RECURSSION:-

**Problem Statement**

During an archaeological expedition, a team discovers the "Disk of Light," an artifact consisting of concentric rings with unique symbols. To unlock its powers, they must perform the "Tower of Wisdom" ritual.

In this ritual:

- There are three pedestals: Source (A), Auxiliary (B), and Illuminated (C).
- Disks are stacked on the Source pedestal, smaller ones on top of larger ones.
- Only one disk can be moved at a time.
- A disk can only be placed on top of a larger disk or an empty pedestal.

Write a program to assist the archaeological team in calculating the total number of times each ring of the Disk of Light is moved during the Tower of Wisdom ritual.

**Note:** This kind of question will be helpful in clearing TCS recruitment.

**Input format :**

The input consists of an integer denoting the number of disks.

**Output format :**

The output displays the sequence of moves required to solve the puzzle.

The last line of output displays the total number of times the disk is moved.

**Refer to the sample output for formatting specifications.**

**Code constraints :**

0 < n <= 8

**Sample test cases :**

**Input 1 :**

```
3
```

**Output 1 :**

```
Move disk 1 from A to C
Move disk 2 from A to B
Move disk 1 from C to B
Move disk 3 from A to C
Move disk 1 from B to A
Move disk 2 from B to C
Move disk 1 from A to C
Total number of times the disk is moved: 7
```

**Input 2 :**

```
4
```

**Output 2 :**

```
Move disk 1 from A to B
Move disk 2 from A to C
Move disk 1 from B to C
Move disk 3 from A to B
Move disk 1 from C to A
```

```
Move disk 2 from C to B
Move disk 1 from A to B
Move disk 4 from A to C
Move disk 1 from B to C
Move disk 2 from B to A
Move disk 1 from C to A
Move disk 3 from B to C
Move disk 1 from A to B
Move disk 2 from A to C
Move disk 1 from B to C
Total number of times the disk is moved: 15
```

```
#include <iostream>
using namespace std;

void towerOfHanoiMoves(int n, char source, char destination, char auxiliary, int &total_moves) {
   if (n == 0) {
      return;
   }

   // Move (n-1) disks from source to auxiliary
   towerOfHanoiMoves(n - 1, source, auxiliary, destination, total_moves);

   // Move the nth disk from source to destination
   cout << "Move disk " << n << " from " << source << " to " << destination << endl;
   total_moves++;

   // Move the (n-1) disks from auxiliary to destination
   towerOfHanoiMoves(n - 1, auxiliary, destination, source, total_moves);
}

int main() {
   int n;
   cin >> n;
```

```
    int total_moves = 0;
    towerOfHanoiMoves(n, 'A', 'C', 'B', total_moves);

    cout << "Total number of times the disk is moved: " << total_moves << endl;

    return 0;
}
```

**Problem Statement**

You work in a large warehouse where products are numbered from 0 to N-1. Your task is to identify any products that appear twice in the inventory list and report them in ascending order. This helps ensure accurate inventory management.

For instance, if you have N = 4 products with IDs {0, 1, 3, 2}, and no duplicates exist, you report -1. However, if you have N = 5 products with IDs {1, 1, 2, 2, 3}, you must report the duplicates in ascending order: 1 2. This ensures your warehouse maintains efficient inventory control. The input products must be listed in ascending order.

Now design a program using recursion to solve the given problem statement.

**Input format :**

The first line contains an integer N, the size of the array.
The second line contains N space-separated integers a[0], a[1], ..., a[N-1], representing the elements in the array in ascending order.

**Output format :**

The output should be a space-separated list of integers.
If there are elements that occur twice in the given array, list them in ascending order.
If no such elements are found, the output should be "-1" to indicate that there are no duplicates.

**Refer to the sample output for exact format specifications.**

**Code constraints :**

1 <= N <= 25
0 <= a[i] <= N-1, for each valid i. The input should be in ascending order.

**Sample test cases :**

**Input 1 :**

```
4
0 1 3 2
```

**Output 1 :**

```
-1
```

**Input 2 :**

```
5
1 1 2 2 3
```

**Output 2 :**

```
1 2
```

```
#include <stdio.h>

// Function to find and print duplicate elements in a sorted array
int findAndPrintDuplicates(int arr[], int n, int prev) {
    // Base case: If the array has only one element, it cannot have duplicates
    if (n == 1) {
        if (arr[0] == prev) {
            printf("%d ", arr[0]);
            return arr[0];
        }
        return -1;
    }

    // Recursive case
    int current = arr[n - 1];
    int result = findAndPrintDuplicates(arr, n - 1, current);

    // Check if the current element is a duplicate
    if (current == prev) {
        printf("%d ", current);
        return current;
    }

    return result;
}

int main() {
```

```
    int n;
    //printf("Enter the number of elements: ");
    scanf("%d", &n);

    int arr[n];

    //printf("Enter %d sorted elements: ", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    int result = findAndPrintDuplicates(arr, n, -1);

    if (result == -1) {
        printf("-1");
    }

    return 0;
}
```

MERGE SORT:-

**Problem Statement**

At the "Fresh Mart" grocery store, they recently received a large shipment of various products, and they need to organize their inventory efficiently. They've hired you to help them sort the products in descending order based on their product IDs.

Develop a program to aid the Fresh Mart team in organizing their inventory using a recursive function and implementing the merge sort logic. Each product in the store is identified by a distinct product ID, and the objective is to present the products to customers in descending order.

**Input format :**

The first line consists of an integer, n, representing the number of products in the inventory.

The next line consists of n space-separated integers, each representing a unique Product ID.

**Output format :**

The output displays a single line containing the Product IDs sorted in descending order, separated by spaces.

**Code constraints :**

$1 \leq n \leq 10$

$1 \leq$ Product ID $\leq 1000$

**Sample test cases :**

**Input 1 :**

```
5
1 9 7 6 8
```

**Output 1 :**

```
9 8 7 6 1
```

**Input 2 :**

```
4
754 312 505 44
```

**Output 2 :**

```
754 505 312 44
```

```
#include <stdio.h>


// A function to merge the two halves into a sorted data.
void Merge(int *a, int low, int high, int mid) {
   int i, j, k, temp[high - low + 1];
   i = low;
   k = 0;
   j = mid + 1;

   // Merge the two parts into temp[].
   while (i <= mid && j <= high) {
      if (a[i] < a[j]) {
         temp[k] = a[i];
         k++;
         i++;
      } else {
         temp[k] = a[j];
```

```
            k++;
            j++;
        }
    }

    // Insert all the remaining values from i to mid into temp[].
    while (i <= mid) {
        temp[k] = a[i];
        k++;
        i++;
    }

    // Insert all the remaining values from j to high into temp[].
    while (j <= high) {
        temp[k] = a[j];
        k++;
        j++;
    }

    // Assign sorted data stored in temp[] to a[].
    for (i = low; i <= high; i++) {
        a[i] = temp[i - low];
    }
}

// A function to split the array into two parts.
void MergeSort(int *a, int low, int high) {
    int mid;
    if (low < high) {
        mid = (low + high) / 2;
        // Split the data into two halves.
```

```
        MergeSort(a, low, mid);
        MergeSort(a, mid + 1, high);

        // Merge them to get sorted output.
        Merge(a, low, high, mid);
    }
}

int main() {
    int n, i;
    scanf("%d", &n);

    int arr[n];
    for (i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    MergeSort(arr, 0, n - 1);
    for (i = (n - 1); i >= 0; i--) {
        printf("%d ", arr[i]);
    }

    return 0;
}
```

**Problem Statement**

Arsha is working on a project where binary data is an essential part of the processing. To optimize data handling, Arsha needs to sort an array containing only two types of elements: 0s and 1s. Alex decides to implement a merge sort algorithm to efficiently sort this binary data.

Write a program to assist Arsha in implementing the logic of merge sort along with a recursive function to sort an array of binary data in ascending order.

**Input format :**

The first line contains an integer, representing the number of elements in the array.

The second line contains n space-separated integers, where each integer is either 0 or 1.

**Output format :**

The output displays the following result:

- If the input contains elements other than 0 and 1, print "Invalid input.".
- Otherwise, print a single line containing n space-separated integers, representing the sorted array in ascending order.

**Refer to the sample outputs for the exact format.**

**Code constraints :**

$1 \leq n \leq 20$

input = 0, and 1 only

**Sample test cases :**

**Input 1 :**

```
5
1 0 1 0 1
```

**Output 1 :**

```
0 0 1 1 1
```

**Input 2 :**

```
3
1 0 2
```

**Output 2 :**

```
Invalid input.
```

**Input 3 :**

```
4
1 1 1 1
```

**Output 3 :**

```
1 1 1 1
```

**Input 4 :**

```
2
0 0
```

**Output 4 :**

```
0 0
```

```
#include <iostream>

using namespace std;


void merge(int arr[], int left, int mid, int right) {
```

```
int n1 = mid - left + 1;
int n2 = right - mid;

int L[n1], R[n2];

for (int i = 0; i < n1; i++)
    L[i] = arr[left + i];
for (int j = 0; j < n2; j++)
    R[j] = arr[mid + 1 + j];

int i = 0, j = 0, k = left;

while (i < n1 && j < n2) {
    if (L[i] <= R[j]) {
        arr[k] = L[i];
        i++;
    } else {
        arr[k] = R[j];
        j++;
    }
    k++;
}

while (i < n1) {
    arr[k] = L[i];
    i++;
    k++;
}
while (j < n2) {
    arr[k] = R[j];
    j++;
```

```
        k++;
    }
}

void mergeSort(int arr[], int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;

        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);

        merge(arr, left, mid, right);
    }
}

int main() {
    int n;
    cin >> n;

    int arr[n];

    for (int i = 0; i < n; i++) {
        cin >> arr[i];
        if (arr[i] != 0 && arr[i] != 1) {
            cout << "Invalid input." << endl;
            return 0;
        }
    }

    mergeSort(arr, 0, n - 1);
```

```
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    return 0;
}
```

**Problem Statement**

In a mystical land known as Eldoria, ancient wizards use magical runes to cast powerful spells. These runes are represented by single characters, each possessing a unique magical property. However, the wizards have a challenge: they need these magical runes sorted in a specific order for their spells to work correctly.

Write a program to help the wizards of Eldoria sort their magical runes based on their potency. Each rune is represented by a single character, and each character holds a unique level of magical power, determined by its position in the ASCII table.

Your task is to implement a merge sorting logic and recursive function to arrange these magical runes in descending order of their magical potency.

**Input format :**

The first line of input consists of an integer n, representing the number of magical runes to be sorted.

The second line contains n space-separated characters, each representing a magical rune.

**Output format :**

The output displays a single line containing the magical runes sorted in descending order of their magical potency, separated by spaces.

**Refer to the sample output for the formatting specifications.**

**Code constraints :**

1 <= n <= 25

Each character in the array is a single uppercase letter or a single lowercase letter.

**Sample test cases :**

**Input 1 :**

```
8
G h I J K L m n
```

**Output 1 :**

```
n m h L K J I G
```

**Input 2 :**

```
6
a Z A B M Z
```

**Output 2 :**

```
a Z Z M B A
```

```
#include <iostream>

using namespace std;


const int MAX_ARRAY_LENGTH = 100; // Maximum array length


void merge_descending(char arr[], int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;


    char left_arr[n1];
    char right_arr[n2];


    for (int i = 0; i < n1; i++)
        left_arr[i] = arr[left + i];
    for (int j = 0; j < n2; j++)
        right_arr[j] = arr[mid + 1 + j];


    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        if (left_arr[i] >= right_arr[j]) {  // Compare in descending order
            arr[k] = left_arr[i];
            i++;
        } else {
            arr[k] = right_arr[j];
            j++;
        }
        k++;
```

```
    }

    while (i < n1) {
        arr[k] = left_arr[i];
        i++;
        k++;
    }

    while (j < n2) {
        arr[k] = right_arr[j];
        j++;
        k++;
    }
}

void mergeSortDescending(char arr[], int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;
        mergeSortDescending(arr, left, mid);
        mergeSortDescending(arr, mid + 1, right);
        merge_descending(arr, left, mid, right);
    }
}

int main() {
    int n;
    cin >> n;

    char arr[MAX_ARRAY_LENGTH];
    for (int i = 0; i < n; i++)
        cin >> arr[i]; // Read single characters
```

```
    mergeSortDescending(arr, 0, n - 1);

    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";

    return 0;
}
```

**Problem Statement**

Raj wants to learn a sequence of integers. His task is to count the frequency of each unique integer in the sequence and then sort them based on their frequencies in ascending order. If two integers have the same frequency, sort them in ascending order of their values.

For this goal, your task is to implement the logic of the merge sort and a recursive function.

**Example 1**

**Input:**
nums = [1, 1, 2, 2, 2, 3]

**Output:**
[3,1,1,2,2,2]

**Explanation:**
'3' has a frequency of 1, '1' has a frequency of 2, and '2' has a frequency of 3.

**Example 2**

**Input:**
nums = [3, 2, 1, 3, 2]

**Output:**
[1,2,2,3,3]

**Explanation:**

'3' and '2' both have a frequency of 2, so they are sorted based on their actual values, with the smaller value coming first.

**Input format :**

The first line of input contains an integer, n, representing the number of integers in the sequence.

The second line of input contains n space-separated integers, arr [i], representing the elements of the sequence.

**Output format :**

The output displays a single line containing the sorted integers separated by space according to their frequencies and values.

**Code constraints :**

max_n = 100

1 <= n <= 25

1 <= arr[i] <= 100

**Sample test cases :**

**Input 1 :**

```
6
1 1 2 2 2 3
```

**Output 1 :**

```
3 1 1 2 2 2
```

**Input 2 :**

```
5
3 2 1 3 2
```

**Output 2 :**

```
1 2 2 3 3
```

```
#include <iostream>

using namespace std;


const int MAX_N = 100;


void merge(int arr[], int temp[], int left, int mid, int right) {

    int n1 = mid - left + 1;

    int n2 = right - mid;


    int L[n1], R[n2];


    for (int i = 0; i < n1; i++) L[i] = arr[left + i];
```

```
    for (int j = 0; j < n2; j++) R[j] = arr[mid + 1 + j];

    int i = 0, j = 0, k = left;

    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k++] = L[i++];
        } else {
            arr[k++] = R[j++];
        }
    }

    while (i < n1) arr[k++] = L[i++];
    while (j < n2) arr[k++] = R[j++];
}

void mergeSort(int arr[], int temp[], int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;

        mergeSort(arr, temp, left, mid);
        mergeSort(arr, temp, mid + 1, right);
        merge(arr, temp, left, mid, right);
    }
}

void countAndSortFrequency(int arr[], int n) {
    int freqArr[MAX_N][2] = {0};
    int values[MAX_N];
    int frequencies[MAX_N];
```

```
    int index = 0;
    for (int i = 0; i < n; i++) {
        int isAlreadyCounted = 0;
        for (int j = 0; j < index; j++) {
            if (values[j] == arr[i]) {
                freqArr[j][1]++;
                isAlreadyCounted = 1;
                break;
            }
        }
        if (!isAlreadyCounted) {
            values[index] = arr[i];
            freqArr[index][0] = arr[i];
            freqArr[index][1] = 1;
            index++;
        }
    }

    for (int i = 0; i < index; i++) {
        for (int j = i + 1; j < index; j++) {
            if (freqArr[i][1] > freqArr[j][1] || (freqArr[i][1] == freqArr[j][1] && freqArr[i][0] > freqArr[j][0]))
{
                swap(freqArr[i][0], freqArr[j][0]);
                swap(freqArr[i][1], freqArr[j][1]);
            }
        }
    }

    for (int i = 0; i < index; i++) {
        for (int j = 0; j < freqArr[i][1]; j++) {
            cout << freqArr[i][0] << " ";
```

```
        }
    }
    cout << endl;
}

int main() {
    int n;
    cin >> n;
    int arr[MAX_N];

    for (int i = 0; i < n; i++) cin >> arr[i];

    countAndSortFrequency(arr, n);

    return 0;
}
```

---

QUICK SORT:-

**Problem Statement**

Imagine you are a computer programmer tasked with creating a program to organize a collection of years in ascending order. This program will take a list of years as input and efficiently sort them using the **Quick-Sort** algorithm, ensuring that the years are arranged chronologically.

**Input format :**

The first line of input consists of an integer **N**, representing the number of years.
The second line consists of **N** space-separated integers, representing the years.

**Output format :**

The output prints the sorted dates in chronological order.

**Code constraints :**

N > 0
Each year is a 4-digit positive integer.

**Sample test cases :**

**Input 1 :**

```
5
```

```
2014 2009 2000 1997 1865
```

**Output 1 :**

```
1865 1997 2000 2009 2014
```

**Input 2 :**

```
3
1496 3015 2056
```

**Output 2 :**

```
1496 2056 3015
```

```
#include <iostream>

#include <string>


using namespace std;


int partition(string arr[], int low, int high) {

   string pivot = arr[high];

   int i = low - 1;


   for (int j = low; j < high; j++) {

      if (arr[j] < pivot) {

         i++;

         swap(arr[i], arr[j]);

      }

   }


   swap(arr[i + 1], arr[high]);


   return i + 1;

}


void quickSort(string arr[], int low, int high) {

   if (low < high) {

      int pi = partition(arr, low, high);
```

```
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

void recursiveQuickSort(string arr[], int n) {
    quickSort(arr, 0, n - 1);
}

int main() {
    int n;
    cin >> n;

    string years[n];

    for (int i = 0; i < n; i++) {
        cin >> years[i];
    }

    recursiveQuickSort(years, n);

    for (int i = 0; i < n; i++) {
        cout << years[i] << " ";
    }

    return 0;
}
```

**Problem Statement**

You are working on a project to implement a feature that displays the list of users in alphabetical order based on their names.

To achieve this, you decide to use the **Quick-Sort** algorithm to efficiently sort the names. The user inputs the number of users and then the name of each user. Once the names are collected, the program must apply the Quick Sort algorithm to sort and display the names in alphabetical order.

**Note:** This kind of question will be helpful in clearing Capgemini recruitment.

**Input format :**

The first line of input consists of an integer **N**, representing the number of users.
The following **N** lines consist of the names of the users (starting with uppercase letters).

**Output format :**

The output prints the sorted list of names in alphabetical order.

**Code constraints :**

N > 0

**Sample test cases :**

**Input 1 :**

```
5
Alice
Denver
Aadhil
Charlie
Zen
```

**Output 1 :**

```
Aadhil
Alice
Charlie
Denver
Zen
```

**Input 2 :**

```
6
Chandler
Monica
Ross
Joey
Rachel
Phoebe
```

**Output 2 :**

```
Chandler
Joey
Monica
Phoebe
Rachel
Ross
```

```
#include <iostream>
#include <string>

using namespace std;

int partition(string names[], int low, int high);

void quickSort(string names[], int low, int high) {
   if (low < high) {
      int pi = partition(names, low, high);
      quickSort(names, low, pi - 1);
      quickSort(names, pi + 1, high);
   }
}

int partition(string names[], int low, int high) {
   string pivot = names[high];
   int i = low - 1;

   for (int j = low; j < high; j++) {
      if (names[j].compare(pivot) < 0) {
         i++;
         swap(names[i], names[j]);
      }
   }

   swap(names[i + 1], names[high]);
   return i + 1;
}

int main() {
```

```
    int n;
    cin >> n;

    string names[n];

    for (int i = 0; i < n; i++) {
        cin >> names[i];
    }

    quickSort(names, 0, n - 1);

    for (int i = 0; i < n; i++) {
        cout << names[i] << endl;
    }

    return 0;
}
```

**Problem Statement**

As a diligent teacher, you have conducted a challenging exam for your students. Now that the papers have been graded, you are eager to find out which students performed the best and identify those who might need extra support.

To efficiently rank your students based on their marks, you decide to create a computer program that sorts their exam marks in descending order using the **Quick-Sort** algorithm.

**Input format :**

The first line of input consists of an integer **N**, representing the number of students.
The second line consists of **N** floating-point numbers, representing the marks of the students.

**Output format :**

The output prints the marks in descending order, placing the highest mark first and the lowest mark last.

**Code constraints :**

N > 0

Marks > 0.0

**Sample test cases :**

**Input 1 :**

```
5
78.3 54.7 96.4 32.7 53.1
```

**Output 1 :**

```
96.4 78.3 54.7 53.1 32.7
```

**Input 2 :**

```
4
50.6 65.1 84.3 36.4
```

**Output 2 :**

```
84.3 65.1 50.6 36.4
```

```
#include <iostream>
#include <iomanip> // Required for std::fixed and std::setprecision

using namespace std;

void printArray(float arr[], int size) {
    for (int i = 0; i < size; i++) {
        cout << fixed << setprecision(1) << arr[i] << " ";
    }
}

int partition(float arr[], int low, int high) {
    float pivot = arr[high];
    int i = low - 1;
    for (int j = low; j < high; j++) {
        if (arr[j] >= pivot) {
            i++;
            swap(arr[i], arr[j]);
        }
    }
    swap(arr[i + 1], arr[high]);
    return i + 1;
```

```
}

void quickSort(float arr[], int low, int high) {
    if (low < high) {
        int pivot = partition(arr, low, high);
        quickSort(arr, low, pivot - 1);
        quickSort(arr, pivot + 1, high);
    }
}

int main() {
    int n;
    cin >> n;

    float* marks = new float[n];
    for (int i = 0; i < n; i++) {
        cin >> marks[i];
    }

    quickSort(marks, 0, n - 1);

    printArray(marks, n);

    delete[] marks;
    return 0;
}
```

**Problem Statement**

You are working as a software developer at a language processing company. As part of the company's text processing system, you need to sort a collection of words in reverse lexicographical (dictionary) order using the **Quick-Sort** algorithm.

Write a program that takes an array of words as input and uses the QuickSort algorithm to sort the words in reverse lexicographical order.

**Input format :**

The first line of input consists of an integer **N**, representing the number of strings.

The second line consists of **N** strings, separated by space.

**Output format :**

The output displays the words sorted in reverse lexicographical order.

**Code constraints :**

N > 0

**Sample test cases :**

**Input 1 :**

```
2
Cap Cat
```

**Output 1 :**

```
Cat Cap
```

**Input 2 :**

```
5
Chair Door Mouse Keyboard Table
```

**Output 2 :**

```
Table Mouse Keyboard Door Chair
```

```
#include <iostream>
#include <string>

using namespace std;

int partition(string arr[], int low, int high) {
    string pivot = arr[high];
    int i = low - 1;
    for (int j = low; j < high; j++) {
        if (arr[j] >= pivot) { // Change the comparison condition to sort in descending order
            i++;
            swap(arr[i], arr[j]);
        }
    }
    swap(arr[i + 1], arr[high]);
```

```
    return i + 1;
}

void quickSort(string arr[], int low, int high) {
    if (low < high) {
        int pivot = partition(arr, low, high);
        quickSort(arr, low, pivot - 1);
        quickSort(arr, pivot + 1, high);
    }
}

int main() {
    int n;
    cin >> n;

    string *words = new string[n];
    for (int i = 0; i < n; i++) {
        cin >> words[i];
    }

    quickSort(words, 0, n - 1);

    for (int i = 0; i < n; i++) {
        cout << words[i] << " ";
    }

    delete[] words;
    return 0;
}
```