Rajalakshmi Engineering College

Name: Vikashini M
Email: 241801314@rajalakshmi.edu.in
Roll no: 241801314
Phone: 9345747519
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_CY_Updated

Attempt : 1
Total Mark : 30
Marks Obtained : 30

Section 1 : Coding

1. 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>
#include <stdlib.h>
struct Node {
   int data;
   struct Node* next;
};

struct Node* newNode(int data) {
   struct Node* temp = (struct Node*) malloc(sizeof(struct Node));
   temp->data = data;
   temp->next = NULL;
   return temp;
}

void append(struct Node** headRef, int data) {
   struct Node* temp = newNode(data);
   if (*headRef == NULL) {
      *headRef = temp;
   } else {
      struct Node* current = *headRef;
      while (current->next != NULL)
         current = current->next;
      current->next = temp;
   }
}
```

```
struct Node* sortedMerge(struct Node* a, struct Node* b) {
   if (a == NULL) return b;
   if (b == NULL) return a;

   struct Node* result = NULL;
   if (a->data <= b->data) {
      result = a;
      result->next = sortedMerge(a->next, b);
   } else {
      result = b;
      result->next = sortedMerge(a, b->next);
   }
   return result;
}
void frontBackSplit(struct Node* source, struct Node** frontRef, struct Node**
backRef) {
   struct Node* slow = source;
   struct Node* fast = source->next;

   while (fast != NULL) {
      fast = fast->next;
      if (fast != NULL) {
         slow = slow->next;
         fast = fast->next;
      }
   }

   *frontRef = source;
   *backRef = slow->next;
   slow->next = NULL;
}
void mergeSort(struct Node** headRef) {
   struct Node* head = *headRef;
   if (head == NULL || head->next == NULL)
      return;

   struct Node* a;
   struct Node* b;

   frontBackSplit(head, &a, &b);

   mergeSort(&a);
```

```
    mergeSort(&b);

    *headRef = sortedMerge(a, b);
}
void printList(struct Node* head) {
    while (head != NULL) {
        printf("%d ", head->data);
        head = head->next;
    }
    printf("\n");
}

int main() {
    int n, val;
    struct Node* head = NULL;

    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &val);
        append(&head, val);
    }
    mergeSort(&head);
    printList(head);

    return 0;
}
```

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

2. Problem Statement

Marie, the teacher, wants her students to implement the ascending order of numbers while also exploring the concept of prime numbers.

Students need to write a program that sorts an array of integers using the merge sort algorithm while counting and returning the number of prime integers in the array. Help them to complete the program.

Input Format

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

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

Output Format

The first line of output prints the sorted array of integers in ascending order.

The second line prints the number of prime integers in the array.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 7
5 3 6 8 9 7 4

Output: Sorted array: 3 4 5 6 7 8 9
Number of prime integers: 3

Answer

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


struct Node {
   int data;
   struct Node* next;
};
struct Node* newNode(int data) {
   struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
   temp->data = data;
   temp->next = NULL;
   return temp;
}
void append(struct Node** headRef, int data) {
   struct Node* temp = newNode(data);
   if (*headRef == NULL) {
      *headRef = temp;
   } else {
        struct Node* current = *headRef;
        while (current->next != NULL)
            current = current->next;
        current->next = temp;
    }
}
int isPrime(int num) {
    if (num <= 1) return 0;
    if (num == 2) return 1;
    if (num % 2 == 0) return 0;
    for (int i = 3; i <= sqrt(num); i += 2) {
        if (num % i == 0)
            return 0;
    }
    return 1;
}
struct Node* sortedMerge(struct Node* a, struct Node* b) {
    if (a == NULL) return b;
    if (b == NULL) return a;

    struct Node* result = NULL;
    if (a->data <= b->data) {
        result = a;
        result->next = sortedMerge(a->next, b);
    } else {
        result = b;
        result->next = sortedMerge(a, b->next);
    }
    return result;
}
void frontBackSplit(struct Node* source, struct Node** frontRef, struct Node**
backRef) {
    struct Node* slow;
    struct Node* fast;
    slow = source;
    fast = source->next;

    while (fast != NULL) {
        fast = fast->next;
        if (fast != NULL) {
            slow = slow->next;
            fast = fast->next;
        }
    }

    *frontRef = source;
    *backRef = slow->next;
    slow->next = NULL;
}


void mergeSort(struct Node** headRef) {
    struct Node* head = *headRef;
    if (head == NULL || head->next == NULL) {
        return;
    }

    struct Node* a;
    struct Node* b;

    frontBackSplit(head, &a, &b);

    mergeSort(&a);
    mergeSort(&b);

    *headRef = sortedMerge(a, b);
}


void printList(struct Node* head) {
    while (head != NULL) {
        printf("%d ", head->data);
        head = head->next;
    }
}


int countPrimes(struct Node* head) {
    int count = 0;
    while (head != NULL) {
        if (isPrime(head->data)) count++;
        head = head->next;
    }
    return count;
```

```
}

int main() {
    int n, val;
    struct Node* head = NULL;

    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &val);
        append(&head, val);
    }

    mergeSort(&head);

    printf("Sorted array: ");
    printList(head);
    printf(" Number of prime integers: %d\n", countPrimes(head));

    return 0;
}
```

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

3. 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

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
   int data;
   struct Node* next;
};
struct Node* newNode(int data) {
   struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
   temp->data = data;
   temp->next = NULL;
   return temp;
}
void append(struct Node** headRef, int data) {
   struct Node* temp = newNode(data);
```

```
    if (*headRef == NULL) {
        *headRef = temp;
    } else {
        struct Node* current = *headRef;
        while (current->next != NULL)
            current = current->next;
        current->next = temp;
    }
}
struct Node* insertionSortAsc(struct Node* head) {
    struct Node* sorted = NULL;
    while (head != NULL) {
        struct Node* curr = head;
        head = head->next;

        if (sorted == NULL || curr->data < sorted->data) {
            curr->next = sorted;
            sorted = curr;
        } else {
            struct Node* temp = sorted;
            while (temp->next != NULL && temp->next->data < curr->data) {
                temp = temp->next;
            }
            curr->next = temp->next;
            temp->next = curr;
        }
    }
    return sorted;
}
struct Node* insertionSortDesc(struct Node* head) {
    struct Node* sorted = NULL;
    while (head != NULL) {
        struct Node* curr = head;
        head = head->next;

        if (sorted == NULL || curr->data > sorted->data) {
            curr->next = sorted;
            sorted = curr;
        } else {
            struct Node* temp = sorted;
            while (temp->next != NULL && temp->next->data > curr->data) {
                temp = temp->next;
            }
            curr->next = temp->next;
            temp->next = curr;
        }
    }
    return sorted;
}
void printMerged(struct Node* odd, struct Node* even) {
    while (odd != NULL || even != NULL) {
        if (odd != NULL) {
            printf("%d ", odd->data);
            odd = odd->next;
        }
        if (even != NULL) {
            printf("%d ", even->data);
            even = even->next;
        }
    }
    printf("\n");
}

int main() {
    int n, val;
    struct Node *odd = NULL, *even = NULL;

    scanf("%d", &n);

    for (int i = 0; i < n; i++) {
        scanf("%d", &val);
        if (i % 2 == 0) {
            append(&odd, val);
        } else {
            append(&even, val);
        }
    }

    odd = insertionSortDesc(odd);
    even = insertionSortAsc(even);

    printMerged(odd, even);

    return 0;
```

```
}
```

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