

# Rajalakshmi Engineering College

Name: ROSSHNI PL  
Email: 240801279@rajalakshmi.edu.in  
Roll no: 240801279  
Phone: 9150237513  
Branch: REC  
Department: I ECE AF  
Batch: 2028  
Degree: B.E - ECE

Scan to verify results



## NeoColab\_REC\_CS23231\_DATA STRUCTURES

### REC\_DS using C\_Week 6\_PAH\_Updated

Attempt : 1  
Total Mark : 50  
Marks Obtained : 50

#### Section 1 : Coding

##### 1. Problem Statement

You are working as a programmer at a sports academy, and the academy holds various sports competitions regularly.

As part of the academy's system, you need to sort the scores of the participants in descending order using the Quick Sort algorithm.

Write a program that takes the scores of n participants as input and uses the Quick Sort algorithm to sort the scores in descending order. Your program should display the sorted scores after the sorting process.

##### ***Input Format***

The first line of input consists of an integer n, which represents the number of scores.

The second line of input consists of n integers, which represent scores separated by spaces.

### **Output Format**

Each line of output represents an iteration of the Quick Sort algorithm, displaying the elements of the array at that iteration.

After the iterations are complete, the last line of output prints the sorted scores in descending order separated by space.

Refer to the sample outputs for the formatting specifications.

### **Sample Test Case**

Input: 5

78 54 96 32 53

Output: Iteration 1: 78 54 96 53 32

Iteration 2: 96 54 78

Iteration 3: 78 54

Sorted Order: 96 78 54 53 32

### **Answer**

```
#include <stdio.h>
```

```
void swap(int *a, int *b) {  
    int temp = *a;  
    *a = *b;  
    *b = temp;  
}
```

```
int partition(int arr[], int low, int high, int iteration) {  
    int 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]);
```

```
printf("Iteration %d:", iteration);
for (int k = low; k <= high; k++) {
    printf(" %d", arr[k]);
}
printf("\n");
```

```
return i + 1;
}
```

```
void quickSort(int arr[], int low, int high, int *iteration) {
    if (low < high) {
        (*iteration)++;
        int pi = partition(arr, low, high, *iteration);

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

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

```
    scanf("%d", &n);
    int scores[n];
```

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

```
    int iteration = 0;
    quickSort(scores, 0, n - 1, &iteration);
```

```
printf("Sorted Order:");  
for (int i = 0; i < n; i++) {  
    printf(" %d", scores[i]);  
}  
printf("\n");  
  
return 0;  
}
```

**Status :** Correct

**Marks :** 10/10

## 2. Problem Statement

You are working on an optimization task for a sorting algorithm that uses insertion sort. Your goal is to determine the efficiency of the algorithm by counting the number of swaps needed to sort an array of integers.

Write a program that takes an array as input and calculates the number of swaps performed during the insertion sort process.

Example 1:

Input:

5

2 1 3 1 2

Output:

4

Explanation:

Step 1: [2, 1, 3, 1, 2] (No swaps)

Step 2: [1, 2, 3, 1, 2] (1 swap, element 1 shifts 1 place to the left)

Step 3: [1, 2, 3, 1, 2] (No swaps)

Step 4: [1, 1, 2, 3, 2] (2 swaps; element 1 shifts 2 places to the left)

Step 5: [1, 1, 2, 2, 3] (1 swap, element 2 shifts 1 place to the left)

Total number of swaps:  $1 + 2 + 1 = 4$

Example 2:

Input:

7

12 15 1 5 6 14 11

Output:

10

Explanation:

Step 1: [12, 15, 1, 5, 6, 14, 11] (No swaps)

Step 2: [12, 15, 1, 5, 6, 14, 11] (1 swap, element 15 shifts 1 place to the left)

Step 3: [12, 15, 1, 5, 6, 14, 11] (No swaps)

Step 4: [1, 12, 15, 5, 6, 14, 11] (2 swaps, element 1 shifts 2 places to the left)

Step 5: [1, 5, 12, 15, 6, 14, 11] (1 swap, element 5 shifts 1 place to the left)

Step 6: [1, 5, 6, 12, 15, 14, 11] (2 swaps, element 6 shifts 2 places to the left)

Step 7: [1, 5, 6, 12, 14, 15, 11] (1 swap, element 14 shifts 1 place to the left)

Step 8: [1, 5, 6, 11, 12, 14, 15] (3 swaps, element 11 shifts 3 places to the left)

Total number of swaps:  $1 + 2 + 1 + 2 + 1 + 3 = 10$

### ***Input Format***

The first line of input consists of an integer  $n$ , representing the number of elements in the array.

The second line of input consists of  $n$  space-separated integers, representing the

elements of the array.

### **Output Format**

The output prints the number of swaps performed during the insertion sort process.

Refer to the sample output for the formatting specifications.

### **Sample Test Case**

Input: 5

2 1 3 1 2

Output: 4

### **Answer**

```
// You are using GCC
```

```
#include <stdio.h>
```

```
int main() {  
    int n, i, j, key, swaps = 0;  
    scanf("%d", &n);  
    int arr[n];
```

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

```
    for (i = 1; i < n; i++) {  
        key = arr[i];  
        j = i - 1;
```

```
        while (j >= 0 && arr[j] > key) {  
            arr[j + 1] = arr[j];  
            j--;  
            swaps++;  
        }  
        arr[j + 1] = key;
```

```
}  
  
    printf("%d", swaps);  
    return 0;  
}
```

**Status :** Correct

**Marks :** 10/10

### 3. Problem Statement

You're a coach managing a list of finishing times for athletes in a race. The times are stored in an array, and you need to sort this array in ascending order to determine the rankings.

You'll use the insertion sort algorithm to accomplish this.

#### ***Input Format***

The first line of input contains an integer  $n$ , representing the number of athletes.

The second line contains  $n$  space-separated integers, each representing the finishing time of an athlete in seconds.

#### ***Output Format***

The output prints the sorted finishing times of the athletes in ascending order.

Refer to the sample output for formatting specifications.

#### ***Sample Test Case***

Input: 5

75 89 65 90 70

Output: 65 70 75 89 90

#### ***Answer***

```
// You are using GCC  
#include <stdio.h>
```

```

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

    int times[20];

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

    for (int i = 1; i < n; i++) {
        int key = times[i];
        int j = i - 1;

        while (j >= 0 && times[j] > key) {
            times[j + 1] = times[j];
            j = j - 1;
        }
        times[j + 1] = key;
    }

    for (int i = 0; i < n; i++) {
        printf("%d", times[i]);
        if (i != n - 1) {
            printf(" ");
        }
    }

    return 0;
}

```

**Status :** Correct

**Marks :** 10/10

#### 4. Problem Statement

Vishnu, a math enthusiast, is given a task to explore the magic of numbers. He has an array of positive integers, and his goal is to find the integer with



the highest digit sum in the sorted array using the merge sort algorithm.

You have to assist Vishnu in implementing the merge sort algorithm.

### ***Input Format***

The first line of input consists of an integer N, representing the number of elements in the array.

The second line consists of N space-separated integers, representing the array elements.

### ***Output Format***

The first line of output prints "The sorted array is: " followed by the sorted array, separated by a space.

The second line prints "The integer with the highest digit sum is: " followed by an integer representing the highest-digit sum.

Refer to the sample output for formatting specifications.

### ***Sample Test Case***

Input: 5

123 456 789 321 654

Output: The sorted array is: 123 321 456 654 789

The integer with the highest digit sum is: 789

### ***Answer***

```
#include <stdio.h>
```

```
int digitSum(int n) {  
    int sum = 0;  
    while (n > 0) {  
        sum += n % 10;  
        n /= 10;  
    }  
    return sum;  
}
```

```
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;  
    int j = 0;  
    int 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;

    scanf("%d", &N);

    int arr[N];

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

    mergeSort(arr, 0, N - 1);

    printf("The sorted array is:");
    for (int i = 0; i < N; i++) {
        printf(" %d", arr[i]);
    }
    printf("\n");
}

```

```

int maxDigitSum = -1;
int maxDigitSumNumber = -1;

```

```

for (int i = 0; i < N; i++) {
    int sum = digitSum(arr[i]);
    if (sum > maxDigitSum) {
        maxDigitSum = sum;
        maxDigitSumNumber = arr[i];
    } else if (sum == maxDigitSum && arr[i] > maxDigitSumNumber) {
        maxDigitSumNumber = arr[i];
    }
}

```

```

printf("The integer with the highest digit sum is: %d\n", maxDigitSumNumber);

```

```

return 0;
}

```

**Status :** Correct

**Marks :** 10/10

## 5. Problem Statement

Alex is working on a project that involves merging and sorting two arrays. He wants to write a program that merges two arrays, sorts the merged array in ascending order, removes duplicates, and prints the sorted array without duplicates.

Help Alex to implement the program using the merge sort algorithm.

### **Input Format**

The first line of input consists of an integer N, representing the number of elements in the first array.

The second line consists of N integers, separated by spaces, representing the elements of the first array.

The third line consists of an integer M, representing the number of elements in the second array.

The fourth line consists of M integers, separated by spaces, representing the

elements of the second array.

### **Output Format**

The output prints space-separated integers, representing the merged and sorted array in ascending order, with duplicate elements removed.

Refer to the sample output for the formatting specifications.

### **Sample Test Case**

Input: 4

1 2 3 4

3

3 4 5

Output: 1 2 3 4 5

### **Answer**

```
// You are using GCC
```

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

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

```
    int i, j, k;
```

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

```
    int n2 = right - mid;
```

```
    int L[n1], R[n2];
```

```
    for (i = 0; i < n1; i++)
```

```
        L[i] = arr[left + i];
```

```
    for (j = 0; j < n2; j++)
```

```
        R[j] = arr[mid + 1 + j];
```

```
    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 removeDuplicates(int arr[], int n) {
```

```
if (n == 0)
    return 0;
```

```
int j = 0;
for (int i = 1; i < n; i++) {
    if (arr[i] != arr[j]) {
        j++;
        arr[j] = arr[i];
    }
}
return j + 1;
}
```

```
int main() {
    int N, M;
```

```
    // Read the size of the first array
    scanf("%d", &N);
    int A[N];
    for (int i = 0; i < N; i++)
        scanf("%d", &A[i]);
```

```
    // Read the size of the second array
    scanf("%d", &M);
    int B[M];
    for (int i = 0; i < M; i++)
        scanf("%d", &B[i]);
```

```
    // Merge the two arrays into a third array
    int mergedSize = N + M;
    int merged[mergedSize];
    for (int i = 0; i < N; i++)
        merged[i] = A[i];
    for (int i = 0; i < M; i++)
        merged[N + i] = B[i];
```

```
    // Sort the merged array using merge sort
    mergeSort(merged, 0, mergedSize - 1);
```

```
    int uniqueSize = removeDuplicates(merged, mergedSize);
```

```
for (int i = 0; i < uniqueSize; i++) {  
    printf("%d", merged[i]),  
    if (i != uniqueSize - 1)  
        printf(" ");  
}  
  
return 0;  
}
```

**Status :** Correct

**Marks :** 10/10