Rajalakshmi Engineering College

Name: Libin Mathew
Email: 241801137@rajalakshmi.edu.in
Roll no: 241801137
Phone: 9947119753
Branch: REC
Department: I AI & DS AF
Batch: 2028
Degree: B.E - AI & DS

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 7_MCQ_Updated_1

Attempt : 1
Total Mark : 20
Marks Obtained : 18

Section 1 : MCQ

1. Merge sort is _____.

Answer

Comparison-based sorting algorithm

Status : Correct ***Marks : 1/1***

2. Which of the following strategies is used to improve the efficiency of Quicksort in practical implementations?

Answer

Choosing the pivot randomly or using the median-of-three method

Status : Correct ***Marks : 1/1***

3. The following code snippet is an example of a quick sort. What do the 'low' and 'high' parameters represent in this code?

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

Answer

The range of elements to sort within the array

Status : Correct ***Marks : 1/1***

4. Which of the following is true about Quicksort?

Answer

It is an in-place sorting algorithm

Status : Correct ***Marks : 1/1***

5. Why is Merge Sort preferred for sorting large datasets compared to Quick Sort?

Answer

Merge Sort has better worst-case time complexity

Status : Correct ***Marks : 1/1***

6. What happens during the merge step in Merge Sort?

Answer

Two sorted subarrays are combined into one sorted array

Status : Correct ***Marks : 1/1***

7. Let P be a quick sort program to sort numbers in ascending order using the first element as a pivot. Let t1 and t2 be the number of comparisons made by P for the inputs {1, 2, 3, 4, 5} and {4, 1, 5, 3, 2}, respectively. Which one of the following holds?

Answer

t1 > t2

Status : Correct ***Marks : 1/1***

8. Is Merge Sort a stable sorting algorithm?

Answer

Yes, always stable.

Status : Correct ***Marks : 1/1***

9. Which of the following modifications can help Quicksort perform better on small subarrays?

Answer

Switching to Insertion Sort for small subarrays

Status : Correct ***Marks : 1/1***

10. What is the best sorting algorithm to use for the elements in an array that are more than 1 million in general?

Answer

Quick sort.

Status : Correct ***Marks : 1/1***

11. Which of the following sorting algorithms is based on the divide and conquer method?

Answer

Merge Sort

Status : Correct ***Marks : 1/1***

12. In a quick sort algorithm, where are smaller elements placed to the pivot during the partition process, assuming we are sorting in increasing order?

Answer

To the left of the pivot

Status : Correct ***Marks : 1/1***

13. What happens when Merge Sort is applied to a single-element array?

Answer

The array remains unchanged and no merging is required

Status : Correct ***Marks : 1/1***

14. Which of the following scenarios is Merge Sort preferred over Quick Sort?

Answer

When sorting small datasets

Status : Wrong ***Marks : 0/1***

15. Which of the following is not true about QuickSort?

Answer

It as an adaptive sorting algorithm

Status : Wrong ***Marks : 0/1***

16. Consider the Quick Sort algorithm, which sorts elements in ascending order using the first element as a pivot. Then which of the following input

sequences will require the maximum number of comparisons when this algorithm is applied to it?

Answer

22 25 56 67 89

Status : Correct ***Marks : 1/1***

17. In a quick sort algorithm, what role does the pivot element play?

Answer

It is used to partition the array

Status : Correct ***Marks : 1/1***

18. What is the main advantage of Quicksort over Merge Sort?

Answer

Quicksort requires less auxiliary space

Status : Correct ***Marks : 1/1***

19. Which of the following statements is true about the merge sort algorithm?

Answer

It requires additional memory for merging

Status : Correct ***Marks : 1/1***

20. Which of the following methods is used for sorting in merge sort?

Answer

merging

Status : Correct ***Marks : 1/1***

Rajalakshmi Engineering College

Name: Libin Mathew
Email: 241801137@rajalakshmi.edu.in
Roll no: 241801137
Phone: 9947119753
Branch: REC
Department: I AI & DS AF
Batch: 2028
Degree: B.E - AI & DS

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 7_COD_Question 5

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

Jose has an array of N fractional values, represented as double-point numbers. He needs to sort these fractions in increasing order and seeks your help.

Write a program to help Jose sort the array using the merge sort algorithm.

Input Format

The first line of input consists of an integer N, representing the number of fractions to be sorted.

The second line consists of N double-point numbers, separated by spaces, representing the fractions array.

Output Format

The output prints N double-point numbers, sorted in increasing order, and rounded to three decimal places.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 4
0.123 0.543 0.321 0.789

Output: 0.123 0.321 0.543 0.789

Answer

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


int compare(double a, double b) {
   if (a < b) return -1;
   else if (a > b) return 1;
   else return 0;
}

void merge(double arr[], int l, int m, int r) {
   int i, j, k;
   int n1 = m - l + 1;
   int n2 = r - m;

   double L[n1], R[n2];

   for (i = 0; i < n1; i++)
      L[i] = arr[l + i];
   for (j = 0; j < n2; j++)
      R[j] = arr[m + 1 + j];

   i = 0;
   j = 0;
   k = l;

   while (i < n1 && j < n2) {
```

```
        if (compare(L[i], R[j]) <= 0) {
            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(double arr[], int l, int r) {
    if (l < r) {
        int m = l + (r - l) / 2;
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}

int main() {
    int n;
    scanf("%d", &n);
    double fractions[n];
    for (int i = 0; i < n; i++) {
        scanf("%lf", &fractions[i]);
    }
    mergeSort(fractions, 0, n - 1);
    for (int i = 0; i < n; i++) {
        printf("%.3f ", fractions[i]);
    }
```

```
    return 0;
}
```

Status : Correct ***Marks : 10/10***

Rajalakshmi Engineering College

Name: Libin Mathew
Email: 241801137@rajalakshmi.edu.in
Roll no: 241801137
Phone: 9947119753
Branch: REC
Department: I AI & DS AF
Batch: 2028
Degree: B.E - AI & DS

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 7_PAH_Updated

Attempt : 1
Total Mark : 50
Marks Obtained : 47.5

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

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

int iteration = 1;

// Function to swap two integers
void swap(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

// Partition function for descending order
int partition(int arr[], int low, int high) {
    int pivot = arr[high];  // pivot
    int i = low - 1;
```

```
    for (int j = low; j < high; j++) {
        if (arr[j] > pivot) { // For descending order
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);

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

    return i + 1;
}

// QuickSort function
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high); // partition index

        quickSort(arr, low, pi - 1);  // Left side
        quickSort(arr, pi + 1, high); // Right side
    }
}

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

    int arr[10]; // max size is 10 as per constraints

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

    quickSort(arr, 0, n - 1);

    // Print final sorted order
    printf("Sorted Order:");
```

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

    return 0;
}
```

Status : Partially correct **Marks : 7.5/10**

2. 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, i, j, key;
    int times[20];  // max n = 20

    // Read number of athletes
    scanf("%d", &n);

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

    // Insertion sort algorithm
    for (i = 1; i < n; i++) {
        key = times[i];
        j = i - 1;

        // Move elements greater than key to one position ahead
        while (j >= 0 && times[j] > key) {
            times[j + 1] = times[j];
            j = j - 1;
        }
        times[j + 1] = key;
    }

    // Print sorted finishing times
    for (i = 0; i < n; i++) {
        printf("%d ", times[i]);
    }
    printf("\n");

    return 0;
}
```

Status : Correct **Marks : 10/10**

3. 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>

// Function to merge two halves
void merge(int arr[], int left, int mid, int right) {
    int i = left, j = mid + 1, k = 0;
```

```
    int temp[20];  // Max size is N + M = 20

    while (i <= mid && j <= right) {
        if (arr[i] < arr[j])
            temp[k++] = arr[i++];
        else
            temp[k++] = arr[j++];
    }

    while (i <= mid)
        temp[k++] = arr[i++];
    while (j <= right)
        temp[k++] = arr[j++];

    for (i = left, k = 0; i <= right; i++, k++)
        arr[i] = temp[k];
}

// Merge sort function
void mergeSort(int arr[], int left, int right) {
    if (left < right) {
        int mid = (left + right) / 2;
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}

// Function to remove duplicates from sorted array
int removeDuplicates(int arr[], int n, int result[]) {
    int j = 0;
    for (int i = 0; i < n; i++) {
        if (i == 0 || arr[i] != arr[i - 1]) {
            result[j++] = arr[i];
        }
    }
    return j; // New size after removing duplicates
}

int main() {
    int N, M;
    int arr1[10], arr2[10], merged[20], result[20];
```

```
    // Input first array
    scanf("%d", &N);
    for (int i = 0; i < N; i++) {
        scanf("%d", &arr1[i]);
    }

    // Input second array
    scanf("%d", &M);
    for (int i = 0; i < M; i++) {
        scanf("%d", &arr2[i]);
    }

    // Merge both arrays
    for (int i = 0; i < N; i++) {
        merged[i] = arr1[i];
    }
    for (int i = 0; i < M; i++) {
        merged[N + i] = arr2[i];
    }

    int totalSize = N + M;

    // Sort the merged array
    mergeSort(merged, 0, totalSize - 1);

    // Remove duplicates
    int newSize = removeDuplicates(merged, totalSize, result);

    // Print result
    for (int i = 0; i < newSize; i++) {
        printf("%d ", result[i]);
    }
    printf("\n");

    return 0;
}
```

Status : Correct ***Marks : 10/10***

4. 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
```

```c
#include <stdio.h>

int main() {
    int n, i, j, key, swaps = 0;
    int arr[10];  // As n ≤ 10

    // Read input size
    scanf("%d", &n);

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

    // Insertion sort with swap count
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;

        // Count how many elements need to be shifted
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            swaps++; // Each shift is a "swap" in this context
            j--;
        }
        arr[j + 1] = key;
    }

    // Print total number of swaps
    printf("%d\n", swaps);

    return 0;
}
```

Status : Correct **Marks : 10/10**

5. 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

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

// Function to calculate digit sum
int digitSum(int num) {
    int sum = 0;
    while (num != 0) {
        sum += num % 10;
        num /= 10;
    }
    return sum;
}
```

```
// Merge function for Merge Sort
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[10], R[10]; // Max size as per constraint (N <= 10)

    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++];
        else
            arr[k++] = R[j++];
    }

    while (i < n1)
        arr[k++] = L[i++];

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

// Merge Sort function
void mergeSort(int arr[], int left, int right) {
    if (left < right) {
        int mid = (left + right)/2;
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, mid, right);
    }
}

int main() {
    int N, arr[10], i;
```

```
    // Input
    scanf("%d", &N);
    for (i = 0; i < N; i++)
        scanf("%d", &arr[i]);

    // Sort the array
    mergeSort(arr, 0, N - 1);

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

    // Find integer with the highest digit sum
    int maxDigitSum = -1, numberWithMaxDigitSum;
    for (i = 0; i < N; i++) {
        int dSum = digitSum(arr[i]);
        if (dSum > maxDigitSum) {
            maxDigitSum = dSum;
            numberWithMaxDigitSum = arr[i];
        }
    }

    // Output result
    printf("The integer with the highest digit sum is: %d\n",
numberWithMaxDigitSum);

    return 0;
}
```

Status : Correct **Marks : 10/10**

Rajalakshmi Engineering College

Name: Libin Mathew
Email: 241801137@rajalakshmi.edu.in
Roll no: 241801137
Phone: 9947119753
Branch: REC
Department: I AI & DS AF
Batch: 2028
Degree: B.E - AI & DS

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 7_CY_Updated

Attempt : 1
Total Mark : 30
Marks Obtained : 21

Section 1 : Coding

1. Problem Statement

Priya, a data analyst, is working on a dataset of integers. She needs to find the maximum difference between two successive elements in the sorted version of the dataset. The dataset may contain a large number of integers, so Priya decides to use QuickSort to sort the array before finding the difference. Can you help Priya solve this efficiently?

Input Format

The first line of input consists of an integer n, representing the size of the array.

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

Output Format

The output prints a single integer, representing the maximum difference between

two successive elements in the sorted form of the array.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 1
10

Output: Maximum gap: 0

Answer

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

// Function to swap two integers
void swap(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

// Partition function for QuickSort
int partition(int arr[], int low, int high) {
    int pivot = arr[high]; // last element as pivot
    int i = low - 1;

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

// QuickSort function
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int p = partition(arr, low, high);
```

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

int main() {
    int n;
    int arr[10];

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

    // Edge case: only 1 element
    if (n < 2) {
        printf("Maximum gap: 0\n");
        return 0;
    }

    // Sort the array
    quickSort(arr, 0, n - 1);

    // Find maximum gap between successive elements
    int maxGap = 0;
    for (int i = 1; i < n; i++) {
        int gap = arr[i] - arr[i - 1];
        if (gap > maxGap)
            maxGap = gap;
    }

    printf("Maximum gap: %d\n", maxGap);
    return 0;
}
```

Status : Correct **Marks : 10/10**

2. Problem Statement

Ravi is given an array of integers and is tasked with sorting it in a unique way. He needs to sort the elements in such a way that the elements at odd

positions are in descending order, and the elements at even positions are in ascending order. Ravi decided to use the Insertion Sort algorithm for this task.

Your task is to help ravi, to create even_odd_insertion_sort function to sort the array as per the specified conditions and then print the sorted array.

Example

Input:

10

25 36 96 58 74 14 35 15 75 95

Output:

96 14 75 15 74 36 35 58 25 95

Input Format

The first line of input consists of a single integer, N, which represents the size of the array.

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

Output Format

The output displays the sorted array using the even-odd insertion sort algorithm and prints the sorted array.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 4
3 1 4 2
Output: 4 1 3 2

Answer

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

// Insertion sort for descending order
void insertionSortDescending(int arr[], int n) {
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;

        // Move elements smaller than key to the right
        while (j >= 0 && arr[j] < key) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}

// Insertion sort for ascending order
void insertionSortAscending(int arr[], int n) {
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;

        // Move elements greater than key to the right
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}

void even_odd_insertion_sort(int arr[], int n) {
    int odd[10], even[10]; // For index positions
    int oddCount = 0, evenCount = 0;

    // Split into odd (0,2,4..) and even (1,3,5..) index elements
    for (int i = 0; i < n; i++) {
        if (i % 2 == 0)
            odd[oddCount++] = arr[i];
        else
```

```
            even[evenCount++] = arr[i];
        }

        // Sort odd-position elements in descending order
        insertionSortDescending(odd, oddCount);

        // Sort even-position elements in ascending order
        insertionSortAscending(even, evenCount);

        // Merge sorted values back into original array
        int o = 0, e = 0;
        for (int i = 0; i < n; i++) {
            if (i % 2 == 0)
                arr[i] = odd[o++];
            else
                arr[i] = even[e++];
        }

        // Print the sorted array
        for (int i = 0; i < n; i++)
            printf("%d ", arr[i]);
        printf("\n");
    }

    int main() {
        int n, arr[10];

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

        even_odd_insertion_sort(arr, n);

        return 0;
    }
```

Status : Correct **Marks : 10/10**

3. Problem Statement

Sheela wants to distribute cookies to her children, but each child will only be happy if the cookie size meets or exceeds their individual greed factor. She has a limited number of cookies and wants to make as many children happy as possible. Priya decides to sort both the greed factors and cookie sizes using QuickSort to efficiently match cookies with children. Your task is to help Sheela determine the maximum number of children that can be made happy.

Input Format

The first line of input consists of an integer n, representing the number of children.

The second line contains n space-separated integers, where each integer represents the greed factor of a child.

The third line contains an integer m, representing the number of cookies.

The fourth line contains m space-separated integers, where each integer represents the size of a cookie.

Output Format

The output prints a single integer, representing the maximum number of children that can be made happy.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 3
1 2 3
2
1 1
Output: The child with greed factor: 1

Answer

```
#include <stdio.h>

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

```
    if (low < high) {
        int i = low, j = high;
        int pivot = arr[low];
        while (i < j) {
            while (i <= high && arr[i] <= pivot) i++;
            while (arr[j] > pivot) j--;
            if (i < j) {
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
        int temp = arr[low];
        arr[low] = arr[j];
        arr[j] = temp;

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

void matchChildren(int greed[], int n, int cookies[], int m) {
    quickSort(greed, 0, n - 1);
    quickSort(cookies, 0, m - 1);

    int i = 0, j = 0;

    while (i < n && j < m) {
        if (cookies[j] >= greed[i]) {
            printf("The child with greed factor: %d\n", greed[i]);
            i++;
            j++;
        } else {
            j++;
        }
    }
}

int main() {
    int n, m;
    int greed[100], cookies[100];
```

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

    scanf("%d", &m);
    for (int i = 0; i < m; i++)
        scanf("%d", &cookies[i]);

    matchChildren(greed, n, cookies, m);

    return 0;
}
```

Status : Partially correct **Marks : 1/10**