# LAB RECORD DSA

1) Write a menu driven program with the following options to construct a binary search tree (BST) recursively and traverse the elements:

1. Insert
2. Pre-orded traversal 3-In order traversal

4-Post order traversal 5-Exit

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

struct node

{

int data;

struct node \*left; struct node \*right;

} \*root = NULL;

struct node \*insert(struct node \*, int); void preorder(struct node \*);

void inorder(struct node \*); void postorder(struct node \*);

main()

{

int ch, x; while (1)

{

printf("\nMenu: \n1: insert\n2: pre-order traversal\n 3: in-order traversal\n 4: post-order traversal\n \n 5: exit\n");

printf("\n Enter your choice"); scanf("%d", &ch);

switch (ch)

{

case (1):

printf("enter the data to insert:"); scanf("%d", &x);

root = insert(root, x); break;

case (2): preorder(root); break;

case (3): inorder(root); break;

case (4): postorder(root); break;

case (5):

exit(0); default:

printf("Invalid option");

}

}

}

struct node \*insert(struct node \*temp, int ele)

{

if (temp == NULL)

{

temp = (struct node \*)malloc(sizeof(struct node)); temp->data = ele;

temp->left = NULL; temp->right = NULL;

}

else

{

if (ele < temp->data)

temp->left = insert(temp->left, ele); else

{

if (ele > temp->data)

temp->right = insert(temp->right, ele);

}

}

return temp;

}

void preorder(struct node \*ptr)

{

if (ptr != NULL)

{

printf("%d\t", ptr->data); preorder(ptr->left); preorder(ptr->right);

}

}

void inorder(struct node \*ptr)

{

if (ptr != NULL)

{

inorder(ptr->left); printf("%d\t", ptr->data); inorder(ptr->right);

}

}

void postorder(struct node \*ptr)

{

if (ptr != NULL)

{

postorder(ptr->left); postorder(ptr->right); printf("%d\t", ptr->data);

}

}

# 2-

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

struct node

{

int data;

struct node \*left; struct node \*right;

} \*root = NULL;

struct node \*insert(struct node \*, int); void inorder(struct node \*);

struct node \*search(struct node \*, int); main()

{

int ch, x, val; while (1)

{

printf("\nMenu: \n1: insert\n2: in-order traversal\n 3: Search\n 4: exit\n"); printf("\n Enter your choice");

scanf("%d", &ch);

switch (ch)

{

case (1):

printf("enter the data to insert:"); scanf("%d", &x);

root = insert(root, x); break;

case (2):

inorder(root); break;

case (3):

printf("Enter element to be searched"); scanf("%d", &val);

root = search(root, val); case (4):

exit(0); break;

default:

printf("Invalid option");

}

}

}

struct node \*insert(struct node \*temp, int ele)

{

if (temp == NULL)

{

temp = (struct node \*)malloc(sizeof(struct node)); temp->data = ele;

temp->left = NULL; temp->right = NULL;

}

else

{

if (ele < temp->data)

temp->left = insert(temp->left, ele); else

{

if (ele > temp->data)

temp->right = insert(temp->right, ele);

}

}

return temp;

}

void inorder(struct node \*p)

{

if (p != NULL)

{

inorder(p->left); printf("%d \t", p->data); inorder(p->right);

}

}

struct node \*search(struct node \*temp, int val)

{

struct node \*p;

p = temp;

if (p != NULL && p->data != val)

{

if (val < p->data)

{

p = p->left;

}

else

{

if (val > p->data) p = p->right;

}

}

if (p == NULL)

{

printf("Element not found");

}

else

{

return p;

}

}

# 3-

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

struct node

{

int data;

struct node \*left; struct node \*right;

} \*root = NULL;

struct node \*insert(struct node \*, int); void inorder(struct node \*);

struct node \*delete (struct node \*temp, int val);

main()

{

int ch, x, val; while (1)

{

printf("\nMenu: \n1: insert\n2: in-order traversal\n 3: Delete\n 4: exit\n"); printf("\n Enter your choice");

scanf("%d", &ch); switch (ch)

{

case (1):

printf("enter the data to insert:"); scanf("%d", &x);

root = insert(root, x); break;

case (2):

inorder(root); break;

case (3):

printf("Enter element to be searched"); scanf("%d", &val);

root = delete (root, val);

case (4):

exit(0); break;

default:

printf("Invalid option");

}

}

}

struct node \*insert(struct node \*temp, int ele)

{

if (temp == NULL)

{

temp = (struct node \*)malloc(sizeof(struct node)); temp->data = ele;

temp->left = NULL; temp->right = NULL;

}

else

{

if (ele < temp->data)

temp->left = insert(temp->left, ele); else

{

if (ele > temp->data)

temp->right = insert(temp->right, ele);

}

}

return temp;

}

void inorder(struct node \*p)

{

if (p != NULL)

{

inorder(p->left); printf("%d \t", p->data);

inorder(p->right);

}

}

struct node \*delete (struct node \*temp, int val)

{

if (temp == NULL) return temp;

if (val < temp->data)

{

temp->left = delete (temp->left, val);

}

else

{

if (val >> temp->data)

{

temp->right = delete (temp->right, val);

}

else

{

if (temp->left == NULL)

{

struct node \*p = temp->right; free(temp);

return p;

}

else if (temp->right == NULL)

{

struct node \*p = temp->left; free(temp);

return p;

}

}

}

}

# 4-

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

struct node

{

int data;

struct node \*left; struct node \*right;

} \*root = NULL;

struct node \*insert(struct node \*, int); void inorder(struct node \*);

struct node \*min(struct node \*temp); struct node \*max(struct node \*temp);

main()

{

int ch, x, val; while (1)

{

printf("\nMenu: \n1: insert\n2: in-order traversal\n 3: Minimum\n \n 4: Maximum\n 5: exit\n"); printf("\n Enter your choice");

scanf("%d", &ch); switch (ch)

{

case (1):

printf("enter the data to insert:"); scanf("%d", &x);

root = insert(root, x); break;

case (2): inorder(root); break;

case (3): min(root); break;

case (4): max(root);

break; case (5):

exit(0); break;

default:

printf("Invalid option");

}

}

}

struct node \*insert(struct node \*temp, int ele)

{

if (temp == NULL)

{

temp = (struct node \*)malloc(sizeof(struct node)); temp->data = ele;

temp->left = NULL; temp->right = NULL;

}

else

{

if (ele < temp->data)

temp->left = insert(temp->left, ele); else

{

if (ele > temp->data)

temp->right = insert(temp->right, ele);

}

}

return temp;

}

void inorder(struct node \*p)

{

if (p != NULL)

{

inorder(p->left); printf("%d \t", p->data); inorder(p->right);

}

}

struct node \*min(struct node \*temp)

{

if (temp == NULL) return NULL;

if (temp->left == NULL)

{

return temp;

}

else

{

return (min(temp->left));

}

}

struct node \*max(struct node \*temp)

{

if (temp == NULL) return NULL;

if (temp->right == NULL)

{

return temp;

}

else

{

return (max(temp->right));

}

}

# 5-

## /\*Implement bubble sort to sort the elements of any user entered array in ascending order. \*/

#include <stdio.h>

void bubblesort(int[], int); main()

{

int a[20], n, i;

printf("Enter the number of elements in the array"); scanf("%d", &n);

printf("Enter the array elements"); for (i = 0; i < n; i++)

{

scanf("%d\t", &a[i]);

}

printf("The unsorted array is ", a[i]);

}

void bubblesort(int a[], int n)

{

int i, j, temp;

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

{

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

{

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

{

temp = a[j]; a[j] = a[j + 1]; a[j + 1] = temp;

}

}

}

printf("Sorted list in ascending order:\n");

for (i = 0; i < n; i++) printf("%d\n", a[i]);

return 0;

}

# 6-

## /\*Implement insertion sort to sort the elements of any user entered array in ascending order. \*/

#include <stdio.h>

void insertionsort(int[], int); main()

{

int a[20], i, n;

printf("Enter the number of elements in the array"); scanf("%d", &n);

printf("Enter the array elements"); for (i = 0; i < n; i++)

{

scanf("%d \t", &a[i]);

}

printf("The unsorted array is %d", a[i]);

}

void insertionsort(int a[], int n)

{

int key, i, j, ;

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

{

key = a[i]; j = i - 1;

while (j >= 0 && a[j] > key)

{

a[j + 1] = a[j]; j = j - 1;

}

a[j + 1] = key;

}

}

# 7-

## /\*Implement selection sort to sort the elements of any user entered array in ascending order. \*/

#include <stdio.h>

void selectionsort(int[], int); main()

{

int a[20], i, n;

printf("Enter the number of elements in the array"); scanf("%d", &n);

printf("Enter the array elements"); for (i = 0; i < n; i++)

{

scanf("%d \t", &a[i]);

}

printf("The unsorted array is %d", a[i]);

}

void selectionsort(int a[], int n)

{

int i, j, min, temp;

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

{

min = i;

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

{

if (a[min] > a[j])

;

min = j;

}

if (min != i)

{

temp = a[i]; a[i] = a[min]; a[min] = temp;

}

}

printf("Sorted list in ascending order:\n");

for (i = 0; i < n; i++) printf("%d\n", a[i]);

return 0;

}

# 8-

## /\*Implement merge sort to sort the elements of any user entered array in ascending order\*/

#include<stdio.h>

void mergesort(int a[],int i,int j);

void merge(int a[],int i1,int j1,int i2,int j2); int main()

{

int a[30],n,i;

printf("Enter no of elements:"); scanf("%d",&n);

printf("Enter array elements:");

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

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

mergesort(a,0,n-1);

printf("\nSorted array is :"); for(i=0;i<n;i++)

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

return 0;

}

void mergesort(int a[],int i,int j)

{

int mid;

if(i<j)

{

mid=(i+j)/2; mergesort(a,i,mid); mergesort(a,mid+1,j); merge(a,i,mid,mid+1,j);

}

}

void merge(int a[],int i1,int j1,int i2,int j2)

{

int temp[50]; int i,j,k;

i=i1; j=i2; k=0;

while(i<=j1 && j<=j2)

{

if(a[i]<a[j])

temp[k++]=a[i++];

else

temp[k++]=a[j++];

}

while(i<=j1)

temp[k++]=a[i++];

while(j<=j2)

temp[k++]=a[j++]; for(i=i1,j=0;i<=j2;i++,j++)

a[i]=temp[j];

}

# 9-

## /\*Implement quick sort to sort the elements of any user entered array in ascending order\*/

#include <stdio.h>

void quicksort(int a[], int p, int r); int partition(int a[], int p, int r);

int main()

{

int a[30], n, i;

printf("Enter no of elements:"); scanf("%d", &n);

printf("Enter array elements:");

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

quicksort(a, 0, n - 1);

printf("\nSorted array is :"); for (i = 0; i < n; i++)

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

return 0;

}

void quicksort(int a[], int p, int r)

{

int q;

if (p < r)

{

q = partition(a, p, r); quicksort(a, p, q); quicksort(a, q + 1, r);

}

}

int partition(int a[], int p, int r)

{

int pivot, i, j, temp; pivot = a[p];

i = p - 1; j = r + 1; while (1)

{

do

{

j = j - 1;

} while (a[j] > pivot); do

{

i = i + 1;

} while (a[i] < pivot); if (i < j)

{

temp = a[i]; a[i] = a[j]; a[j] = temp;

}

else

return (j);

}

}