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 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

```
#include <stdio.h>

int insertionSortCountSwaps(int arr[], int n) {
    int swapCount = 0;
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;
        while(j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            swapCount++;
            j--;
        }
        arr[j + 1] = key;
    }
    return swapCount;
}

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

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

    int swaps = insertionSortCountSwaps(arr, n);
    printf("%d", swaps);

    return 0;
}
```

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

2. 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>

int iteration = 1;

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

void printArraySection(int arr[], int low, int high) {
    for (int k = low; k <= high; k++) {
        printf("%d ", arr[k]);
    }
    printf("\n");
}

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

        printf("Iteration %d: ", iteration++);
        printArraySection(arr, low, high);

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

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

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

    quickSort(arr, 0, n - 1);

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

    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

```
#include <stdio.h>

// Function to perform insertion sort on the array in ascending order.
void insertionSort(int arr[], int n) {
   for (int i = 1; i < n; i++) {
      int key = arr[i];
      int j = i - 1;
      // Shift elements of arr[0..i-1] that are greater than key one position to the
right.
      while (j >= 0 && arr[j] > key) {
         arr[j + 1] = arr[j];
         j--;
      }
      arr[j + 1] = key;
   }
}
```

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

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

    // Sort the finishing times in ascending order.
    insertionSort(times, n);

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

    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>

// Merges two sorted subarrays arr[left..mid] and arr[mid+1..right]
void merge(int arr[], int left, int mid, int right) {
   int n1 = mid - left + 1;
   int n2 = right - mid;
   int leftArr[n1], rightArr[n2];

   // Copy data into temporary arrays
   for (int i = 0; i < n1; i++) {
      leftArr[i] = arr[left + i];
   }
   for (int j = 0; j < n2; j++) {
      rightArr[j] = arr[mid + 1 + j];
   }

   // Merge the temporary arrays back into arr[left..right]
   int i = 0, j = 0, k = left;
   while (i < n1 && j < n2) {
      if (leftArr[i] <= rightArr[j]) {
         arr[k++] = leftArr[i++];
      } else {
         arr[k++] = rightArr[j++];
      }
   }
```

```
    // Copy remaining elements of leftArr[], if any
    while (i < n1) {
        arr[k++] = leftArr[i++];
    }
    // Copy remaining elements of rightArr[], if any
    while (j < n2) {
        arr[k++] = rightArr[j++];
    }
}

// Merge sort algorithm to sort the array in ascending order
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);
    }
}

// Returns the sum of digits of a positive integer
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]);
    }

    // Sort the array using merge sort
    mergeSort(arr, 0, n - 1);
    // Print the sorted array
    printf("The sorted array is: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    // Find the integer with the highest digit sum
    int maxSum = -1;
    int maxNum = 0;
    for (int i = 0; i < n; i++) {
        int currSum = digitSum(arr[i]);
        if (currSum > maxSum) {
            maxSum = currSum;
            maxNum = arr[i];
        }
    }

    // Print the integer with the highest digit sum
    printf("The integer with the highest digit sum is: %d", maxNum);
    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

```
#include <stdio.h>

// Merge function to combine two sorted subarrays:
// arr[left..mid] and arr[mid+1..right]
void merge(int arr[], int left, int mid, int right) {
   int n1 = mid - left + 1; // Size of left subarray
   int n2 = right - mid;    // Size of right subarray

   int leftArr[n1], rightArr[n2];

   // Copy data into temporary subarrays.
   for (int i = 0; i < n1; i++) {
      leftArr[i] = arr[left + i];
   }
   for (int j = 0; j < n2; j++) {
      rightArr[j] = arr[mid + 1 + j];
   }
```

```
    // Merge the temporary arrays back into arr[left..right]
    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        if (leftArr[i] <= rightArr[j]) {
            arr[k++] = leftArr[i++];
        } else {
            arr[k++] = rightArr[j++];
        }
    }

    // Copy any remaining elements of leftArr[], if any.
    while (i < n1) {
        arr[k++] = leftArr[i++];
    }
    // Copy any remaining elements of rightArr[], if any.
    while (j < n2) {
        arr[k++] = rightArr[j++];
    }
}

// Merge sort algorithm to sort the array in ascending order.
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, m;

    // Read the first array.
    scanf("%d", &n);
    int arr1[11];  // n is at most 10.
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr1[i]);
    }

    // Read the second array.
    scanf("%d", &m);
    int arr2[11];  // m is at most 10.
    for (int i = 0; i < m; i++) {
        scanf("%d", &arr2[i]);
    }

    // Merge both arrays into a single array.
    int total = n + m;
    int merged[21]; // Size is n+m, at most 20.

    for (int i = 0; i < n; i++) {
        merged[i] = arr1[i];
    }
    for (int i = 0; i < m; i++) {
        merged[n + i] = arr2[i];
    }

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

    // Print the merged, sorted array without duplicates.
    // As the array is sorted, duplicate elements will be adjacent.
    for (int i = 0; i < total; i++) {
        if (i == 0 || merged[i] != merged[i - 1]) {
            printf("%d ", merged[i]);
        }
    }

    return 0;
}
```

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