstdio>
#include <vector>
#include <iostream>
#include <algorithm>
using namespace std;

string strings_xor(string s, string t) {

    string res = "";
    for(int i = 0; i < s.size(); i++) {
        if(s[i]== t[i])
            res += '0';
        else
            res += '1';
    }

    return res;
}

int main() {
    string s, t;
    cin >> s >> t;
    cout << strings_xor(s, t) << endl;
    return 0;
}
```