November 11, 2023

Darshil Solanki

data structure and algorithms

assignment

Certificate

This is to certify that DARSHILKUMAR MANSUKHBHAI SOLANKI of MCA SEM-1 has successfully completed the DATA STRUCTURE AND ALGORITHMS lab work during academic year 20203-24.

Date of Completion: 04/11/2023

Signature of HOD Signature of Professor

Table of Contents

[1. Introduction to array. Create an array to perform the following operation. 3](#_Toc151896553)

[2. Introduction to string. Perform the following operation on string. 4](#_Toc151896554)

[3. Implement a program for stack that performs following operations using arrays. 6](#_Toc151896555)

[4. Implement a program to convert infix notation to postfix notation using stack. 8](#_Toc151896556)

[5. Write a program to implement Queue using arrays that performs following operations. 10](#_Toc151896557)

[6. Write a program to implement Circular Queue using arrays that performs following operations. 12](#_Toc151896558)

[7. Write a menu driven program to implement following operations on the singly linked list. 14](#_Toc151896559)

[8. Write a program to implement following operations on the doubly linked list. 19](#_Toc151896560)

[9. Write a program to create a Binary search tree. 23](#_Toc151896561)

[10. Implement recursive and non-recursive tree traversing methods in-order,pre-order and post-order traversal. 24](#_Toc151896562)

[11. Implement AVL tree. 26](#_Toc151896563)

[12. Implement heap sort. 30](#_Toc151896564)

[13. Create an one dimensional array with values 22,4,67,34,12,87,89,2656,9. Perform the following sorting operation on an array to sort the data in ascending order. 31](#_Toc151896565)

[15. Write a C program to implement DFS traversal using Adjacency Matrix in a Graph. 36](#_Toc151896566)

1. Introduction to array. Create an array to perform the following operation.

1. Find the second largest element in the array.

#include <stdio.h>

int main() {

intarr[]={12,33,34,67,7,89,56},i,j,temp,n;

clrscr();

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

for(i=0;i<7;i++){

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

if(arr[j-1]>arr[j]){

temp=arr[j-1];

arr[j-1]=arr[j];

arr[j =temp;

}

}

}

printf("Second Largest is : %d",arr[n-2]);

getch()

return 0;

}

1. Reverse the array elements.

#include <stdio.h>

void arrrev(intarr[],intrevarr[],int n){

inti,j;

j=0;

for(i=n-1;i>=0;i--){

revarr[j]=arr[i];

j++;

}

}

int main() {

int arr[]={12,33,34,67,7,89,56},i,n;

clrscr();

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

intrevarr[n];

arrrev(arr,revarr,n);

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

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

}

getch();

return 0;

}

2. Introduction to string. Perform the following operation on string.

(a) Check if the string is palindrome.

#include<stdio.h>

#include<string.h>

int isPalindrome(char n[],int len){

int j=len-1,i;

int flag=1;

for(i=0;i<len/2;i++){

if(n[i]!=n[j]){

flag=0;

break;

}

j--;

}

return flag;

}

void main()

{

char name[20];

clrscr();

printf("Enter a name:");

gets(name);

printf("String Palindrome status is: %d",isPalindrome(name,strlen(name)));

getch();

}

(b) Convert string into uppercase & lowercase.

#include<stdio.h>

#include<string.h>

void uppLow(char name[]){

int i,ichar;

char upper[50],lower[50];

for(i=0;i<strlen(name);i++){

ichar=(int)name[i];

if(ichar>64 &&ichar<91){

upper[i]=name[i];

lower[i]=(char)ichar+32;

}

else if(ichar>96 &&ichar<123){

lower[i]=name[i];

upper[i]=(char)ichar-32;

}

else{

lower[i]=upper[i]=name[i];

}

}

upper[i]=lower[i]='\0';

printf("Name in Uppercase: %s",upper);

printf("Name in Lowercase: %s",lower);

}

void main()

{

char name[20];

clrscr();

printf("Enter a name:");

gets(name);

uppLow(name);

getch();

}

1. Copy one string to another.

#include<stdio.h>

void strcpy(char des[],char source[]){

int i=0;

while(source[i]!='\0'){

des[i]=source[i];

i++;

}

des[i]='\0';

}

void main()

{

char name[20],copyName[20];

clrscr();

printf("Enter a name:");

gets(name);

strcpy(copyName,name);

printf("Copied name: %s",copyName);

getch();

}

(d) Calculate the length of the string.

#include<stdio.h>

int strlen(char n[]){

int c=0;

while (n[c]!='\0'){

c++;

}

return c;

}

void main()

{

char name[20];

clrscr();

printf("Enter a name:");

gets(name);

printf("length of your name is: %d",strlen(name));

getch();

}

3. Implement a program for stack that performs following operations using arrays.

(a) PUSH (b) POP (c) PEEK (d) DISPLAY

#include <stdio.h>

#define MAX 15

int top=-1;

int stack[MAX];

int isEmpty(){

return top == -1;

}

int isFull(){

return top == MAX-1;

}

int peek(){

if(!isEmpty()){

printf("Peek Element is : %d.\n",stack[top]);

return stack[top];

}

else{

printf("Stack is Empty!!!\n");

return -1;

}

}

void push(){

if(!isFull()){

int data;

printf("Enter data:");

scanf("%d",&data);

stack[++top]=data;

printf("Inserted Successfully.\n");

}

else

printf("Stack is Full(overflow)!!!\n");

}

int pop(){

if(!isEmpty()){

printf("%d Deleted Successfully.\n",stack[top]);

return stack[top--];

}

else{

printf("Stack is Empty(underflow)!!!\n");

return -1;

}

}

void printStack(){

if(!isEmpty()){

int temp=top;

for(;temp>=0;temp--){

printf("%d ",stack[temp]);

}

}

else

printf("Stack is Empty!!!");

printf("\n");

}

void main(){

int data;

int choice;

while(1){

printf("Enter your choice:\n");

printf("1 : Peek Element.\n");

printf("2 : Push Operation.\n");

printf("3 : Pop Operation.\n");

printf("4 : Print Stack.\n");

printf("5 : Exit.\n");

scanf("%d",&choice);

clrscr();

switch(choice){

case 1:

peek();

break;

case 2:

push();

break;

case 3:

pop();

break;

case 4:

printStack();

break;

case 5:

exit(0);

default:

printf("Invalid Choice!!!\n");

getch();

exit(0);

}

}

}

4. Implement a program to convert infix notation to postfix notation using stack.

#include <stdio.h>

#include <string.h>

#define MAX 15

int top=-1;

char stack[MAX];

int isEmpty(){

return top == -1;

}

int isFull(){

return top == MAX-1;

}

void push(char item){

if(!isFull())

stack[++top]=item;

}

char pop(){

if(!isEmpty())

return stack[top--];

}

char peek(){

return stack[top];

}

int precedence(char item){

switch(item){

case '+':

case '-':

return 1;

case '\*':

case '/':

case '%':

return 2;

case '^':

return 3;

}

return -1;

}

void infixToPostfix(char exp[]){

char prefix[100];

int i=0,j=-1;

while(exp[i]!='\0'){

if((exp[i]>='A' && exp[i]<='Z') || (exp[i]>='a' && exp[i]<='z'))

prefix[++j]=exp[i];

else if('('==exp[i])

push(exp[i]);

else if(')'==exp[i]){

while(!isEmpty() && '('!=peek())

prefix[++j]=pop();

pop(); //remove left bracket

}

else{ // if an opertor

while (!isEmpty() && precedence(exp[i]) <= precedence(peek()) ){

if(precedence(peek())==precedence(exp[i])==3)

break;

prefix[++j] = pop();

}

push(exp[i]);

}

i++;

}

while(!isEmpty())

prefix[++j] = pop();

prefix[++j]='\0';

strcpy(exp,prefix);

}

void main(){

int i;

char exp[]="A^B\*C/(D\*E-F)";

clrscr();

printf("Infix Expression:\n");

printf("%s\n",exp);

infixToPostfix(exp);

printf("Postfix Expression:\n");

printf("%s\n",exp);

getch();

}

5. Write a program to implement Queue using arrays that performs following operations.

(a) INSERT (b) DELETE (c) DISPLAY

#include <stdio.h>

#define MAX 15

int front=-1;

int rear=-1;

int queue[MAX];

int isEmpty(){

return front == -1;

}

int isFull(){

return rear==(MAX-1);

}

void enqueue(){

if(!isFull()){

int data;

printf("Enter data:");

scanf("%d",&data);

if(isEmpty()){

front++;

queue[++rear]=data;

}

else{

queue[++rear]=data;

}

printf("Inserted Successfully.\n");

}

else

printf("Queue is Full!!!\n");

}

int dequeue(){

if(!isEmpty()){

printf("%d Deleted Successfully.\n",queue[front]);

if(front==rear){

int temp=front;

front=rear=-1;

return queue[temp];

}

else

return queue[front++];

}

else{

printf("Queue is Empty!!!\n");

return -1;

}

}

void printQueue(){

if(!isEmpty()){

int temp=front;

for(;temp<=rear;temp++){

printf("%d ",queue[temp]);

}

}

else

printf("Queue is Empty!!!");

printf("\n");

}

void main(){

int data;

int choice;

while(1){

printf("Enter your choice:\n");

printf("1 : Enqueue Operation.\n");

printf("2 : Dequeue Operation.\n");

printf("3 : Print Queue.\n");

printf(“4 : Exit.\n”);

scanf("%d",&choice);

clrscr();

switch(choice){

case 1:

enqueue();

break;

case 2:

dequeue();

break;

case 3:

printQueue();

break;

case 4:

exit(0);

default:

printf("Invalid Choice!!!\n");

getch();

exit(0);

}

}

}

6. Write a program to implement Circular Queue using arrays that performs following operations.

(a) INSERT(b) DELETE(c) DISPLAY

#include <stdio.h>

#define MAX 15

int front=-1;

int rear=-1;

int queue[MAX];

int isEmpty(){

if(front==-1){

printf("Queue is Empty!!!\n");

return 1;

}

else

return 0;

}

int isFull(){

if((rear+1)%MAX==front){

printf("Queue is Full!!!\n");

return 1;

}

else

return 0;

}

void enqueue(){

if(!isFull()){

int data;

printf("Enter data:");

scanf("%d",&data);

if(isEmpty()){

front++;

queue[++rear]=data;

}

else{

rear=(rear+1)%MAX;

queue[rear]=data;

}

printf("Inserted Successfully.\n");

}

}

int dequeue(){

if(!isEmpty()){

int temp=front;

printf("%d Deleted Successfully.\n",queue[front]);

if(front==rear){

front=rear=-1;

return queue[temp];

}

else{

front=(front+1)%MAX;

return queue[temp];

}

}

else

return -1;

}

void printQueue(){

if(!isEmpty()){

int temp=front;

while(temp!=rear){

printf("%d ",queue[temp]);

temp=(temp+1)%MAX;

}

printf("%d ",queue[temp]);

}

printf("\n");

}

void main(){

int data;

int choice;

while(1){

printf("Enter your choice:\n");

printf("1 : Enqueue Operation.\n");

printf("2 : Dequeue Operation.\n");

printf("3 : Print Queue.\n");

printf("4 : Exit.\n");

scanf("%d",&choice);

clrscr();

switch(choice){

case 1:

enqueue();

break;

case 2:

dequeue();

break;

case 3:

printQueue();

break;

case 4:

exit(0);

default:

printf("Invalid Choice!!!\n");

getch();

exit(0);

}

}

}

7. Write a menu driven program to implement following operations on the singly linked list.

(a) Insert a node at the front of the linked list.

(b) Insert a node at the end of the linked list.

(c) Insert a node such that the linked list is in ascending order.

(d) Delete the First node of the linked list.

(e) Delete a node before specified position.

(f) Delete a node after specified position.

#include <stdio.h>

#include <stdlib.h>

struct list{

int data;

struct list \*next;

}\*head;

void insInEmpty(int data){

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

head->data=data;

head->next=NULL;

printf("Inserted successfully.\n");

}

void insAtBeg(int data){

if(isEmpty()){

insInEmpty(data);

}

else{

struct list \*temp=(struct list \*)malloc(sizeof(struct list));

temp->data=data;

temp->next=head;

head=temp;

printf("Inserted successfully.\n");

}

}

void insAtEnd(int data){

if(isEmpty()){

insInEmpty(data);

}

else{

struct list \*temp=(struct list \*)malloc(sizeof(struct list));

struct list \*temp1=(struct list \*)malloc(sizeof(struct list));

temp=head;

while(temp->next!=NULL)

temp=temp->next;

temp1->data=data;

temp1->next=NULL;

temp->next=temp1;

printf("Inserted successfully.\n");

}

}

void insAtPos(int pos,int data){

if(pos==1)

insAtBeg(data);

else{

int c=1;

struct list \*tempData=(struct list \*)malloc(sizeof(struct list));

struct list \*temp=(struct list \*)malloc(sizeof(struct list));

struct list \*prev=(struct list \*)malloc(sizeof(struct list));

tempData->data=data;

temp=head;

while(temp!=NULL){

prev=temp;

temp=temp->next;

c++;

if(c==pos)

break;

}

tempData->next=(temp!=NULL?temp:NULL);

prev->next=tempData;

printf("Inserted successfully.\n");

}

}

void delAtBeg(){

if(isEmpty())

printf("List is empty!!!\n");

else{

struct list \*temp=(struct list \*)malloc(sizeof(struct list));

if(head->next==NULL){

printf("%d Deleted successfully\n",head->data);

free(head);

head=NULL;

}

else{

temp=head;

head=head->next;

printf("%d Deleted successfully\n",temp->data);

free(temp);

temp=NULL;

}

}

}

void delAtEnd(){

if(isEmpty())

printf("List is empty!!!\n");

else{

struct list \*temp=(struct list \*)malloc(sizeof(struct list));

struct list \*prev=(struct list \*)malloc(sizeof(struct list));

if(head->next==NULL){

printf("%d Deleted successfully\n",head->data);

free(head);

head=NULL;

}

else{

temp=head;

while(temp->next!=NULL){

prev=temp;

temp=temp->next;

}

prev->next=NULL;

printf("%d Deleted successfully\n",temp->data);

free(temp);

temp=NULL;

}

}

}

void delAtPos(int pos){

if(pos==1)

delAtBeg();

else{

int c=1;

struct list \*temp=(struct list \*)malloc(sizeof(struct list));

struct list \*prev=(struct list \*)malloc(sizeof(struct list));

temp=head;

while(temp!=NULL && c!=pos){

prev=temp;

temp=temp->next;

c++;

}

if(c==pos){

if(temp->next==NULL)

prev->next=NULL;

else

prev->next=temp->next;

printf("%d Deleted Successfully.\n",temp->data);

free(temp);

temp=NULL;

}

}

}

int count(){

int c=1;

struct list \*temp=(struct list \*)malloc(sizeof(struct list));

temp=head;

while(temp->next!=NULL){

temp=temp->next;

c++;

}

return c;

}

int isEmpty(){

if(head==NULL)

return 1;

else

return 0;

}

void main(){

int choice,data,pos;

while(1){

printf("Enter your choice:\n");

printf("1 : insert at beg\n");

printf("2 : insert at end\n");

printf("3 : insert at pos\n");

printf("4 : delete at beg\n");

printf("5 : delete at end\n");

printf("6 : delete at pos\n");

printf("7 : Exit\n");

scanf("%d",&choice);

clrscr();

switch(choice){

case 1:

printf("Enter data:");

scanf("%d",&data);

insAtBeg(data);

break;

case 2:

printf("Enter data:");

scanf("%d",&data);

insAtEnd(data);

break;

case 3:

printf("Enter Position:");

scanf("%d",&pos);

if(isEmpty() && pos>1)

printf("List is Empty and Invalid Positon!!!\n");

else if(pos<1 || pos>count()+1)

printf("Invalid position!!!\n");

else{

printf("Enter data:");

scanf("%d",&data);

insAtPos(pos,data);

}

break;

case 4:

delAtBeg();

break;

case 5:

delAtEnd();

break;

case 6:

if(isEmpty())

printf("List is Empty!!!\n");

else{

printf("Enter Position:");

scanf("%d",&pos);

if(pos<1 || pos>count())

printf("Invalid position!!!\n");

else

delAtPos(pos);

}

break;

case 7:

exit(0);

break;

default:

printf("Invalid Choice!!!\n");

getch();

exit(0);

}

}

}

8. Write a program to implement following operations on the doubly linked list.

(a) Insert a node at the front of the linked list.

(b) Insert a node at the end of the linked list.

(c) Delete the last node of the linked list.

(d) Delete a node before specified position.

#include <stdio.h>

#include <stdlib.h>

struct list{

int data;

struct list \*prev;

struct list \*next;

}\*head;

void insInEmpty(int data){

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

if(head==NULL)

printf("List is Full!!!(overflow)\n");

else{

head->data=data;

head->prev=NULL;

head->next=NULL;

printf("Inserted successfully.\n");

}

}

void insAtBeg(int data){

if(isEmpty()){

insInEmpty(data);

}

else{

struct list \*temp=(struct list \*)malloc(sizeof(struct list));

if(temp==NULL)

printf("List is Full!!!(overflow)\n");

else{

temp->data=data;

temp->prev=NULL;

temp->next=head;

head->prev=temp;

head=temp;

printf("Inserted successfully.\n");

}

}

}

void insAtEnd(int data){

if(isEmpty()){

insInEmpty(data);

}

else{

struct list \*temp=(struct list \*)malloc(sizeof(struct list));

struct list \*tempData=(struct list \*)malloc(sizeof(struct list));

if(temp==NULL || tempData==NULL)

printf("List is Full!!!(overflow)\n");

else{

temp=head;

while(temp->next!=NULL)

temp=temp->next;

tempData->data=data;

tempData->prev=temp;

tempData->next=NULL;

temp->next=tempData;

printf("Inserted successfully.\n");

}

}

}

void delAtBeg(){

if(isEmpty())

printf("List is empty!!!\n");

else{

struct list \*temp=(struct list \*)malloc(sizeof(struct list));

if(head->next==NULL){

printf("%d Deleted successfully\n",head->data);

free(head);

head=NULL;

}

else{

temp=head;

head=head->next;

head->prev=NULL;

printf("%d Deleted successfully\n",temp->data);

free(temp);

temp=NULL;

}

}

}

void delAtEnd(){

if(isEmpty())

printf("List is empty!!!\n");

else{

struct list \*temp=(struct list \*)malloc(sizeof(struct list));

struct list \*prev=(struct list \*)malloc(sizeof(struct list));

if(head->next==NULL){

printf("%d Deleted successfully\n",head->data);

free(head);

head=NULL;

}

else{

temp=head;

while(temp->next!=NULL){

prev=temp;

temp=temp->next;

}

prev->next=NULL;

printf("%d Deleted successfully\n",temp->data);

free(temp);

temp=NULL;

}

}

}

int isEmpty(){

if(head==NULL)

return 1;

else

return 0;

}

void main(){

int choice,data,pos;

while(1){

printf("Enter your choice:\n");

printf("1 : insert at beg\n");

printf("2 : insert at end\n");

printf("3 : delete at beg\n");

printf("4 : delete at end\n");

printf("5 : Exit\n");

scanf("%d",&choice);

clrscr();

switch(choice){

case 1:

printf("Enter data:");

scanf("%d",&data);

insAtBeg(data);

break;

case 2:

printf("Enter data:");

scanf("%d",&data);

insAtEnd(data);

break;

case 3:

delAtBeg();

break;

case 4:

delAtEnd();

break;

case 5:

exit(0);

default:

printf("Invalid Choice!!!\n");

getch();

exit(0);

}

}

}

9. Write a program to create a Binary search tree.

#include <stdio.h>

#include <stdlib.h>

// Node structure for the Binary Search Tree

struct Node {

int data;

struct Node\* left;

struct Node\* right;

};

// Function to create a new node

struct Node\* createNode(int key) {

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

newNode->data = key;

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

return newNode;

}

// Function to insert a new key into BST

struct Node\* insert(struct Node\* root, int key) {

if (root == NULL)

return createNode(key);

if (key < root->data)

root->left = insert(root->left, key);

else if (key > root->data)

root->right = insert(root->right, key);

else // Duplicate keys not allowed

printf(“Duplicate not allowed!!!\n”);

return root;

}

int main() {

struct Node\* root = NULL;

// Inserting keys into the BST

root = insert(root, 50);

insert(root, 30);

insert(root, 20);

insert(root, 40);

insert(root, 70);

insert(root, 60);

insert(root, 80);

return 0;

}

10. Implement recursive and non-recursive tree traversing methods in-order,pre-order and post-order traversal.

#include <stdio.h>

#include <stdlib.h>

// Node structure for the Binary Search Tree

struct Node {

int data;

struct Node\* left;

struct Node\* right;

};

// Function to create a new node

struct Node\* createNode(int key) {

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

newNode->data = key;

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

return newNode;

}

// Function to insert a new key into BST

struct Node\* insert(struct Node\* root, int key) {

if (root == NULL)

return createNode(key);

if (key < root->data)

root->left = insert(root->left, key);

else if (key > root->data)

root->right = insert(root->right, key);

else // Duplicate keys not allowed

printf(“Duplicate not allowed!!!\n”);

return root;

}

// Function to perform in-order traversal (recursive)

void inOrderRecursive(struct Node\* root) {

if (root != NULL) {

inOrderRecursive(root->left);

printf("%d ", root->data);

inOrderRecursive(root->right);

}

}

// Function to perform pre-order traversal (recursive)

void preOrderRecursive(struct Node\* root) {

if (root != NULL) {

printf("%d ", root->data);

preOrderRecursive(root->left);

preOrderRecursive(root->right);

}

}

// Function to perform post-order traversal (recursive)

void postOrderRecursive(struct Node\* root) {

if (root != NULL) {

postOrderRecursive(root->left);

postOrderRecursive(root->right);

printf("%d ", root->data);

}

}

// Function to perform in-order traversal (non-recursive)

void inOrderNonRecursive(struct Node\* root) {

// TODO: Implement non-recursive in-order traversal

}

// Function to perform pre-order traversal (non-recursive)

void preOrderNonRecursive(struct Node\* root) {

// TODO: Implement non-recursive pre-order traversal

}

// Function to perform post-order traversal (non-recursive)

void postOrderNonRecursive(struct Node\* root) {

// TODO: Implement non-recursive post-order traversal

}

int main() {

struct Node\* root = NULL;

// Inserting keys into the BST

root = insert(root, 50);

insert(root, 30);

insert(root, 20);

insert(root, 40);

insert(root, 70);

insert(root, 60);

insert(root, 80);

// Recursive traversals

printf("In-order (recursive): ");

inOrderRecursive(root);

printf("\n");

printf("Pre-order (recursive): ");

preOrderRecursive(root);

printf("\n");

printf("Post-order (recursive): ");

postOrderRecursive(root);

printf("\n");

return 0;

}

11. Implement AVL tree.

#include <stdio.h>

#include <stdlib.h>

// AVL Tree Node structure (extends Node)

struct AVLNode {

int data;

struct AVLNode\* left;

struct AVLNode\* right;

int height; // Height of the node

};

// Function to get the height of a node

int height(struct AVLNode\* node) {

if (node == NULL)

return 0;

return node->height;

}

// Function to get the maximum of two integers

int max(int a, int b) {

return (a > b) ? a : b;

}

// Function to create a new AVL node

struct AVLNode\* createAVLNode(int key) {

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

newNode->data = key;

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

newNode->height = 1; // New node is initially at height 1

return newNode;

}

// Function to rotate the subtree rooted with y to the left

struct AVLNode\* leftRotate(struct AVLNode\* y) {

struct AVLNode\* x = y->right;

struct AVLNode\* T2 = x->left;

// Perform rotation

x->left = y;

y->right = T2;

// Update heights

y->height = max(height(y->left), height(y->right)) + 1;

x->height = max(height(x->left), height(x->right)) + 1;

return x;

}

// Function to rotate the subtree rooted with x to the right

struct AVLNode\* rightRotate(struct AVLNode\* x) {

struct AVLNode\* y = x->left;

struct AVLNode\* T2 = y->right;

// Perform rotation

y->right = x;

x->left = T2;

// Update heights

x->height = max(height(x->left), height(x->right)) + 1;

y->height = max(height(y->left), height(y->right)) + 1;

return y;

}

// Get the balance factor of a node

int getBalance(struct AVLNode\* node) {

if (node == NULL)

return 0;

return height(node->left) - height(node->right);

}

// Function to insert a key into AVL tree

struct AVLNode\* insertAVL(struct AVLNode\* node, int key) {

// Perform normal BST insertion

if (node == NULL)

return createAVLNode(key);

if (key < node->data)

node->left = insertAVL(node->left, key);

else if (key > node->data)

node->right = insertAVL(node->right, key);

else // Duplicate keys not allowed

return node;

// Update height of current node

node->height = 1 + max(height(node->left), height(node->right));

// Get the balance factor to check if this node became unbalanced

int balance = getBalance(node);

// Left Left Case

if (balance > 1 && key < node->left->data)

return rightRotate(node);

// Right Right Case

if (balance < -1 && key > node->right->data)

return leftRotate(node);

// Left Right Case

if (balance > 1 && key > node->left->data) {

node->left = leftRotate(node->left);

return rightRotate(node);

}

// Right Left Case

if (balance < -1 && key < node->right->data) {

node->right = rightRotate(node->right);

return leftRotate(node);

}

return node;

}

// AVL tree in-order traversal (recursive)

void inOrderAVL(struct AVLNode\* root) {

if (root != NULL) {

inOrderAVL(root->left);

printf("%d ", root->data);

inOrderAVL(root->right);

}

}

// Driver program for AVL tree

int main() {

struct AVLNode\* root = NULL;

// Inserting keys into the AVL tree

root = insertAVL(root, 10);

root = insertAVL(root, 20);

root = insertAVL(root, 30);

// In-order traversal of AVL tree

printf("In-order (AVL): ");

inOrderAVL(root);

printf("\n");

return 0;

}

12. Implement heap sort.

#include <stdio.h>

#include <stdlib.h>

// Function to heapify a subtree rooted with node i which is an index in arr[]

void heapify(int arr[], int n, int i) {

int largest = i; // Initialize largest as root

int left = 2 \* i + 1; // Left child

int right = 2 \* i + 2; // Right child

// If left child is larger than root

if (left < n && arr[left] > arr[largest])

largest = left;

// If right child is larger than largest so far

if (right < n && arr[right] > arr[largest])

largest = right;

// If largest is not root

if (largest != i) {

// Swap arr[i] and arr[largest]

int temp = arr[i];

arr[i] = arr[largest];

arr[largest] = temp;

// Recursively heapify the affected sub-tree

heapify(arr, n, largest);

}

}

// Function to perform heap sort

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

// Build heap (rearrange array)

for (int i = n / 2 - 1; i >= 0; i--)

heapify(arr, n, i);

// One by one extract an element from the heap

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

// Move current root to end

int temp = arr[0];

arr[0] = arr[i];

arr[i] = temp;

// Call max heapify on the reduced heap

heapify(arr, i, 0);

}

}

// Function to print an array

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

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

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

printf("\n");

}

// Driver program for Heap Sort

int main() {

int arr[] = {12, 11, 13, 5, 6, 7};

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

printf("Original array: \n");

printArray(arr, n);

// Perform heap sort

heapSort(arr, n);

printf("Sorted array: \n");

printArray(arr, n);

return 0;

}

13. Create an one dimensional array with values 22,4,67,34,12,87,89,2656,9. Perform the following sorting operation on an array to sort the data in ascending order.

(a) Bubble sort

#include <stdio.h>

void bubbleSort(int arr[]){

int i,j,temp;

for(i=0;i<10;i++){

for(j=0;j<9-i;j++){

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

temp=arr[j];

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

arr[j+1]=temp;

}

}

}

}

void main(){

int i,arr[10];

clrscr();

arr={22,4,67,34,12,87,89,2656,9};

bubbleSort(arr);

printf("Sorted array is: ");

for(i=0;i<10;i++){

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

}

getch();

}

(b) Insertion

#include <stdio.h>

void insertionSort(int arr[]){

int i,j,key;

for(i=1;i<9;i++){

key=arr[i];

j=i-1;

while(j>=0 &&arr[j]>key){

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

j--;

}

arr[j+1]=key;

}

}

void main(){

int i,arr[10];

clrscr();

arr={22,4,67,34,12,87,89,2656,9};

insertionSort(arr);

printf("Sorted array is: ");

for(i=0;i<10;i++){

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

}

getch();

}

(c) Quick sort

#include <stdio.h>

int partition(int arr[],int beg,int end){

int pivot = arr[beg];

int i= beg+1;

int j=end;

int temp;

do{

while(arr[i]<=pivot){

i++;

}

while(arr[j]>pivot){

j--;

}

if(i<j){

temp=arr[i];

arr[i]=arr[j];

arr[j]=temp;

}

}while(i<j);

temp=arr[beg];

arr[beg]=arr[j];

arr[j]=temp;

return j;

}

void quickSort(int arr[],int beg,int end){

if(beg<end){

int p = partition(arr,beg,end);

quickSort(arr,beg,p-1);

quickSort(arr,p+1,end);

}

}

void main(){

int i,arr[10],n;

clrscr();

arr={22,4,67,34,12,87,89,2656,9};

quickSort(arr,0,9);

printf("Sorted array is: ");

for(i=0;i<10;i++){

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

}

getch();

}

(d) Merge sort

#include <stdio.h>

void merge(int arr[],int beg,intmid,int end){

int l,r,i,temp[10];

for(l=beg,r=mid+1,i=beg; l<=mid && r<=end;i++){

if(arr[l]<=arr[r])

temp[i]=arr[l++];

else

temp[i]=arr[r++];

}

while(l<=mid)

temp[i++]=arr[l++];

while(r<=end)

temp[i++]=arr[r++];

for(i=beg;i<=end;i++)

arr[i]=temp[i];

}

void mergeSort(int arr[],int beg,int end){

int mid;

if(beg<end){

mid=(beg+end)/2;

mergeSort(arr,beg,mid);

mergeSort(arr,mid+1,end);

merge(arr,beg,mid,end);

}

}

void main(){

int i,arr[10],n;

clrscr();

arr={22,4,67,34,12,87,89,2656,9};

mergeSort(arr,0,9);

printf("Sorted array is: ");

for(i=0;i<10;i++){

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

}

getch();

}

14. Write a program to implement Binary Search.

#include <stdio.h>

int i;

void binarySearch(int arr[],int low,int high){

int mid,temp;

printf("Enter item to search:\n");

scanf("%d",&temp);

while(low<=high){

mid=(low+high)/2;

if(temp==arr[mid]){

printf("found at index %d",mid);

return ;

}

else if(temp<arr[mid])

high=mid-1;

else

low=mid+1;

}

printf("Not found!!!");

}

int main(){

int arr[10];

clrscr();

printf("Enter array item only in sorted manner!!!\n");

for(i=0;i<10;i++){

printf("Enter an item:\n");

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

}

binarySearch(arr,0,9);

getch();

return 0;

}

15. Write a C program to implement DFS traversal using Adjacency Matrix in a Graph.

#include <stdio.h>

#include <stdlib.h>

#define MAX 100

// Function to perform Depth-First Search (DFS) traversal

void dfs(int matrix[MAX][MAX], int visited[MAX], int n, int current) {

int i;

printf("%d ", current);

visited[current] = 1;

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

if (matrix[current][i] == 1 && !visited[i]) {

dfs(matrix, visited, n, i);

}

}

}

// Function to initialize the adjacency matrix

void initializeMatrix(int matrix[MAX][MAX], int n) {

int i,j;

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

for (j = 0; j < n; j++) {

matrix[i][j] = 0;

}

}

}

// Driver program

int main() {

int matrix[MAX][MAX], visited[MAX], n, i, j;

printf("Enter the number of vertices: ");

scanf("%d", &n);

initializeMatrix(matrix, n);

printf("Enter the adjacency matrix:\n");

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

for (j = 0; j < n; j++) {

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

}

}

// Initializing visited array

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

visited[i] = 0;

}

printf("DFS Traversal starting from vertex 0: ");

dfs(matrix, visited, n, 0);

printf("\n");

return 0;

}