DATA STRUCTURE

DAY - 1 24/7/2024 CSA0390

1) Write a c program for binary search.

```
#include <stdio.h>

int binarySearch(int arr[], int left, int right, int item){

    if (right >= left){

        int mid = left + (right - left)/2;

        if (arr[mid] == item)
            return mid;

        if (arr[mid] > item)
            return binarySearch(arr, left, mid-1, item);

        else
            return binarySearch(arr, mid+1, right, item);
    }
    else
       return -1;
}
int main(){
    // array needs to be sorted to impliment binary search
    int arr[8] = {10, 20, 30, 40, 50, 60, 70, 80};
    int n = sizeof(arr) / sizeof(arr[0]);
    int item = 70;

    int position = binarySearch(arr, 0, n-1, item);

    if(position == -1)
        printf("%d Not Found",item);
    else
        printf("%d Found at index : %d",item, position);
}
```

output :

```
70 Found at index : 6
```

2) write a c program for liner search.

```c
      #include <stdio.h>

void LinearSearch(int arr[], int len, int item)
{
    for(int i=0;i < len;i++)
    {
        if(arr[i] == item)
        {
        printf("%d Found at index %d", item, i);
        return;
        }
    }
    printf("Not Found");
}

int main()
{
    int arr[] = {10, 20, 30, 40, 50};


    int len = sizeof(arr)/sizeof(arr[0]);

  int item = 40;
    LinearSearch(arr, len, item);

    return 0;
}
```

output :

```
40 Found at index 3
```

3) write a c program to implement following operations

1. traverse
2. search
3. insert
4. delete
5. update

```c
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node *left, *right;
};
struct Node* newNode(int data) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = data;
```

```
    node->left = node->right = NULL;
    return node;
}
void inOrder(struct Node* root) {
    if (root != NULL) {
        inOrder(root->left);
        printf("%d ", root->data);
        inOrder(root->right);
    }
}struct Node* search(struct Node* root, int key) {
    if (root == NULL || root->data == key)
        return root;
    if (root->data < key)
        return search(root->right, key);
    return search(root->left, key);
}
struct Node* insert(struct Node* node, int data) {
    if (node == NULL) return newNode(data);
    if (data < node->data)
        node->left = insert(node->left, data);
    else if (data > node->data)
        node->right = insert(node->right, data);
    return node;
}
struct Node* minValueNode(struct Node* node) {
    struct Node* current = node;
    while (current && current->left != NULL)
        current = current->left;
    return current;
}
struct Node* deleteNode(struct Node* root, int key) {
    if (root == NULL) return root;
    if (key < root->data)
        root->left = deleteNode(root->left, key);
    else if (key > root->data)
        root->right = deleteNode(root->right, key);
    else {
        if (root->left == NULL) {
            struct Node* temp = root->right;
            free(root);
            return temp;
        } else if (root->right == NULL) {
            struct Node* temp = root->left;
            free(root);
            return temp;
        }
        struct Node* temp = minValueNode(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}
```

```
struct Node* update(struct Node* root, int oldData, int newData) {
    root = deleteNode(root, oldData);
    root = insert(root, newData);
    return root;
}
int main() {
    struct Node* root = NULL;
    root = insert(root, 50);
    root = insert(root, 30);
    root = insert(root, 20);
    root = insert(root, 40);
    root = insert(root, 70);
    root = insert(root, 60);
    root = insert(root, 80);
    printf("In-order traversal: ");
    inOrder(root);
    printf("\n");
    printf("Search for 40: %s\n", search(root, 40) != NULL ? "Found" :
"Not Found");
    printf("Delete 20\n");
    root = deleteNode(root, 20);
    printf("In-order traversal: ");
    inOrder(root);
    printf("\n");
    printf("Update 30 to 35\n");
    root = update(root, 30, 35);
    printf("In-order traversal: ");
    inOrder(root);
    printf("\n");
    return 0;
}
```

output :

```
Update 30 to 35
In-order traversal: 35 40 50 60 70 80
Delete 20
In-order traversal: 30 40 50 60 70 80
Update 30 to 35
In-order traversal: 35 40 50 60 70 80
```

4) writing a recursive function to calculate the factorial of a number.

```
#include <stdio.h>

int factorial(int n) {
    if (n == 0) {
        return 1;
```

```
    } else {
        return n * factorial(n - 1);
    }
}

int main() {
    int number = 5;
    int result = factorial(number);
    printf("Factorial of %d = %d", number, result);
    return 0;
}
```

output :

```
Factorial of 5 = 120
```

5) write a c program to find duplicate element in an array.

```
#include <stdio.h>

int main() {
    int arr[] = {1, 2, 3, 4, 2, 5, 6, 3};
    int size = sizeof(arr) / sizeof(arr[0]);

    printf("Duplicate elements in the array are: ");
    for (int i = 0; i < size; i++) {
        for (int j = i + 1; j < size; j++) {
            if (arr[i] == arr[j]) {
                printf("%d ", arr[j]);
                break;
            }
        }
    }

    return 0;
}
```

output :

```
Duplicate elements in the array are: 2 3
```

6) write c program to find max and min from an array elements.

```
#include <stdio.h>

int main() {
    int arr[] = {10, 5, 8, 20, 15};
    int n = sizeof(arr) / sizeof(arr[0]);
    int max = arr[0];
```

```
    int min = arr[0];

    for (int i = 1; i < n; i++) {
        if (arr[i] > max) {
            max = arr[i];
        }
        if (arr[i] < min) {
            min = arr[i];
        }
    }

    printf("Maximum element in the array: %d\n", max);
    printf("Minimum element in the array: %d\n", min);

    return 0;
}
```

output :

```
Maximum element in the array: 20
Minimum element in the array: 5
```

7) given a number n. the task is to print the fibonacci series and the sum of the series using
recursion.
input : n = 10
output : fibonacci series
0, 1, 1, 2, 3, 5, 8, 13, 21, 34
sum : 88

```
#include <stdio.h>

int fibonacci(int n) {
    if (n <= 1)
        return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}

int main() {
    int n = 10;
    int sum = 0;

    printf("Fibonacci Series:\n");
    for (int i = 0; i < n; i++) {
        printf("%d, ", fibonacci(i));
        sum += fibonacci(i);
    }

    printf("\nSum: %d\n", sum);
```

```
    return 0;
}
```

output:

```
Fibonacci Series:
0, 1, 1, 2, 3, 5, 8, 13, 21, 34,
Sum: 88
```

8) you are given an array arr in increasing oerder. find the element x from arr using binary search.
example 1 : arr = {1,5,6,7,9,10} , x = 6
output : element found at location 2
example 2 : arr={1,5,6,7,9,10}, x = 11
output : element not found at location 2

```
#include <stdio.h>

int binarySearch(int arr[], int left, int right, int x) {
    while (left <= right) {
        int mid = left + (right - left) / 2;

        if (arr[mid] == x)
            return mid;

        if (arr[mid] < x)
            left = mid + 1;
        else
            right = mid - 1;
    }

    return -1;
}

int main() {
    int arr[] = {1, 5, 6, 7, 9, 10};
    int x = 6;
    int n = sizeof(arr) / sizeof(arr[0]);
    int result = binarySearch(arr, 0, n - 1, x);

    if (result == -1)
        printf("Element not found\n");
    else
        printf("Element found at location %d\n", result);

    return 0;
}
```

output :

Element found at location 2