Rajalakshmi Engineering College

Name: pradeepa .p
Email: 240801243@rajalakshmi.edu.in
Roll no: 2116240801243
Phone: 8270260637
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 7_MCQ_Updated

Attempt : 1
Total Mark : 20
Marks Obtained : 17

Section 1 : MCQ

1. What does a deleted slot in linear probing typically contain?

Answer

A special "deleted" marker

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

2. Which of the following values of 'm' is recommended for the division method in hashing?

Answer

A prime number

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

3. Which data structure is primarily used in linear probing?

Answer

Array

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

4. What is the worst-case time complexity for inserting an element in a hash table with linear probing?

Answer

O(n)

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

5. Which folding method divides the key into equal parts, reverses some of them, and then adds all parts?

Answer

Folding reversal method

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

6. Which of the following best describes linear probing in hashing?

Answer

Resolving collisions by linearly searching for the next free slot

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

7. What is the initial position for a key k in a linear probing hash table?

Answer

k % table_size

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

8. In C, how do you calculate the mid-square hash index for a key k, assuming we extract two middle digits and the table size is 100?

Answer

(k * k) % 100

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

9. What would be the result of folding 123456 into three parts and summing: (12 + 34 + 56)?

Answer

102

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

10. What is the primary disadvantage of linear probing?

Answer

Clustering

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

11. In the division method of hashing, the hash function is typically written as:

Answer

h(k) = k % m

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

12. Which situation causes clustering in linear probing?

Answer

Poor hash function

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

13. In division method, if key = 125 and m = 13, what is the hash index?

Answer

8

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

14. What is the output of the mid-square method for a key k = 123 if the hash table size is 10 and you extract the middle two digits of k * k?

Answer

1

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

15. In linear probing, if a collision occurs at index i, what is the next index checked?

Answer

(i + 1) % table_size

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

16. Which C statement is correct for finding the next index in linear probing?

Answer

index = (index + 1) % size;

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

17. What happens if we do not use modular arithmetic in linear probing?

Answer

Index goes out of bounds

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

18. Which of the following statements is TRUE regarding the folding method?

Answer

It divides the key into parts and adds them.

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

19. In the folding method, what is the primary reason for reversing alternate parts before addition?

Answer

To reduce the chance of collisions caused by similar digit patterns

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

20. Which of these hashing methods may result in more uniform distribution with small keys?

Answer

Division

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

Rajalakshmi Engineering College

Name: pradeepa .p
Email: 240801243@rajalakshmi.edu.in
Roll no: 2116240801243
Phone: 8270260637
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 6_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

```
#include <stdio.h>

void swap(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

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

    return i + 1;
}

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

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

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

    int iteration = 0;
    quickSort(scores, 0, n - 1, &iteration);


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

    return 0;
}
```

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

2. 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>
#include <stdlib.h>
int digitSum(int num) {
```

```
    int sum = 0;
    while (num > 0) {
        sum += num % 10;
        num /= 10;
    }
    return sum;
}
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 = (int*)malloc(n1 * sizeof(int));
    int* R = (int*)malloc(n2 * sizeof(int));
    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];
            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++;
    }


    free(L);
    free(R);
}


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 = (int*)malloc(N * sizeof(int));
    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]);
    }
    printf("\n");
```

```
    int maxDigitSum = 0;
    int maxDigitSumNumber = arr[0];
    for (int i = 0; i < N; i++) {
        int currentDigitSum = digitSum(arr[i]);
        if (currentDigitSum > maxDigitSum) {
            maxDigitSum = currentDigitSum;
            maxDigitSumNumber = arr[i];
        }
    }

    printf("The integer with the highest digit sum is: %d\n", maxDigitSumNumber);


    free(arr);

    return 0;
}
```

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

3. 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 insertionSortAndCountSwaps(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];
            j--;
            swapCount++;
        }
        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 totalSwaps = insertionSortAndCountSwaps(arr, n);

    printf("%d\n", totalSwaps);

    return 0;
}
```

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

4. 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>
#include <stdlib.h>
void merge(int arr[], int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;
    int* L = (int*)malloc(n1 * sizeof(int));
    int* R = (int*)malloc(n2 * sizeof(int));
    for (int i = 0; i < n1; i++)
        L[i] = arr[left + i];
    for (int j = 0; j < n2; j++)
        R[j] = arr[mid + 1 + j];
    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            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++;
    }
    free(L);
    free(R);
}
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 removeDuplicates(int arr[], int n) {
    if (n == 0 || n == 1) return n;

    int j = 0;
    for (int i = 1; i < n; i++) {
        if (arr[i] != arr[j]) {
            j++;
            arr[j] = arr[i];
        }
    }
    return j + 1;
}

int main() {
    int N, M;
    scanf("%d", &N);
    int* arr1 = (int*)malloc(N * sizeof(int));
    for (int i = 0; i < N; i++) {
        scanf("%d", &arr1[i]);
    }
    scanf("%d", &M);
```

```
    int* arr2 = (int*)malloc(M * sizeof(int));
    for (int i = 0; i < M; i++) {
        scanf("%d", &arr2[i]);
    }
    int totalSize = N + M;
    int* mergedArray = (int*)malloc(totalSize * sizeof(int));
    for (int i = 0; i < N; i++) {
        mergedArray[i] = arr1[i];
    }
    for (int i = 0; i < M; i++) {
        mergedArray[N + i] = arr2[i];
    }
    mergeSort(mergedArray, 0, totalSize - 1);
    int newSize = removeDuplicates(mergedArray, totalSize);
    for (int i = 0; i < newSize; i++) {
        printf("%d ", mergedArray[i]);
    }
    printf("\n");
    free(arr1);
    free(arr2);
    free(mergedArray);

    return 0;
}
```

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

5. 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>

void insertionSort(int arr[], int n) {
    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];
            j--;
        }
        arr[j + 1] = key;
    }
}

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

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

    insertionSort(arr, n);

    for (int i = 0; i < n; i++) {
```

```
        printf("%d ", arr[i]);
    }
    printf("\n");

    return 0;
}
```

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

Rajalakshmi Engineering College

Name: pradeepa .p
Email: 240801243@rajalakshmi.edu.in
Roll no: 2116240801243
Phone: 8270260637
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 7_COD_Question 2

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

Priya is developing a simple student management system. She wants to store roll numbers in a hash table using Linear Probing, and later search for specific roll numbers to check if they exist.

Implement a hash table using linear probing with the following operations:

Insert all roll numbers into the hash table.For a list of query roll numbers, print "Value x: Found" or "Value x: Not Found" depending on whether it exists in the table.

Input Format

The first line contains two integers, n and table_size — the number of roll numbers to insert and the size of the hash table.

The second line contains n space-separated integers — the roll numbers to insert.

The third line contains an integer q — the number of queries.

The fourth line contains q space-separated integers — the roll numbers to search for.

Output Format

The output print q lines — for each query value x, print: "Value x: Found" or "Value x: Not Found"

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5 10
21 31 41 51 61
3
31 60 51

Output: Value 31: Found
Value 60: Not Found
Value 51: Found

Answer

```
#include <stdio.h>

#define MAX 100


void initializeTable(int table[], int size) {
   for (int i = 0; i < size; i++) {
      table[i] = -1;
   }
}

int linearProbe(int table[], int size, int num) {
   int index = num % size;
   int original_index = index;
```

```
    while (table[index] != -1) {
        index = (index + 1) % size;
        if (index == original_index) {
            return -1;
        }
    }
    return index;
}

void insertIntoHashTable(int table[], int size, int arr[], int n) {
    for (int i = 0; i < n; i++) {
        int index = linearProbe(table, size, arr[i]);
        if (index != -1) {
            table[index] = arr[i];
        }
    }
}

int searchInHashTable(int table[], int size, int num) {
    int index = num % size;
    int original_index = index;
    while (table[index] != -1) {
        if (table[index] == num) {
            return 1;
        }
        index = (index + 1) % size;
        if (index == original_index) {
            break;
        }
    }
    return 0;
}


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

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

    initializeTable(table, table_size);
```

```
    insertIntoHashTable(table, table_size, arr, n);

    int q, x;
    scanf("%d", &q);
    for (int i = 0; i < q; i++) {
        scanf("%d", &x);
        if (searchInHashTable(table, table_size, x))
            printf("Value %d: Found\n", x);
        else
            printf("Value %d: Not Found\n", x);
    }

    return 0;
}
```

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

Rajalakshmi Engineering College

Name: pradeepa .p
Email: 240801243@rajalakshmi.edu.in
Roll no: 2116240801243
Phone: 8270260637
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 7_COD_Question 4

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

Develop a program using hashing to manage a fruit contest where each fruit is assigned a unique name and a corresponding score. The program should allow the organizer to input the number of fruits and their names with scores.

Then, it should enable them to check if a specific fruit, identified by its name, is part of the contest. If the fruit is registered, the program should display its score; otherwise, it should indicate that it is not included in the contest.

Input Format

The first line consists of an integer N, representing the number of fruits in the contest.

The following N lines contain a string K and an integer V, separated by a space, representing the name and score of each fruit in the contest.

The last line consists of a string T, representing the name of the fruit to search for.

Output Format

If T exists in the dictionary, print "Key "T" exists in the dictionary.".

If T does not exist in the dictionary, print "Key "T" does not exist in the dictionary.".

Refer to the sample outputs for the formatting specifications.

Sample Test Case

Input: 2
banana 2
apple 1
Banana
Output: Key "Banana" does not exist in the dictionary.

Answer

```
int keyExists(KeyValuePair* dictionary, int size, const char* key) {
    for (int i = 0; i < size; i++) {
        if (strcmp(dictionary[i].key, key) == 0) {
            return 1;
        }
    }
    return 0;
}
```

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

Rajalakshmi Engineering College

Name: pradeepa .p
Email: 240801243@rajalakshmi.edu.in
Roll no: 2116240801243
Phone: 8270260637
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 5_PAH_Updated

Attempt : 1
Total Mark : 50
Marks Obtained : 50

Section 1 : Coding

1. Problem Statement

Joseph, a computer science student, is interested in understanding binary search trees (BST) and their node arrangements. He wants to create a program to explore BSTs by inserting elements into a tree and displaying the nodes using post-order traversal of the tree.

Write a program to help Joseph implement the program.

Input Format

The first line of input consists of an integer N, representing the number of elements to insert into the BST.

The second line consists of N space-separated integers data, which is the data to be inserted into the BST.

Output Format

The output prints N space-separated integer values after the post-order traversal.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 4
10 15 5 3

Output: 3 5 15 10

Answer

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


struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};


struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

struct Node* insert(struct Node* root, int data) {
    if (root == NULL) {
        return createNode(data);
    }
    if (data < root->data) {
        root->left = insert(root->left, data);
```

```
    } else {
        root->right = insert(root->right, data);
    }
    return root;
}


void postOrder(struct Node* root) {
    if (root == NULL) {
        return;
    }
    postOrder(root->left);
    postOrder(root->right);
    printf("%d ", root->data);
}

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

    struct Node* root = NULL;

    for (int i = 0; i < N; i++) {
        scanf("%d", &data);
        root = insert(root, data);
    }


    postOrder(root);
    printf("\n");

    return 0;
}
```

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

2. Problem Statement

Arun is exploring operations on binary search trees (BST). He wants to write a program with an unsorted distinct integer array that represents the

BST keys and construct a height-balanced BST from it.

After constructing, he wants to perform the following operations that can alter the structure of the tree and traverse them using a level-order traversal:

InsertionDeletion

Your task is to assist Arun in completing the program without any errors.

Input Format

The first line of input consists of an integer N, representing the number of initial keys in the BST.

The second line consists of N space-separated integers, representing the initial keys.

The third line consists of an integer X, representing the new key to be inserted into the BST.

The fourth line consists of an integer Y, representing the key to be deleted from the BST.

Output Format

The first line of output prints "Initial BST: " followed by a space-separated list of keys in the initial BST after constructing it in level order traversal.

The second line prints "BST after inserting a new node X: " followed by a space-separated list of keys in the BST after inserting X n level order traversal.

The third line prints "BST after deleting node Y: " followed by a space-separated list of keys in the BST after deleting Y n level order traversal.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
25 14 56 28 12
34

12
Output: Initial BST: 25 14 56 12 28
BST after inserting a new node 34: 25 14 56 12 28 34
BST after deleting node 12: 25 14 56 28 34

Answer

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

struct TreeNode {
    int val;
    struct TreeNode *left;
    struct TreeNode *right;
};


struct TreeNode* createNode(int val) {
    struct TreeNode* node = (struct TreeNode*)malloc(sizeof(struct TreeNode));
    node->val = val;
    node->left = node->right = NULL;
    return node;
}


struct TreeNode* insert(struct TreeNode* root, int val) {
    if (root == NULL) return createNode(val);
    if (val < root->val)
        root->left = insert(root->left, val);
    else
        root->right = insert(root->right, val);
    return root;
}


struct TreeNode* findMin(struct TreeNode* root) {
    while (root && root->left)
        root = root->left;
    return root;
}


struct TreeNode* deleteNode(struct TreeNode* root, int val) {
```

```
    if (root == NULL) return NULL;
    if (val < root->val)
        root->left = deleteNode(root->left, val);
    else if (val > root->val)
        root->right = deleteNode(root->right, val);
    else {

        if (root->left == NULL) {
            struct TreeNode* temp = root->right;
            free(root);
            return temp;
        }
        else if (root->right == NULL) {
            struct TreeNode* temp = root->left;
            free(root);
            return temp;
        }
        struct TreeNode* temp = findMin(root->right);
        root->val = temp->val;
        root->right = deleteNode(root->right, temp->val);
    }
    return root;
}


void levelOrderTraversal(struct TreeNode* root) {
    if (!root) return;
    struct TreeNode* queue[100];
    int front = 0, rear = 0;
    queue[rear++] = root;
    while (front < rear) {
        struct TreeNode* curr = queue[front++];
        printf("%d ", curr->val);
        if (curr->left) queue[rear++] = curr->left;
        if (curr->right) queue[rear++] = curr->right;
    }
}

int main() {
    int N;
    scanf("%d", &N);
    int keys[15];
```

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

    struct TreeNode* root = NULL;

    for (int i = 0; i < N; i++) {
        root = insert(root, keys[i]);
    }

    printf("Initial BST: ");
    levelOrderTraversal(root);
    printf("\n");

    int X, Y;
    scanf("%d", &X);
    scanf("%d", &Y);


    root = insert(root, X);
    printf("BST after inserting a new node %d: ", X);
    levelOrderTraversal(root);
    printf("\n");


    root = deleteNode(root, Y);
    printf("BST after deleting node %d: ", Y);
    levelOrderTraversal(root);
    printf("\n");

    return 0;
}
```

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

3. Problem Statement

Yogi is working on a program to manage a binary search tree (BST) containing integer values. He wants to implement a function that removes nodes from the tree that fall outside a specified range defined by a minimum and maximum value.

Help Yogi by writing a function that achieves this.

Input Format

The first line of input consists of an integer N, representing the number of elements to be inserted into the BST.

The second line consists of N space-separated integers, representing the elements to be inserted into the BST.

The third line consists of two space-separated integers min and max, representing the minimum value and the maximum value of the range.

Output Format

The output prints the remaining elements of the BST in an in-order traversal, after removing nodes that fall outside the specified range.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
10 5 15 20 12
5 15
Output: 5 10 12 15

Answer

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


struct TreeNode {
    int val;
    struct TreeNode* left;
    struct TreeNode* right;
};
```

```
struct TreeNode* createNode(int val) {
    struct TreeNode* node = (struct TreeNode*) malloc(sizeof(struct TreeNode));
    node->val = val;
    node->left = NULL;
    node->right = NULL;
    return node;
}


struct TreeNode* insert(struct TreeNode* root, int val) {
    if (root == NULL) return createNode(val);

    if (val < root->val)
        root->left = insert(root->left, val);
    else
        root->right = insert(root->right, val);

    return root;
}


struct TreeNode* trimBST(struct TreeNode* root, int min, int max) {
    if (root == NULL) return NULL;


    root->left = trimBST(root->left, min, max);
    root->right = trimBST(root->right, min, max);


    if (root->val < min) {
        struct TreeNode* rightChild = root->right;
        free(root);
        return rightChild;
    }


    if (root->val > max) {
        struct TreeNode* leftChild = root->left;
        free(root);
        return leftChild;
    }
```

```
    return root;
}
void inorderTraversal(struct TreeNode* root) {
    if (root == NULL) return;
    inorderTraversal(root->left);
    printf("%d ", root->val);
    inorderTraversal(root->right);
}

int main() {
    int N;
    scanf("%d", &N);
    int values[20];
    for (int i = 0; i < N; i++) {
        scanf("%d", &values[i]);
    }

    int min, max;
    scanf("%d %d", &min, &max);

    struct TreeNode* root = NULL;
    for (int i = 0; i < N; i++) {
        root = insert(root, values[i]);
    }

    root = trimBST(root, min, max);

    inorderTraversal(root);
    printf("\n");

    return 0;
}
```

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

4. Problem Statement

Viha, a software developer, is working on a project to automate searching for a target value in a Binary Search Tree (BST). She needs to create a program that takes an integer target value as input and determines if that

value is present in the BST or not.

Write a program to assist Viha.

Input Format

The first line of input consists of integers separated by spaces, which represent the elements to be inserted into the BST. The input is terminated by entering -1.

The second line consists of an integer target, which represents the target value to be searched in the BST.

Output Format

If the target value is found in the BST, print "[target] is found in the BST".

Else, print "[target] is not found in the BST"

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5 3 7 1 4 6 8 -1
4

Output: 4 is found in the BST

Answer

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

// Definition of tree node
struct TreeNode {
    int val;
    struct TreeNode* left;
    struct TreeNode* right;
};

// Create new node
struct TreeNode* createNode(int val) {
    struct TreeNode* node = (struct TreeNode*)malloc(sizeof(struct TreeNode));
```

```
    node->val = val;
    node->left = node->right = NULL;
    return node;
}

// Insert into BST
struct TreeNode* insert(struct TreeNode* root, int val) {
    if (root == NULL) return createNode(val);
    if (val < root->val)
        root->left = insert(root->left, val);
    else
        root->right = insert(root->right, val);
    return root;
}

// Search BST for target
int search(struct TreeNode* root, int target) {
    if (root == NULL) return 0;
    if (root->val == target) return 1;
    if (target < root->val)
        return search(root->left, target);
    else
        return search(root->right, target);
}

int main() {
    struct TreeNode* root = NULL;
    int val;

    // Read integers until -1 encountered
    while (1) {
        if (scanf("%d", &val) != 1) break;
        if (val == -1) break;
        root = insert(root, val);
    }

    int target;
    scanf("%d", &target);

    if (search(root, target))
        printf("%d is found in the BST\n", target);
    else
```

```
    printf("%d is not found in the BST\n", target);

    return 0;
}
```

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

5. Problem Statement

Aishu is participating in a coding challenge where she needs to reconstruct a Binary Search Tree (BST) from given preorder traversal data and then print the in-order traversal of the reconstructed BST.

Since Aishu is just learning about tree data structures, she needs your help to write a program that does this efficiently.

Input Format

The first line consists of an integer n, representing the number of nodes in the BST.

The second line of input contains n integers separated by spaces, which represent the preorder traversal of the BST.

Output Format

The output displays n space-separated integers, representing the in-order traversal of the reconstructed BST.

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 6
10 5 1 7 40 50
Output: 1 5 7 10 40 50

Answer

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

struct TreeNode {
    int val;
    struct TreeNode* left;
    struct TreeNode* right;
};


struct TreeNode* createNode(int val) {
    struct TreeNode* node = (struct TreeNode*) malloc(sizeof(struct TreeNode));
    node->val = val;
    node->left = NULL;
    node->right = NULL;
    return node;
}


struct TreeNode* constructBST(int* preorder, int* preorderIndex, int n, int min, int
max) {
    if (*preorderIndex >= n) return NULL;
    int val = preorder[*preorderIndex];
    if (val < min || val > max) return NULL;


    struct TreeNode* root = createNode(val);
    (*preorderIndex)++;


    root->left = constructBST(preorder, preorderIndex, n, min, val - 1);

    root->right = constructBST(preorder, preorderIndex, n, val + 1, max);

    return root;
}

void inOrderTraversal(struct TreeNode* root) {
    if (root == NULL) return;
    inOrderTraversal(root->left);
    printf("%d ", root->val);
```

```
    inOrderTraversal(root->right);
}

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

    int preorder[15];
    for (int i = 0; i < n; i++) {
        scanf("%d", &preorder[i]);
    }

    int preorderIndex = 0;
    struct TreeNode* root = constructBST(preorder, &preorderIndex, n, INT_MIN,
INT_MAX);

    inOrderTraversal(root);
    printf("\n");

    return 0;
}
```

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

Rajalakshmi Engineering College

Name: pradeepa .p
Email: 240801243@rajalakshmi.edu.in
Roll no: 2116240801243
Phone: 8270260637
Branch: REC
Department: I ECE AF
Batch: 2028
Degree: B.E - ECE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 7_COD_Question 1

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

Ravi is building a basic hash table to manage student roll numbers for quick lookup. He decides to use Linear Probing to handle collisions.

Implement a hash table using linear probing where:

The hash function is: index = roll_number % table_sizeOn collision, check subsequent indexes (i+1, i+2, ...) until an empty slot is found.

You need to:

Insert a list of n student roll numbers into the hash table.Print the final state of the hash table. If a slot is empty, print -1.

Input Format

The first line of the input contains two integers n and table_size, where n is the

number of roll numbers to be inserted, and table_size is the size of the hash table.

The second line contains n space-separated integers — the roll numbers to insert into the hash table.

Output Format

The output should print a single line with table_size space-separated integers representing the final state of the hash table after all insertions.

If any slot remains unoccupied, it should be represented as -1.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 4 7
50 700 76 85

Output: 700 50 85 -1 -1 -1 76

Answer

```
#include <stdio.h>

#define MAX 100

void initializeTable(int table[], int size) {
    for (int i = 0; i < size; i++) {
        table[i] = -1;
    }
}

int linearProbe(int table[], int size, int num) {
    int index = num % size;
    int original_index = index;
    while (table[index] != -1) {
        index = (index + 1) % size;
        if (index == original_index) {
            return -1;
        }
```

```
    }
    return index;
}

void insertIntoHashTable(int table[], int size, int arr[], int n) {
    initializeTable(table, size);
    for (int i = 0; i < n; i++) {
        int index = linearProbe(table, size, arr[i]);
        if (index != -1) {
            table[index] = arr[i];
        }
    }
}

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


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

    int arr[MAX];
    int table[MAX];

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

    initializeTable(table, table_size);
    insertIntoHashTable(table, table_size, arr, n);
    printTable(table, table_size);

    return 0;
}
```

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