NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 6_CY_Updated

Attempt : 1
Total Mark : 30
Marks Obtained : 30

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

```
#include <stdio.h>

// Partition function using the Lomuto partition scheme
int partition(int arr[], int low, int high) {
   int pivot = arr[high];
   int i = low - 1; // place for swapping
   for (int j = low; j < high; j++) {
      if (arr[j] <= pivot) {
         i++;
         // Swap arr[i] and arr[j]
         int temp = arr[i];
         arr[i] = arr[j];
         arr[j] = temp;
      }
   }
   // Place pivot in the correct position
   int temp = arr[i + 1];
   arr[i + 1] = arr[high];
   arr[high] = temp;
   return i + 1;
}

// QuickSort algorithm to sort the array in ascending order.
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;
   scanf("%d", &n);

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

   // If there is only one element, the maximum gap is 0.
   if (n < 2) {
      printf("Maximum gap: 0");
      return 0;
   }

   // Sort the array in ascending order using QuickSort.
   quickSort(arr, 0, n - 1);

   int maxGap = 0;
   // Calculate the maximum difference between successive elements.
   for (int i = 0; i < n - 1; i++) {
      int diff = arr[i + 1] - arr[i];
      if (diff > maxGap) {
         maxGap = diff;
      }
   }

   // Print the result in the required format.
   printf("Maximum gap: %d", maxGap);
   return 0;
}
```

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

2. Problem Statement

Meera is organizing her art supplies, which are represented as a list of integers: red (0), white (1), and blue (2). She needs to sort these supplies so that all items of the same color are adjacent, in the order red, white, and blue. To achieve this efficiently, Meera decides to use QuickSort to sort the items. Can you help Meera arrange her supplies in the desired order?

Input Format

The first line of input consists of an integer n, representing the number of items in the list.

The second line consists of n space-separated integers, where each integer is either 0 (red), 1 (white), or 2 (blue).

Output Format

The output prints the sorted list of integers in a single line, where integers are arranged in the order red (0), white (1), and blue (2).

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 6
2 0 2 1 1 0

Output: Sorted colors:
0 0 1 1 2 2

Answer

```
#include <stdio.h>

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

// Partition function using the Lomuto partition scheme
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++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return i + 1;
}

// QuickSort routine to sort the array in ascending order
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(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]);
    }

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

    // Print the sorted colors in the desired format
    printf("Sorted colors:\n");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }

    return 0;
}
```

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

3. Problem Statement

Reshma is passionate about sorting algorithms and has recently learned about the merge sort algorithm. She wants to implement a program that utilizes the merge sort algorithm to sort an array of integers, both positive and negative, in ascending order.

Help her in implementing the program.

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 N space-separated integers, representing the array elements sorted in ascending order.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 9
5 -3 0 12 7 -8 2 1 6
Output: -8 -3 0 1 2 5 6 7 12

Answer

```
#include <stdio.h>

void merge(int arr[], int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;

    int leftArr[n1], rightArr[n2];

    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];
    }

    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++];
        }
    }

    while (i < n1) {
        arr[k++] = leftArr[i++];
    }

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

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);

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

    return 0;
}
```

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