Problem Statement 1

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>
#include<vector>
using namespace std;

void merge(vector<int>& arr, int low, int mid, int high){
 vector<int> temp;
 int left = low;
 int right = mid +1;

 while(left <= mid && right <= high){
  if(arr[left] <= arr[right]){
   temp.push_back(arr[left]);
   left++;
  }else{
   temp.push_back(arr[right]);
   right++;
  }
 }

 while(left <= mid){
```

```
    temp.push_back(arr[left]);
    left++;
  }

  while(right <= high){
    temp.push_back(arr[right]);
    right++;
  }

  for(int i=low; i<= high; i++){
    arr[i] = temp[i - low];
  }
}

void mergeSort(vector<int>& arr, int low, int high){
  if(low == high){
    return;
  }

  int mid = (low+high)/2;
  mergeSort(arr, low, mid);
  mergeSort(arr, mid+1, high);
  merge(arr, low, mid, high);
}

int main(){
  int n;
  cin>>n;
  bool valid = true;

  vector<int> arr(n);
```

```
 for(int i=0; i<n; i++){
  cin>>arr[i];
  if(arr[i] != 0 && arr[i] != 1){
   valid = false;
  }
 }

 if(!valid){
  cout<<"Invalid input"<<endl;
 }else{
  mergeSort(arr, 0, n-1);
  for(int i = 0; i<n; i++){
   cout<<arr[i]<<" ";
  }
  cout<<endl;
 }

 return 0;
}
```

Problem Statement 2

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>
#include<vector>
using namespace std;

void merge(vector<char>& arr, int low, int mid, int high){
  vector<char> temp;
  int left = low;
  int right = mid+1;
```

```
  while(left<= mid && right<= high){
    if(arr[left] >= arr[right]){
      temp.push_back(arr[left]);
      left++;
    }else{
      temp.push_back(arr[right]);
      right++;
    }
  }

  while(left <= mid){
    temp.push_back(arr[left]);
    left++;
  }

  while(right<= high){
    temp.push_back(arr[right]);
    right++;
  }

  for(int i=low; i<= high; i++){
    arr[i] = temp[i-low];
  }
}

void mergeSort(vector<char>& arr, int low, int high){
  if(low == high){
    return;
  }
```

```
  int mid = (low+high)/2;
  mergeSort(arr, low, mid);
  mergeSort(arr, mid+1, high);
  merge(arr, low, mid, high);
}

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

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

  mergeSort(arr, 0, n-1);

  for(char a : arr){
    cout<<a<<" ";
  }
  cout<<endl;

  return 0;
}
```

Problem Statement 3

You are working for an e-commerce company that wants to analyze customer purchase data. The company is interested in understanding the frequency of items purchased by customers and then displaying these items sorted by their purchase frequency. If two items have the same frequency, the item with the higher value should come first in the output.

Given a list of purchased items, your task is to count the frequency of each item and then sort these items based on their frequency in descending order using the **merge sort algorithm**. If two items have the same frequency, sort them in descending order based on their values.

Example

Input

6

1 1 2 3 3 3

Output

3 3 3 1 1 2

Explanation

- Item 3 appears the most frequently (3 times), so it comes first.
- Item 1 appears next (2 times).
- Item 2 appears last (1 time).

Each item is repeated according to its frequency, and the overall ordering is consistent with the frequency and value sorting criteria.

Input format :

The first line of input contains an integer n, the number of items purchased.

The second line contains n integers where each integer represents an item purchased.

Output format :

The output displays the items in a single line separated by spaces. The output should display items sorted first by their frequency (in descending order), and for items with the same frequency, by their values (also in descending order).

Code constraints :

max_n = 100

$1 \leq n \leq 25$

1 <= arr[i] <= 100

Sample test cases :

Input 1 :

6

1 1 2 3 3 3

Output 1 :

3 3 3 1 1 2

Input 2 :

7

2 2 3 1 3 2 3

Output 2 :

3 3 3 2 2 2 1

Input 3 :

6

1 2 3 1 2 3

Output 3 :

3 3 2 2 1 1

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


struct Item{
int value;
int frequency;
};

void merge(vector<Item>& arr, int low, int mid, int high){
 vector<Item> temp;
 int left = low;
 int right = mid+1;

 while(left<= mid && right <= high){
  if(arr[left].frequency > arr[right].frequency){
    temp.push_back(arr[left]);
    left++;
  }else if(arr[left].frequency < arr[right].frequency){
```

```
    temp.push_back(arr[right]);
    right++;
   }else{
    if(arr[left].value > arr[right].value){
     temp.push_back(arr[left]);
     left++;
    }else{
     temp.push_back(arr[right]);
     right++;
    }
   }
  }

  while(left<= mid){
   temp.push_back(arr[left]);
   left++;
  }

  while(right <= high){
   temp.push_back(arr[right]);
   right++;
  }

  for(int i=low; i<=high; i++){
   arr[i] = temp[i-low];
  }
}

void mergeSort(vector<Item>& arr, int low, int high){
  if(low == high){
   return;
```

```
  }
  int mid = (low+high)/2;
  mergeSort(arr, low, mid);
  mergeSort(arr, mid+1, high);
  merge(arr, low, mid, high);
}

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

  vector<int> arr(n);
  unordered_map<int, int> freq;
  for(int i=0; i<n; i++){
    cin>>arr[i];
    freq[arr[i]]++;
  }

  vector<Item> items;
  for(auto& it: freq){
    items.push_back({it.first, it.second});
  }

  mergeSort(items, 0, items.size()-1);

  for(auto& item : items){
    for(int i=0; i<item.frequency; i++){
      cout<<item.value<<" ";
    }
  }
  cout<<endl;
```

```
 return 0;
}
```

Problem Statement 4

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, nums[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 :

1 <= n <= 25

1 <= nums[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>
#include<vector>
#include<unordered_map>
using namespace std;


struct Item{
int value;
int frequency;
};

void merge(vector<Item>& arr, int low, int mid, int high){
  vector<Item> temp;
```

```
int left = low;
int right = mid+1;

while(left<= mid && right <= high){
  if(arr[left].frequency < arr[right].frequency){
    temp.push_back(arr[left]);
    left++;
  }else if(arr[left].frequency > arr[right].frequency){
    temp.push_back(arr[right]);
    right++;
  }else{
    if(arr[left].value < arr[right].value){
      temp.push_back(arr[left]);
      left++;
    }else{
      temp.push_back(arr[right]);
      right++;
    }
  }
}

while(left<= mid){
  temp.push_back(arr[left]);
  left++;
}

while(right <= high){
  temp.push_back(arr[right]);
  right++;
}
```

```
  for(int i=low; i<=high; i++){
    arr[i] = temp[i-low];
  }
}

void mergeSort(vector<Item>& arr, int low, int high){
  if(low == high){
    return;
  }
  int mid = (low+high)/2;
  mergeSort(arr, low, mid);
  mergeSort(arr, mid+1, high);
  merge(arr, low, mid, high);
}

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

  vector<int> arr(n);
  unordered_map<int, int> freq;
  for(int i=0; i<n; i++){
    cin>>arr[i];
    freq[arr[i]]++;
  }

  vector<Item> items;
  for(auto& it: freq){
    items.push_back({it.first, it.second});
  }
```

```
 mergeSort(items, 0, items.size()-1);

 for(auto& item : items){
  for(int i=0; i<item.frequency; i++){
   cout<<item.value<<" ";
  }
 }
 cout<<endl;


 return 0;
}
```

Problem Statement 5

You are a student attending a class on sorting techniques taught by Professor Smith. Today, Professor Smith is explaining how to efficiently sort both positive and negative values using the merge sort algorithm and a recursive function. To better understand the concept, you decide to write a program.

Your task is to write code to implement a recursive merge sort algorithm for sorting positive and negative floating-point numbers, as explained by Professor Smith.

Input format :

The first line contains an integer n, representing the number of values to be sorted.

The second line contains n space-separated floating-point numbers.

Output format :

The output displays a single line containing n space-separated floating-point numbers, representing the sorted values in ascending order. Each number is rounded to two decimal places.

Refer to the sample output for formatting specifications.

Code constraints :

$1 \leq n \leq 15$

$-100.0 \leq$ floating point numbers ≤ 100.0

Sample test cases :

Input 1 :

5

3.14 1.1 2.71 0.5 1.618

Output 1 :

Sorted Array:

0.50 1.10 1.62 2.71 3.14

Input 2 :

6

-2.5 5.0 0.0 -1.5 2.0 3.5

Output 2 :

Sorted Array:

-2.50 -1.50 0.00 2.00 3.50 5.00

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

void merge(vector<float>& arr, int low, int mid, int high){
 vector<float> temp;
 int left = low;
 int right = mid+1;
 while(left <= mid && right <= high){
  if(arr[left] <= arr[right]){
   temp.push_back(arr[left]);
   left++;
  }else{
   temp.push_back(arr[right]);
   right++;
  }
 }
```

```
  while(left <= mid){
    temp.push_back(arr[left]);
    left++;
  }

  while(right <= high){
    temp.push_back(arr[right]);
    right++;
  }

  for(int i=low; i<=high; i++){
    arr[i] = temp[i-low];
  }
}

void mergeSort(vector<float>& arr, int low, int high){
  if(low == high){
    return;
  }
  int mid = (low+high)/2;
  mergeSort(arr, low, mid);
  mergeSort(arr, mid+1, high);
  merge(arr, low, mid, high);
}

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

  vector<float> arr(n);
```

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

 mergeSort(arr, 0, n-1);
cout<<”Sorted Array:”<<endl;
 for(int i=0; i<n;i++){
   cout<<fixed<<setprecision(2)<<arr[i]<<" ";
 }

 return 0;
}
```

Problem Statement 6

John is a student who just received his test scores (floating-point numbers) for various subjects. He wants to organize his scores in descending order so that he can see his highest scores first. Can you help him write a program to sort his test scores in descending order?

Your task is to write a program to sort John's test scores in descending order using the merge sort algorithm and a recursive function.

Input format :

The first line contains an integer, n, representing the number of test scores John has received.

The next line contains n floating-point numbers separated by spaces, each representing John's test score.

Output format :

The output displays the sorted test scores in descending order, each rounded to two decimal places.

Refer to the sample outputs for the exact format.

Code constraints :

$1 \leq n \leq 15$

0.0 <= test score <= 100.0

Sample test cases :

Input 1 :

4

9.8 1.1 3.3 7.7

Output 1 :

Sorted Array:

9.80 7.70 3.30 1.10

Input 2 :

5

3.14 1.1 2.71 0.5 1.618

Output 2 :

Sorted Array:

3.14 2.71 1.62 1.10 0.50

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

void merge(vector<float>& arr, int low, int mid, int high){
 vector<float> temp;
 int left = low;
 int right = mid+1;
 while(left <= mid && right <= high){
  if(arr[left] >= arr[right]){
   temp.push_back(arr[left]);
   left++;
  }else{
   temp.push_back(arr[right]);
   right++;
  }
 }
```

```
  while(left <= mid){
    temp.push_back(arr[left]);
    left++;
  }

  while(right <= high){
    temp.push_back(arr[right]);
    right++;
  }

  for(int i=low; i<= high; i++){
    arr[i] = temp[i-low];
  }
}

void mergeSort(vector<float>& arr, int low, int high){
  if(low == high){
    return;
  }

  int mid = (low+high)/2;
  mergeSort(arr, low, mid);
  mergeSort(arr, mid+1, high);
  merge(arr, low, mid, high);
}

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

```
  vector<float> arr(n);

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

  mergeSort(arr, 0, n-1);

  cout<<"Sorted Array:"<<endl;

  for(int i=0; i<n; i++){
    cout<<fixed<<setprecision(2)<<arr[i]<<" ";
  }

  return 0;
}
```