1. **You are given with an array arr which contains integer elements. Sort these elements in ascending order using insertion sort and print the 6th Iteration result.**

**Example:**

**Input:98,23,45,14,6,67,33,42**

**Output:6,14,23,33,45,67,98,42**

#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 arr[] = {98, 23, 45, 14, 6, 67, 33, 42};

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

insertionSort(arr, n);

printf("Result after 6th iteration: ");

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

printf("%d", arr[i]);

if (i < n - 1) {

printf(",");

}

}

printf("\n");

return 0;

}

2.**You are given an array arr in increasing order. 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 n = sizeof(arr) / sizeof(arr[0]);

int x;

scanf("%d", &x);

int result = binarySearch(arr, 0, n - 1, x);

if (result != -1)

printf("Element found at location %d.\n", result);

else

printf("Element %d not found in the array.\n", x);

return 0;

}

3.**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);

}

void printFibonacci(int n)

{

int sum = 0;

printf("Fibonacci series: ");

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

{

int fib = fibonacci(i);

sum += fib;

printf("%d, ", fib);

}

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

}

int main()

{

int n = 10;

printFibonacci(n);

return 0;

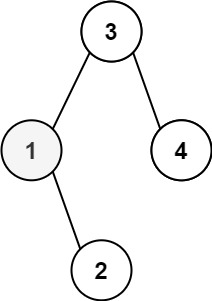
}

4.**Given the root of a binary search tree and K as input, find Kth smallest element in BST.**

**For example, in the following BST, **

**if k = 3, then the output should be 10, and**

**if k = 5, then the output should be 14.**



#include <stdio.h>

#include <stdlib.h>

struct Node

{

int data;

struct Node \*left, \*right;

};

struct Node\* createNode(int value) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = value;

newNode->left = newNode->right = NULL;

return newNode;

}

void kthSmallestUtil(struct Node\* root, int k, int\* count, int\* result) {

if (root == NULL || \*count >= k) {

return;

}

kthSmallestUtil(root->left, k, count, result);

(\*count)++;

if (\*count == k) {

\*result = root->data;

return;

}

kthSmallestUtil(root->right, k, count, result);

}

int kthSmallest(struct Node\* root, int k) {

int count = 0;

int result = -1;

kthSmallestUtil(root, k, &count, &result);

return result;

}

int main() {

struct Node\* root = createNode(20);

root->left = createNode(8);

root->right = createNode(22);

root->left->left = createNode(4);

root->left->right = createNode(12);

root->left->right->left = createNode(10);

root->left->right->right = createNode(14);

int k = 3;

printf("Kth smallest element for k = %d is: %d\n", k, kthSmallest(root, k));

k = 5;

printf("Kth smallest element for k = %d is: %d\n", k, kthSmallest(root, k));

Return 0;

}

5.**Given a string s, find the frequency of characters**

**Example 1:**

**Input: s = "tree"**

**Output t->1, r->1, e->2**

#include <stdio.h>

void charFrequency(char\* s)

{

int count[256] = {0};

for (int i = 0; s[i]; i++) {

count[s[i]]++;

}

printf("Character frequencies:\n");

for (int i = 0; i < 256; i++) {

if (count[i] != 0) {

printf("%c -> %d\n", i, count[i]);

}

}

}

int main()

{

char s[] = "tree";

printf("Input: s = \"%s\"\n", s);

charFrequency(s);

return 0;

}

6.**Given the head of a singly linked list, return true if it is a palindrome or false otherwise.**

**Example 1:**

**Input: head = [1,2,2,1]**

**Output: true**

**Example 2:**

**Input: head = [1,2]**

**Output: false**

**Input: R->A->D->A->R->NULL**

**Output: Yes**

**Input: C->O->D->E->NULL**

**Output: No**

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

struct ListNode {

int val;

struct ListNode \*next;

};

bool isPalindrome(struct ListNode\* head) {

struct ListNode \*slow = head, \*fast = head;

while (fast && fast->next) {

slow = slow->next;

fast = fast->next->next;

}

struct ListNode \*prev = NULL, \*curr = slow, \*next;

while (curr) {

next = curr->next;

curr->next = prev;

prev = curr;

curr = next;

}

struct ListNode \*left = head, \*right = prev;

while (right) {

if (left->val != right->val) {

return false;

}

left = left->next;

right = right->next;

}

return true;

}

int main() {

struct ListNode \*head = malloc(sizeof(struct ListNode));

head->val = 1;

head->next = malloc(sizeof(struct ListNode));

head->next->val = 2;

head->next->next = malloc(sizeof(struct ListNode));

head->next->next->val = 2;

head->next->next->next = malloc(sizeof(struct ListNode));

head->next->next->next->val = 1;

head->next->next->next->next = NULL;

if (isPalindrome(head)) {

printf("Output: true\n");

} else {

printf("Output: false\n");

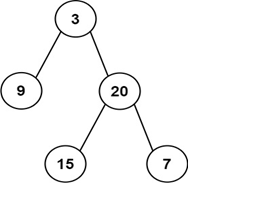
}

return 0;

}

7.**Given two integer arrays preorder and inorder where preorder is the preorder**

**traversal of a binary tree and inorder is the inorder traversal of the same tree,**

**construct and return the binary tree.**

**Input: preorder = [3,9,20,15,7], inorder = [9,3,15,20,7]**

**Output: [3,9,20,null,null,15,7]**

#include <stdio.h>

#include <stdlib.h>

struct TreeNode {

int val;

struct TreeNode \*left;

struct TreeNode \*right;

};

struct TreeNode\* buildTree(int\* preorder, int\* inorder, int inStart, int inEnd, int\* preIndex) {

if (inStart > inEnd)

return NULL;

struct TreeNode\* root = (struct TreeNode\*)malloc(sizeof(struct TreeNode));

root->val = preorder[(\*preIndex)++];

int inIndex;

for (inIndex = inStart; inIndex <= inEnd; inIndex++) {

if (inorder[inIndex] == root->val)

break;

}

root->left = buildTree(preorder, inorder, inStart, inIndex - 1, preIndex);

root->right = buildTree(preorder, inorder, inIndex + 1, inEnd, preIndex);

return root;

}

void printLevelOrder(struct TreeNode\* root) {

if (root == NULL)

return;

struct TreeNode\* queue[100];

int front = -1, rear = -1;

queue[++rear] = root;

while (front != rear) {

struct TreeNode\* temp = queue[++front];

if (temp != NULL) {

printf("%d", temp->val);

if (temp->left || temp->right) {

queue[++rear] = temp->left;

queue[++rear] = temp->right;

}

if (front != rear)

printf(",");

} else {

printf("null");

if (front != rear)

printf(",");

}

}

}

int main() {

int preorder[] = {3, 9, 20, 15, 7};

int inorder[] = {9, 3, 15, 20, 7};

int n = sizeof(preorder) / sizeof(preorder[0]);

int preIndex = 0;

struct TreeNode\* root = buildTree(preorder, inorder, 0, n - 1, &preIndex);

printf("[");

printLevelOrder(root);

printf("]\n");

return 0;

}

8.**Write a program to sort the below numbers in descending order using bubble sort**

**Input 4,7,9,1,2**

**Output:9,7,4,2,1**

#include <stdio.h>

void bubbleSort(int arr[], int n) {

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

if (arr[j] < arr[j + 1]) {

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}

int main() {

int arr[] = {4, 7, 9, 1, 2};

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

bubbleSort(arr, n);

printf("Sorted array in descending order: ");

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

printf("%d ", arr[i]);

printf("\n");

return 0;

}

9.**Write a program to find odd number present in the data part of a node**

**Example Linked List 1->2->3->7**

**Output: 1,3,7**

#include <stdio.h>

#include <stdlib.h>

struct Node { int data; struct Node\* next; };

void findOddNumbers(struct Node\* h)

{

printf("Odd numbers: ");

while (h) { if (h->data % 2) printf("%d ", h->data); h = h->next; } printf("\n");

}

int main()

{

struct Node\* h = NULL; int v[] = {1, 2, 3, 7};

for (int i = sizeof(v)/sizeof(v[0])-1; i >= 0; i--) {

struct Node\* n = (struct Node\*)malloc(sizeof(struct Node)); n->data = v[i];

n->next = h; h = n;

}

findOddNumbers(h);

while (h) { struct Node\* t = h; h = h->next; free(t); }

return 0;

}

10.**Given two strings needle and haystack, return the index of the first occurrence**

**of needle in haystack, or -1 if needle is not part of haystack.**

**Example 1:**

**Input: haystack = "sadbutsad", needle = "sad"**

**Output: 0**

**Explanation: "sad" occurs at index 0 and 6.**

**The first occurrence is at index 0, so we return 0.**

**Input: haystack = "leetcode", needle = "leeto"**

**Output: -1**

**Explanation: "leeto" did not occur in "leetcode", so we return -1.**

#include <stdio.h>

#include <string.h>

int indexOfSubstring(char\* haystack, char\* needle) {

char\* result = strstr(haystack, needle);

if (result == NULL)

return -1;

else

return result - haystack;

}

int main() {

char haystack1[] = "sadbutsad";

char needle1[] = "sad";

printf("Input: haystack = \"%s\", needle = \"%s\"\n", haystack1, needle1);

printf("Output: %d\n", indexOfSubstring(haystack1, needle1));

char haystack2[] = "leetcode";

char needle2[] = "leeto";

printf("Input: haystack = \"%s\", needle = \"%s\"\n", haystack2, needle2);

printf("Output: %d\n", indexOfSubstring(haystack2, needle2));

return 0;

}

11.**Given an array arr, sort the elements in descending order using bubblesort.**

**Arr=[9,10,-9,23,67,-90]**

**Output:[67,23,10,9,-9,-90]**

#include <stdio.h>

void bubbleSort(int arr[], int n)

{

for (int i = 0; i < n - 1; i++)

{

for (int j = 0; j < n - i - 1; j++)

{

if (arr[j] < arr[j + 1])

{

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}

int main()

{

int n;

printf("Enter the number of elements in the array: ");

scanf("%d", &n);

int arr[n];

printf("Enter %d elements:\n", n);

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

{

printf("Enter element %d: ", i + 1);

scanf("%d", &arr[i]);

}

bubbleSort(arr, n);

printf("Sorted array in descending order: ");

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

{

printf("%d ", arr[i]);

}

printf("\n");

return 0;

}

12.**Given an array arr, sort the elements in ascending order using Bubble sort.**

**Arr=[9,10,-9,23,67,-90]**

**Output:[-90,-9,9,10,23,67]**

#include <stdio.h>

void bubbleSort(int arr[], int n)

{

int i, j, temp;

for (i = 0; i < n-1; i++)

{

for (j = 0; j < n-i-1; j++)

{

if (arr[j] > arr[j+1])

{

temp = arr[j];

arr[j] = arr[j+1];

arr[j+1] = temp;

}

}

}

}

int main()

{

int arr[] = {9, 10, -9, 23, 67, -90};

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

bubbleSort(arr, n);

printf("Sorted array: ");

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

{

printf("%d ", arr[i]);

}

printf("\n");

return 0;

}

13.Y**ou have been given a positive integer N. You need to find and print the**

**Factorial of this number without using recursion. The Factorial of a positive**

**integer N refers to the product of all number in the range from 1 to N.**

**Input: N=2**

**Output: 2**

**Input: N=4**

**Output: 24**

#include <stdio.h>

int factorial(int N)

{

int result = 1;

for (int i = 1; i <= N; i++)

{

result \*= i;

}

return result;

}

int main()

{

int N;

scanf("%d", &N);

if (N < 0)

{

printf("Factorial is not defined for negative numbers.\n");

} else

{

printf("Factorial of %d is: %d", N, factorial(N));

}

return 0;

}

14.**Write a program to create and display a linked list**

**Example 1:**

**Nodes : 6,7,8,9**

**Output: 6->7->8->9**

#include <stdio.h>

#include <stdlib.h>

struct ListNode {

int val;

struct ListNode\* next;

};

struct ListNode\* createNode(int val) {

struct ListNode\* newNode = (struct ListNode\*)malloc(sizeof(struct ListNode));

newNode->val = val;

newNode->next = NULL;

return newNode;

}

struct ListNode\* insertEnd(struct ListNode\* head, int val) {

struct ListNode\* newNode = createNode(val);

if (head == NULL) {

head = newNode;

} else {

struct ListNode\* current = head;

while (current->next != NULL) {

current = current->next;

}

current->next = newNode;

}

return head;

}

void displayList(struct ListNode\* head) {

struct ListNode\* current = head;

while (current != NULL) {

printf("%d", current->val);

if (current->next != NULL) {

printf("->");

}

current = current->next;

}

printf("\n");

}

int main() {

struct ListNode\* head = NULL;

head = insertEnd(head, 6);

head = insertEnd(head, 7);

head = insertEnd(head, 8);

head = insertEnd(head, 9);

printf("Output: ");

displayList(head);

return 0;

}

15.**Given the head of a singly linked list, return number of nodes present in a linked**

**Example 1:**

**1->2->3->5->8**

**Output 5**

#include <stdio.h>

#include <stdlib.h>

struct ListNode

{

int val;

struct ListNode \*next;

};

int countNodes(struct ListNode\* head) {

int count = 0;

while (head != NULL) {

count++;

head = head->next;

}

return count;

}

int main()

{

struct ListNode \*head = malloc(sizeof(struct ListNode));

head->val = 1;

head->next = malloc(sizeof(struct ListNode));

head->next->val = 2;

head->next->next = malloc(sizeof(struct ListNode));

head->next->next->val = 3;

head->next->next->next = malloc(sizeof(struct ListNode));

head->next->next->next->val = 5;

head->next->next->next->next = malloc(sizeof(struct ListNode));

head->next->next->next->next->val = 8;

head->next->next->next->next->next = NULL;

printf("Number of nodes: %d\n", countNodes(head));

struct ListNode \*current = head;

struct ListNode \*temp;

while (current != NULL)

{

temp = current;

current = current->next;

free(temp);

}

return 0;

}

16.**Given a string s, sort it in ascending order and find the starting index of repeated character**

**Input: s = "tree"**

**Output: "eert", starting index 0**

**Input: s = "kkj"**

**Output: "jkk", starting index : 1**

**Example 2:**

**Input: s = "cccaaa"**

**Output: "aaaccc", starting index 0,3**

**Example 3:**

**Input: s = "Aabb"**

#include <stdio.h>

#include <string.h>

void findRepeatedCharacter(char \*str) {

int count[256] = {0};

for (int i = 0; str[i]; i++) count[str[i]]++;

for (int i = 0; str[i]; i++) {

if (count[str[i]] > 1) {

printf("\"");

for (int j = 0; str[j]; j++) printf("%c", str[j]);

printf("\", starting index %d\n", i);

return;

}

}

printf("No repeated character found.\n");

}

int main() {

char s[100];

printf("Input: s = ");

fgets(s, sizeof(s), stdin);

s[strcspn(s, "\n")] = '\0';

findRepeatedCharacter(s);

return 0;

}

17.**Given the head of a linked list, insert the node in nth place and return its head.**

**Input: head = [1,3,2,3,4,5], p=3 n = 2**

**Output: [1,3,2,3,4,5]**

**Input: head = [1], p = 0, n = 1**

**Output: [0,1]**

**Input: head = [1,2], p=3, n = 3**

**Output: [1,2,3]**

#include <stdio.h>

#include <stdlib.h>

struct ListNode

{

int val;

struct ListNode \*next;

};

struct ListNode\* insertNth(struct ListNode\* head, int val, int p, int n) {

struct ListNode dummy = {0, head}, \*prev = &dummy, \*newNode = malloc(sizeof(struct ListNode));

newNode->val = val;

while (p-- >= 0 && prev->next) {

if (p == 0) {

newNode->next = prev->next;

prev->next = newNode;

}

prev = prev->next;

}

while (n-- > 0 && prev->next) prev = prev->next;

return dummy.next;

}

void display(struct ListNode\* head) {

while (head) {

printf("%d ", head->val);

head = head->next;

}

printf("\n");

}

int main() {

struct ListNode\* head1 = (struct ListNode\*)malloc(sizeof(struct ListNode));

head1->val = 1;

head1->next = NULL;

display(insertNth(head1, 3, 2, 3));

struct ListNode\* head2 = (struct ListNode\*)malloc(sizeof(struct ListNode));

head2->val = 1;

head2->next = NULL;

display(insertNth(head2, 0, 0, 1));

struct ListNode\* head3 = (struct ListNode\*)malloc(sizeof(struct ListNode));

head3->val = 1;

head3->next = (struct ListNode\*)malloc(sizeof(struct ListNode));

head3->next->val = 2;

head3->next->next = NULL;

display(insertNth(head3, 3, 3, 3));

return 0;

}