LAB TASK-5

1. Implementation of finding the maximum and minimum element using divide and conquer strategy. Analyse and describe how the divide and conquer strategy is better when compared to traditional approach.

```c
#include <stdio.h>



void merge(int arr[], int l, int m, int r) {
    int n1 = m - l + 1;
    int n2 = r - m;

    int left[n1], right[n2];



    for (int i = 0; i < n1; i++)
        left[i] = arr[l + i];
    for (int j = 0; j < n2; j++)
        right[j] = arr[m + 1 + j];

    int i = 0, j = 0, k = l;

    while (i < n1 && j < n2) {
        if (left[i] <= right[j]) {
            arr[k++] = left[i++];
        } else {
            arr[k++] = right[j++];
        }
    }

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

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

```c
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 main() {
    int n;
    printf("Enter number of elements: ");
    scanf("%d", &n);

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

    mergeSort(arr, 0, n - 1);

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

    printf("\nMinimum element: %d", arr[0]);
    printf("\nMaximum element: %d", arr[n - 1]);

    return 0;
}
```

```
Sorted array: 2 5 6 8 9
Minimum element: 2
Maximum element: 9
```

2. Divide and conquer: Implementation of maximum-subarray problem.

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


int max(int a, int b) {
    return (a > b) ? a : b;
}


int max3(int a, int b, int c) {
    return max(max(a, b), c);
}


int maxCrossingSum(int arr[], int l, int m, int r) {
    int sum = 0;
    int left_sum = INT_MIN;


    for (int i = m; i >= l; i--) {
        sum += arr[i];
        if (sum > left_sum)
            left_sum = sum;
    }

    sum = 0;
    int right_sum = INT_MIN;


    for (int i = m + 1; i <= r; i++) {
        sum += arr[i];
        if (sum > right_sum)
            right_sum = sum;
    }

    return left_sum + right_sum;
}


int maxSubArraySum(int arr[], int l, int r) {
    if (l == r)
```

```
        return arr[l];

    int m = (l + r) / 2;

    return max3(maxSubArraySum(arr, l, m),
                maxSubArraySum(arr, m + 1, r),
                maxCrossingSum(arr, l, m, r));
}

int main() {
    int n;
    printf("Enter number of elements: ");
    scanf("%d", &n);

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

    int max_sum = maxSubArraySum(arr, 0, n - 1);
    printf("Maximum Subarray Sum is %d\n", max_sum);

    return 0;
}
```

```
Enter number of elements: 5
Enter 5 elements: 4 3 7 9 29
Maximum Subarray Sum is 52
PS C:\Users\KIIT0001\DAA  LAB>
```

3. Write a program to implement Hash Table with Chaining Method. Perform the Insert, Search and Delete operation on Hash Table by taking user choices as:
4. Insert
5. Search
6. Delete

7. Display
8. Exit

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

#define SIZE 10
struct Node {
    int data;
    struct Node* next;
};


struct Node* hashTable[SIZE];


int hashFunction(int key) {
    return key % SIZE;
}


void insert(int key) {
    int index = hashFunction(key);
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = key;
    newNode->next = hashTable[index];
    hashTable[index] = newNode;
    printf("%d inserted.\n", key);
}


void search(int key) {
    int index = hashFunction(key);
    struct Node* temp = hashTable[index];
    while (temp != NULL) {
        if (temp->data == key) {
            printf("%d found at index %d.\n", key, index);
            return;
        }
        temp = temp->next;
    }
```

```
    printf("%d not found.\n", key);
}


void delete(int key) {
    int index = hashFunction(key);
    struct Node* temp = hashTable[index];
    struct Node* prev = NULL;

    while (temp != NULL) {
        if (temp->data == key) {
            if (prev == NULL) {
                hashTable[index] = temp->next;
            } else {
                prev->next = temp->next;
            }
            free(temp);
            printf("%d deleted.\n", key);
            return;
        }
        prev = temp;
        temp = temp->next;
    }
    printf("%d not found, cannot delete.\n", key);
}


void display() {
    for (int i = 0; i < SIZE; i++) {
        printf("Bucket %d: ", i);
        struct Node* temp = hashTable[i];
        while (temp != NULL) {
            printf("%d -> ", temp->data);
            temp = temp->next;
        }
        printf("NULL\n");
    }
}


int main() {
    int choice, key;
    while (1) {
```

```
        printf("\n--- Hash Table Menu ---\n");
        printf("1. Insert\n");
        printf("2. Search\n");
        printf("3. Delete\n");
        printf("4. Display\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &key);
                insert(key);
                break;
            case 2:
                printf("Enter value to search: ");
                scanf("%d", &key);
                search(key);
                break;
            case 3:
                printf("Enter value to delete: ");
                scanf("%d", &key);
                delete(key);
                break;
            case 4:
                display();
                break;
            case 5:
                exit(0);
            default:
                printf("Invalid choice. Try again.\n");
        }
    }
    return 0;
}
```

```
Enter your choice: 1
Enter value to insert: 4
4 inserted.
```