Rajalakshmi Engineering College

Name: MITHUN CHAKRAVARTHY
Email: 241801158@rajalakshmi.edu.in
Roll no: 241801158
Phone: 9360341516
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 6_MCQ_Updated_1

Attempt : 1
Total Mark : 20
Marks Obtained : 20

Section 1 : MCQ

1. 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***

2. Is Merge Sort a stable sorting algorithm?

Answer

Yes, always stable.

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

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

Answer

Two sorted subarrays are combined into one sorted array

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

4. 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***

5. 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***

6. 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***

7. 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*

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

Answer

merging

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. Which of the following is not true about QuickSort?

Answer

It can be implemented as a stable 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. What is the main advantage of Quicksort over Merge Sort?

Answer

Quicksort requires less auxiliary space

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

13. 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***

14. 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***

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

Answer

It requires additional memory for merging

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

16. 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*

17. Which of the following is true about Quicksort?

Answer

It is an in-place sorting algorithm

Status : Correct *Marks : 1/1*

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

Answer

When sorting linked lists

Status : Correct *Marks : 1/1*

19. Merge sort is ________.

Answer

Comparison-based sorting algorithm

Status : Correct *Marks : 1/1*

20. 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 &gt; t2

Status : Correct *Marks : 1/1*

Rajalakshmi Engineering College

Name: MITHUN CHAKRAVARTHY
Email: 241801158@rajalakshmi.edu.in
Roll no: 241801158
Phone: 9360341516
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 6_COD_Question 1

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

John and Mary are collaborating on a project that involves data analysis. They each have a set of age data, one sorted in ascending order and the other in descending order. However, their analysis requires the data to be in ascending order.

Write a program to help them merge the two sets of age data into a single sorted array in ascending order using merge sort.

Input Format

The first line of input consists of an integer N, representing the number of age values in each dataset.

The second line consists of N space-separated integers, representing the ages of participants in John's dataset (in ascending order).

The third line consists of N space-separated integers, representing the ages of participants in Mary's dataset (in descending order).

Output Format

The output prints a single line containing space-separated integers, which represents the merged dataset of ages sorted in ascending order.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
1 3 5 7 9
10 8 6 4 2

Output: 1 2 3 4 5 6 7 8 9 10

Answer

```
#include <stdio.h>

#include <stdlib.h>

void mergeSort(int arr[], int n) {
    if (n < 2) return;

    int* temp = (int*) malloc(n * sizeof(int));
    if (!temp) return;

    for (int size = 1; size < n; size *= 2) {
        for (int left_start = 0; left_start < n; left_start += 2*size) {
            int mid = left_start + size - 1;
            int right_end = left_start + 2*size - 1;
            if (mid >= n) mid = n - 1;
            if (right_end >= n) right_end = n - 1;

            // Merge subarrays arr[left_start..mid] and arr[mid+1..right_end] into temp
            int i = left_start, j = mid + 1, k = left_start;
            while (i <= mid && j <= right_end) {
                if (arr[i] <= arr[j]) {
                    temp[k++] = arr[i++];
                } else {
                    temp[k++] = arr[j++];
                }
            }
            while (i <= mid) temp[k++] = arr[i++];
            while (j <= right_end) temp[k++] = arr[j++];

            // Copy sorted temp back to arr
            for (int idx = left_start; idx <= right_end; idx++) {
                arr[idx] = temp[idx];
            }
        }
    }
    free(temp);
}

void merge(int merged[], int arr1[], int arr2[], int n1, int n2) {
    int i = 0, j = 0, k = 0;
    while (i < n1 && j < n2) {
        if (arr1[i] <= arr2[j]) {
            merged[k++] = arr1[i++];
        } else {
            merged[k++] = arr2[j++];
        }
    }
    while (i < n1) {
        merged[k++] = arr1[i++];
    }
    while (j < n2) {
        merged[k++] = arr2[j++];
    }
}


int main() {
    int n, m;
    scanf("%d", &n);
    int arr1[n], arr2[n];
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr1[i]);
    }
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr2[i]);
    }
    int merged[n + n];
    mergeSort(arr1, n);
    mergeSort(arr2, n);
    merge(merged, arr1, arr2, n, n);
    for (int i = 0; i < n + n; i++) {
        printf("%d ", merged[i]);
    }
    return 0;
}
```

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

Rajalakshmi Engineering College

Name: MITHUN CHAKRAVARTHY
Email: 241801158@rajalakshmi.edu.in
Roll no: 241801158
Phone: 9360341516
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 6_PAH_Updated

Attempt : 1
Total Mark : 50
Marks Obtained : 40

Section 1 : Coding

1. 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\n", swaps);

    return 0;
}
```

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

2. 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

```
#include <stdio.h>

void merge(int arr[], int l, int m, int r) {
   int i, j, k;
   int n1 = m - l + 1;
   int n2 = r - m;
   int 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 (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 l, int r) {
   if (l < r) {
      int m = (l + r) / 2;
```

```
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}

void printUniqueSorted(int arr[], int size) {
    printf("%d ", arr[0]);
    for (int i = 1; i < size; i++) {
        if (arr[i] != arr[i - 1])
            printf("%d ", arr[i]);
    }
}

int main() {
    int N, M;
    scanf("%d", &N);
    int A[N];
    for (int i = 0; i < N; i++)
        scanf("%d", &A[i]);
    scanf("%d", &M);
    int B[M];
    for (int i = 0; i < M; i++)
        scanf("%d", &B[i]);
    int merged[N + M];
    for (int i = 0; i < N; i++)
        merged[i] = A[i];
    for (int i = 0; i < M; i++)
        merged[N + i] = B[i];
    mergeSort(merged, 0, N + M - 1);
    printUniqueSorted(merged, N + M);
    return 0;
}
```

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

3. 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>

void merge(int arr[], int l, int m, int r) {
   int i, j, k;
   int n1 = m - l + 1;
   int n2 = r - m;
   int 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 (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 l, int r) {
    if (l < r) {
        int m = (l + r) / 2;
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}

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

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]);
    int maxSum = 0, maxNum = 0;
    for (int i = 0; i < N; i++) {
        int sum = digitSum(arr[i]);
        if (sum > maxSum) {
            maxSum = sum;
            maxNum = arr[i];
        }
    }
    printf("\nThe integer with the highest digit sum is: %d\n", maxNum);
    return 0;
}
```

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

4. 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[n];
   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--;
      }
      times[j + 1] = key;
   }

   for (int i = 0; i < n; i++)
      printf("%d ", times[i]);

   return 0;
}
```

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

5. 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>

void quick_sort(int arr[], int low, int high, int iterations[][10], int *iter_count);
int partition(int arr[], int low, int high);
int main() {
    int n;
    scanf("%d", &n);

    int scores[10];
    for (int i = 0; i < n; i++) {
        scanf("%d", &scores[i]);
    }

    int iterations[10][10]; // To store the state of the array at each iteration
    int iter_count = 0;

    quick_sort(scores, 0, n - 1, iterations, &iter_count);

    // Print the iterations
    for (int i = 0; i < iter_count; i++) {
        printf("Iteration %d: ", i + 1);
        for (int j = 0; j < n; j++) {
            printf("%d ", iterations[i][j]);
        }
        printf("\n");
    }

    // Print the sorted order
    printf("Sorted Order: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", scores[i]);
    }
    printf("\n");

    return 0;
}

void quick_sort(int arr[], int low, int high, int iterations[][10], int *iter_count) {
    if (low < high) {
        // Partition the array
        int pi = partition(arr, low, high);

        // Store the current state of the array for output
        for (int i = 0; i < 10; i++) {
            iterations[*iter_count][i] = arr[i];
        }
        (*iter_count)++;

        // Recursively sort the elements before and after partition
        quick_sort(arr, low, pi - 1, iterations, iter_count);
        quick_sort(arr, pi + 1, high, iterations, iter_count);
    }
}

int partition(int arr[], int low, int high) {
    int pivot = arr[high]; // Choosing the last element as pivot
    int i = low - 1;       // Pointer for the smaller element
    for (int j = low; j < high; j++) {
        // If current element is greater than or equal to pivot
        if (arr[j] >= pivot) {
            i++;
            // Swap arr[i] and arr[j]
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
    // Swap pivot
    int temp = arr[i + 1];
    arr[i + 1] = arr[high];
    arr[high] = temp;
    return i + 1;
}
```

Status : Wrong ***Marks : 0/10***