DMI COLLEGE OF ENGINEERING

(AN AUTONOMOUS INSTITUTION)

(Affiliated to Anna University, Chennai-600 025)

Palanchur, Chennai–600123

CS1205- DATA STRUCTURES LAB MANUAL

DEPARTMENT OF INFORMATION TECHNOLOGY

II YEAR / III SEMESTER

2025-2026

LIST OF EXERCISES

CS1205 **DATA STRUCTURES LABORATORY** **L T P C**
0 0 3 1.5

1. Implementation of Singly Linked List and Doubly Linked List
2. Implementation of Polynomial ADT using Linked list
3. Array implementation of Stack and Queue ADTs
4. Linked list implementation of Stack and Queue ADTs
5. Convert a given Infix expression into its Postfix using Stack
6. Evaluate a given Postfix Expressions using Stack
7. Implementation of Binary Search Trees
8. Implementation of AVL Trees
9. Implementation of Heap Trees
10. Implementation of Shortest Path algorithms (Dijkstra's and Floyd's Algorithm)
11. Implementation of Minimum Spanning Tree algorithms (Prim's and Kruskal's Algorithm)
12. Implementation of Linear Search and Binary Search
13. Implementation of Insertion Sort and Selection Sort
14. Implementation of Open Addressing (Linear Probing and Quadratic Probing)

TOTAL: 45 PERIODS

Lab Requirements: for a batch of 24 students

Operating Systems: Linux / Windows

Tools: Dev C++ / Eclipse CDT / Code Blocks / CodeLite / equivalent open source IDE

EX.NO. 1(A) Array implementation of LIST

Aim: To Write a program for Array based implementation of LIST ADT with Create, Insert, Search , Delete & Traverse operations

Algorithm:

1. Start the program and declare an array and variables.
2. Ask the user how many elements they want and read them into the array.
3. Ask the user for a new element and the position to insert it.
4. Check if the position is valid (between 0 and current size).
5. If valid, shift elements to the right from that position.
6. Insert the new element at the given position and increase the size.
7. Ask the user for an element to search and delete.
8. Search the array for that element and store its position if found.
9. If found, shift elements left to remove it and reduce the size.
10. Print the final array after all operations.

Program:

```
// 1 (A ) Array Implementation of List (Content Beyond Syllabus)
#include <stdio.h>

void main() {
   int a[100], n, i, p, x;

   // Create array
   printf("Enter number of elements: ");
   scanf("%d", &n);

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

   // Insert element
   printf("\nEnter element to insert: ");
   scanf("%d", &x);
   printf("Enter position (0 to %d): ", n);
   scanf("%d", &p);

   if (p >= 0 && p <= n) {
      for (i = n; i > p; i--)
         a[i] = a[i - 1];
      a[p] = x;
      n++;
      printf("Element %d inserted at position %d.\n", x, p);
   } else {
      printf("Invalid position. Insertion skipped.\n");
   }

   // Search a element  to delete
```

```
    printf("\nEnter element to search and delete: ");
    scanf("%d", &x);

    p = -1;
    for (i = 0; i < n; i++) {
        if (a[i] == x) {
            p = i;
            break;
        }
    }

//Delete a element
    if (p != -1) {
        printf("Element %d found at position %d.\n", x, p);
        for (i = p; i < n - 1; i++)
            a[i] = a[i + 1];
        n--;
        printf("Element %d deleted.\n", x);
    } else {
        printf("Element %d not found. No deletion performed.\n", x);
    }

    // Display array
    printf("\nFinal array:\n");
    for (i = 0; i < n; i++)
        printf("%d ", a[i]);
    printf("\n");

    }
```

Output:

Result

Thus the program for Array based implementation of LIST ADT with Create, Insert, Search , Delete & Traverse operations has been written,Executed & Output was verified.

// EX.NO.1 (A) ARRAY IMPLEMENTATION OF STACK

```
#include <stdio.h>
#include <conio.h>
#define size 5
int s[size], top = -1, i, x;
void main() {
   int ch;
   clrscr();
   printf("Array Implementation of Stack ");
   while (1) {
      printf("\n 1. Push 2. Pop 3. Display 4. Exit");
      printf("\n Enter your choice: ");
      scanf("%d", &ch);
      switch (ch) {
         case 1:  printf("Enter value to push: ");
                  scanf("%d", &x);
                  if (top == size - 1) {
                  printf("Stack is FULL\n");
                  } else {
                  top = top + 1;
                  s[top] = x;
                  printf("%d pushed onto stack\n", x);
                  }
                  break;
         case 2:if (top == -1) {
                  printf("Stack is empty\n");
                  } else {
                  x = s[top];
                 top = top - 1;
                 printf("%d popped from stack\n", x);
                 }
                 break;
         case 3: if (top == -1) {
                   printf("Stack is empty\n");
                  } else {
                  printf("Stack elements are:\n");
                 for (i = top; i >= 0; i--) {
                  printf("%d ", s[i]);
                 }
                 printf("\n");
                 }
                 break;
         case 4: exit(0);
                 default:
                 printf("Invalid choice\n");
      }
   }}
```

//EX.NO 1 (B)ARRAY IMPLEMENTATION OF QUEUE

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#define size 5
void main() {
   int q[size], front = 0, rear = -1, ch, i, x;
   clrscr();
   printf("\nArray Implementation of Queue\n");
   while (1) {
      printf("\n1. Enqueue 2. Dequeue 3. Display 4. Exit: ");
      printf("\nEnter your choice: ");
      scanf("%d", &ch);
      switch (ch) {
         case 1:              if (rear == size - 1) {
               printf("Queue is full\n");
            } else {
               printf("Enter element to enqueue: ");
               scanf("%d", &x);
               rear = rear + 1;
               q[rear] = x;
            }
            break;
         case 2:              if (front > rear) {
               printf("Queue is empty\n");
            } else {
               x = q[front];
               front = front + 1;
               printf("Dequeued element: %d\n", x);
            }
            break;
         case 3:              if (front > rear) {
               printf("Queue is empty\n");
            } else {
               printf("Queue elements: ");
               for (i = front; i <= rear; i++) {
                  printf("%d ", q[i]);
               }
               printf("\n");
            }
            break;

         case 4:
            exit(0);

         default:
            printf("Invalid choice\n");
      }}}
```

//EX.NO 1 (C) ARRAY IMPLEMENTATION OF CIRCULAR QUEUE

```
#include <stdio.h>
#include<conio.h>
#define SIZE 5  // Define the maximum size of the q
void main() {
   int q[SIZE];
   int f = -1, r = -1;
   int choice, element,i;
     clrscr();
     while (1) {
         printf("\n1. Enq 2. Deq 3. Display 4. Exit\n");
         printf("Enter your choice: ");
         scanf("%d", &choice);
         switch (choice) {
            case 1: if ((f == (r + 1) % SIZE))  // Enq operation
                      printf("Queue is Full\n");
                    else {
                      if(f==-1)
                      f=0;
                      printf("Enter the element to enq: ");
                      scanf("%d", &element);
                      r = (r + 1) % SIZE;  // Circular increment
                      q[r] = element;
                      printf("Inserted %d\n", element);
                            }
                    break;
            case 2:  if (f == -1)// Deq operation
                      printf("Queue is Empty\n");
                    else {
                      printf("Deleted %d\n", q[f]);
                      if (f == r)   // Queue has only one element
                      f = r = -1;
                      else
                      f = (f + 1) % SIZE;  // Circular increment
                      }
                    break;
            case 3: if (f == -1) // Display q
                      printf("Queue is Empty\n");
                    else {
                      printf("Queue elements: ");
                     for( i = f;i != r;i =(i+1)% SIZE)
                            printf("%d ", q[i]);
                              printf("%d\n", q[r]);
                            }
                    break;
            case 4:exit(0);
            default: printf("Invalid choice\n");
         }
   }
}
```

//EX.NO.2.(A) IMPLEMENTATION OF SINGLY LINKED LIST INSERTION & DELETION

```
#include <stdio.h>
#include <stdlib.h>
typedef struct node {
   int data;
   struct node* next;
} node;
node* head = NULL, *pre, *tmp;
int n, x, i;
// Function to create the linked list
void insert(int x) {
 {
          tmp = (node*)malloc(sizeof(node));
          tmp->data= x;
          tmp->next = NULL;
          if (head == NULL) {
             head = tmp;
             pre = head; // first node
          } else {
             pre->next = tmp;
             pre = tmp; // other nodes
          }
   }
}
// Function to display the list
void display() {
   for (tmp = head; tmp != NULL; tmp = tmp->next)
          printf("%d ", tmp->data);
   if (head == NULL)
          printf("\nList is empty");
}

// Function to delete a node with a given integer value
void deleten(int x) {
   tmp = head;
   if (tmp != NULL && tmp->data == x)   //delete first node
   head = head->next;
   //delete other node
   for (pre = tmp;tmp != NULL && tmp->data != x; pre = tmp, tmp = tmp->next);
   pre->next = tmp->next;
   free(tmp);
}
```

```
    void main() {
    clrscr();
    // Create the list
    printf("\nEnter number of elements: ");
    scanf("%d", &n);
    for (i = 0; i < n; ++i)
    {
    printf("\nEnter element %d: ", i + 1);
    scanf("%d",&x);
    insert(x);
    }
    // Display the list
    printf("\nList after creation: ");
    display();
    // Insert at the end
    printf("\nEnter element to be inserted at the end: ");
    scanf("%d", &x);
    insert(x);
    printf("\nList after inserting at the end: ");
    display();
    //Delete a node with a given value
    printf("\nEnter element to be deleted: ");
    scanf("%d", &x);
    deleten(x);
    if (tmp == NULL)
    printf("\nNode with value %d not found", x);
    printf("\nList after deleting the element: ");
    display();
    getch();
}
```

//EX.NO.2(B) SINGLY LINKED LIST CREATE, SEARCH & DISPLAY REVERSE

```
#include <stdio.h>
#include <stdlib.h>
typedef struct node {
   int data;
   struct node* next;
} node;
node* head = NULL, *pre, *tmp ,*res;
int n, x, i;

// Function to create the linked list
void Create(int x) {
   tmp = (node*)malloc(sizeof(node));
   tmp->data = x;
   tmp->next = NULL;
   if (head == NULL) {
                        head = tmp;
                        pre = head; // first node
   } else {
                        pre->next = tmp;
                        pre = tmp; // other nodes
   }
}
// Function to display the list in reverse order
void reverse(node* pre) {
   if (pre == NULL)
                        return;
   reverse(pre->next);
   printf("%d ", pre->data);
}

// Function to search for a node
node* search(int x) {
   for (pre = head; pre != NULL; pre = pre->next)
                        if (pre->data == x)
                            return pre; // number of the node
   return NULL;
}
```

```
void main() {
    // Create the list
    clrscr();
    printf("Enter number of elements: ");
    scanf("%d", &n);
    for (i = 0; i < n; ++i) {
    printf("Enter element %d: ", i + 1);
    scanf("%d", &x);
   Create(x);
    }
     // Search for an element
    printf("\nEnter an element to search: ");
    scanf("%d", &x);
    res = search(x);
    if (res)
    printf("%d found at address %p\n", x, res);
    else
     printf("%d not found in the list\n", x);
     // Display the list in reverse order
    printf("\nList in reverse order: ");
    reverse(head);
    getch();
}
```

//EX.NO.3 (A) LINKED LIST IMPLEMENTATION OF STACK

```
#include <stdio.h>
#include<conio.h>
#include <stdlib.h>
// Define the structure for a singly linked list node
typedef struct node {
   int data;
   struct node* next;
} node;
struct node *top = NULL, *tmp,*pre;
void main() {
   int ch;
   clrscr();
   while (1) {
      printf("\n 1. Push 2. Pop 3. Display 4. Exit");
      printf("\n Enter your choice: ");
      scanf("%d", &ch);
      switch (ch) {
         case 1://push operation
               tmp = (node*)malloc(sizeof(node));
               printf("Enter element to push: ");
               scanf("%d", &tmp->data);
               tmp->next = top;
               top = tmp;
               break;
         case 2: //Pop operation
               if (top == NULL)
                  printf("Stack is empty\n");
                 else {
                  pre = top;
                   printf("Popped element: %d\n",top->data);
                  top = top->next;
                  free(pre);
                 }
               break;
         case 3://display operation
               if (top == NULL)
                  printf("Stack is empty\n");
                  else
                  for(pre=top;pre!= NULL;pre = pre->next)
                  printf("%d ", pre->data);
                 break;
         case 4:exit(0);
         default:printf("Invalid choice\n");
      }
   }
}
```

//EX.NO. 3 (B) LINKED LIST IMPLEMENTATION OF LINEAR QUEUE

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
typedef struct node {
   int data;
   struct node* next;
} node;
struct node* front = NULL,*rear = NULL,*pre,*tmp;
void main() {
   int ch, x;
   clrscr();
   while (1) {
        printf("\n1. Enqueue 2. Dequeue 3. Display 4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &ch);
        switch (ch) {
           case 1:tmp = (node*)malloc(sizeof(node));   // Enqueue operation
                  printf("Enter element to enqueue: ");
                  scanf("%d", &tmp->data);
                  tmp->next = NULL;
                  if (front == NULL)
                     front = rear = tmp;
                   else {
                     rear->next = tmp;
                     rear = tmp;
                  }
                  break;
           case 2: if (front == NULL)  // Dequeue operation
                     printf("Queue is Empty\n");
                    else {
                     pre = front;
                     printf("Element dequeued: %d\n", front->data);
                     front = front->next;
                     free(pre);
                  }
                  break;
            case 3: if (front == NULL)   // Display operation
                     printf("Queue is empty\n");
                     else
                     for (pre = front; pre != NULL; pre = pre->next)
                     printf("%d ", pre->data);
                  break;
           case 4:exit(0);
           default:printf("Invalid choice\n");
        }
    }
        }
```

//EX.NO. 3 (C) DOUBLE ENDED QUEUE/DEQUE USING LINKED LIST

```
#include <stdio.h>
#include <stdlib.h>
typedef struct node {
   int data;
   struct node* next;
} node;
node* front = NULL,* rear = NULL,  *tmp;
void main() {
   int ch;
   clrscr();
     while (1) {
        printf("\n1.Insert@front 2.Insert@rear 3.Delete@front 4.Delete@rear 5.Display 6.Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &ch);
        switch (ch) {
           case 1:// Insert @ Front
                 tmp = (node*)malloc(sizeof(node));
                 printf("Enter element to be inserted at the front: ");
                 scanf("%d", &tmp->data);
                 tmp->next = front;
                 front = tmp;
                 if (rear == NULL)
                 rear = front;
                 break;
           case 2: //Insert @ rear
                 tmp = (node*)malloc(sizeof(node));
                 printf("Enter element to be inserted at the rear: ");
                 scanf("%d", &tmp->data);
                 tmp->next = NULL;
                if (rear == NULL) {
                front = rear = tmp;
                 } else {
                 rear->next = tmp;
                 rear = tmp;
                 }
                 break;
           case 3: //Delete @ Front
                 if (front == NULL)
                    printf("\nDeque is empty\n");
                 else {
                    tmp = front;
                    front = front->next;
                    free(tmp);
                 }
                 break;
```

```
        case 4: //Delete @ rear
                  if (front == NULL)
                    printf("\nDeque is empty\n");
                    else {
                    for (tmp = front; tmp->next != rear; tmp = tmp->next);
                    free(rear);
                    rear = tmp;
                    rear->next = NULL;
                  }
                  break;
             case 5:   if (front == NULL) //Display
                       printf("Deque is empty.\n");
                       else
                       {
                       printf("Deque elements: ");
                       for (tmp = front; tmp != NULL; tmp = tmp->next)
                       printf("%d ", tmp->data);
                       }
                       break;
              case 6:exit(0);
             default: printf("Invalid choice\n");
         }
    }
}
```

//EX.NO.4 IMPLEMENTATION OF POLYNOMIAL MANIPULATION USING LINKED LIST

```
#include <stdio.h>
#include <stdlib.h>
#include<conio.h>
// Define the node structure
typedef struct node {
    int co, po; // Coefficient & Power (exponent)
     struct node* next; // Pointer to the next node
} node;
 struct node *poly =NULL,*poly1, *poly2,*tmp,*cur;

// Function to insert a term into the result polynomial
void insert(node** poly, int co, int po) {
    tmp = (node*)malloc(sizeof(node));
    tmp->co = co;
    tmp->po = po;
    tmp->next = NULL;
    if (*poly == NULL) {
            *poly = tmp;
    } else {
       for(cur = *poly;cur->next != NULL;cur = cur->next);
             cur->next= tmp;
             }
}

void main() {
    int i,n1,n2,co,po;
    // Input for poly1 (user-defined terms)
    clrscr();
    printf("Enter the number of terms for polynomial 1: ");
    scanf("%d", &n1);
    for (i = 0; i < n1; ++i) {
            printf("Enter coefficient and power for term %d: ", i + 1);
            scanf("%d %d", &co, &po);
            insert(&poly1, co, po);
    }
    // Input for poly2 (user-defined terms)
    printf("Enter the number of terms for polynomial 2: ");
    scanf("%d", &n2);
    for (i = 0; i < n2; ++i) {
            printf("Enter coefficient and power for term %d: ", i + 1);
            scanf("%d %d", &co, &po);
            insert(&poly2, co, po);
    }
```

```
// Perform polynomial addition
   while (poly1 != NULL && poly2 != NULL) {
         if (poly1->po == poly2->po) {
            insert(&poly, poly1->co + poly2->co, poly1->po);
            poly1 = poly1->next;
            poly2 = poly2->next;
         } else if (poly1->po > poly2->po) {
            insert(&poly, poly1->co, poly1->po);
            poly1 = poly1->next;
         } else {
            insert(&poly, poly2->co, poly2->po);
            poly2 = poly2->next;
         }
   }
      // Print the result
   printf("Resultant polynomial: ");
    for(cur = poly;cur != NULL;cur=cur->next)
     {
         printf("(%dx^%d)", cur->co, cur->po);
         if (cur->next != NULL) {
            printf(" + ");
         }
            }
  getch();
}
```

//EX.NO.5 (A) IMPLEMENTATION OF EVALUATING POSTFIX EXPRESSIONS

```
#include <stdio.h>
#include <ctype.h>
void main() {
   char exp[100];
   int i = 0, n1, n2, n3,top=-1,stack[100];
   clrscr();
   printf("Enter the postfix expression: ");
   scanf("%s", exp);
     while (exp[i] != '\0') {
         if (isdigit(exp[i])) {
            stack[++top] = exp[i] - '0';
         } else {
            n1 = stack[top--];
            n2 = stack[top--];
            switch (exp[i]) {
                  case '+': n3 = n2 + n1; break;
                  case '-': n3 = n2 - n1; break;
                  case '*': n3 = n2 * n1; break;
                  case '/': n3 = n2 / n1; break;
            }
            stack[++top] = n3;
         }
         i++;
   }
   printf("The result of the expression is: %d\n", stack[top]);
  getch();
}
```

//EX.NO.5(B) INFIX TO POST FIX EXPRESSION

```
#include <stdio.h>
#include <ctype.h>
void main() {
int top = -1,i=0;
char exp[100],x, stack[100];
clrscr();
printf("Enter the expression: ");
scanf("%s", exp);
   while (exp[i] != '\0') {
         if (isalnum(exp[i])) {
            printf("%c ", exp[i]);
         } else if (exp[i] == '(') {
            stack[++top] = exp[i];
         } else if (exp[i] == ')') {
            while (top != -1 && (x = stack[top--]) != '(') {
                  printf("%c ", x);
            }
         } else {
            while (top != -1 && ((stack[top] == '*' || stack[top] == '/') ||
                     ((stack[top] == '+' || stack[top] == '-') && (exp[i] == '+' || exp[i] == '-')))) {
                  printf("%c ", stack[top--]);
            }
            stack[++top] = exp[i];
         }
         i++;
   }
   while (top != -1) {
         printf("%c ", stack[top--]);
   }
getch();
}
```

//EX.NO.6 IMPLEMENTATION OF BINARY SEARCH TREES

```
#include <stdio.h>
#include <stdlib.h>
typedef struct tnode {
   int data;
   struct tnode* left, *right;
   } tnode;
tnode *f, *T=NULL;
tnode* insert(tnode* T, int e) {
   if (T == NULL) {
         T = (tnode*)malloc(sizeof(tnode));
         T->data = e;
         T->left = T->right = NULL;
         return T;
   }
   if (e < T->data)
   T->left = insert(T->left, e);
   else
   T->right = insert(T->right, e);
   return T;
}
tnode* search(tnode* T, int e) {
   if (T == NULL || T->data == e)
   return T;
   if (e < T->data)
   return search(T->left, e);
   return search(T->right, e);
}
void preorder(tnode* T) {
   if (T != NULL) {
         printf("%d ", T->data);
         preorder(T->left);
         preorder(T->right);
   }
}
void main() {
   int n,i, e;
     clrscr();
     printf("Enter number of elements: ");
   scanf("%d", &n);
   printf("Enter the elements: ");
   for (i = 0; i < n; i++) {
         scanf("%d", &e);
         T = insert(T, e);
   }
```

```
  printf("Enter the element to search: ");
  scanf("%d", &e);
  f = search(T, e);
  if (f != NULL) printf("Element %d found in the tree.\n", e);
  else printf("Element %d not found in the tree.\n", e);
  printf("Preorder traversal: ");
  preorder(T);
  printf("\n");
  getch();
}
```

//EX.NO.7 AVL TREE ROTATIONS

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node {
   int data, height;
   struct Node *left, *right;
} Node;
int b,n,data,i;
Node *x,*y,*z ,*root =NULL;

Node* RRrotation(Node* y) {
x = y->right;
y->right = x->left;
x->left = y;
y->height = 1 + (y->left ? y->left->height : 0) > (y->right ? y->right->height : 0) ?
                (y->left ? y->left->height : 0) :  (y->right ? y->right->height : 0);
x->height = 1 + (x->left ? x->left->height : 0) > (x->right ? x->right->height : 0) ?
               (x->left ? x->left->height : 0) : (x->right ? x->right->height : 0);
return x;
}

Node* LLrotation(Node* x) {
Node* y = x->left;
x->left = y->right;
y->right = x;
x->height = 1 + (x->left ? x->left->height : 0) > (x->right ? x->right->height : 0) ?
                 (x->left ? x->left->height : 0) : (x->right ? x->right->height : 0);
y->height = 1 + (y->left ? y->left->height : 0) > (y->right ? y->right->height : 0) ?
               (y->left ? y->left->height : 0) : (y->right ? y->right->height : 0);
return y;
}

Node* RLrotation(Node* z) {
z->right = LLrotation(z->right);
return RRrotation(z);
}
```

```
Node* LRrotation(Node* z) {
z->left = RRrotation(z->left);
return LLrotation(z);
}
Node* insert(Node* node, int data) {
if (!node) {
node = (Node*)malloc(sizeof(Node));
node->data = data;
node->left = node->right = NULL;
node->height = 1;
return node;
}
if (data < node->data) {
node->left = insert(node->left, data);
} else if (data > node->data) {
node->right = insert(node->right, data);
 } else {
return node;
   }
node->height = 1 + (node->left ? node->left->height : 0) > (node->right ? node->right->height : 0) ?
                    (node->left ? node->left->height : 0) : (node->right ? node->right->height : 0);
              b =  (node->left ? node->left->height : 0) - (node->right ? node->right->height : 0);
if (b > 1 && data < node->left->data) return LLrotation(node);
if (b < -1 && data > node->right->data) return RRrotation(node);
if (b > 1 && data > node->left->data) return RLrotation(node);
if (b < -1 && data < node->right->data) return LRrotation(node);
return node;
}

void inorder(Node* root) {
   if (root) {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
   }
}

void main() {
   clrscr();
   printf("\nEnter number of nodes: ");
   scanf("%d", &n);
   for (i = 0; i < n; i++) {
        printf("\nEnter key for node %d: ", i + 1);
        scanf("%d", &data);
        root = insert(root, data);
   }
   printf("\nIn-order traversal of the AVL tree is: ");
   inorder(root);
   getch(); }
```

// EX.NO.8 IMPLEMENTATION OF HEAP TREES

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

// Function to find the maximum element in a max-heap (root element)
findMax(int heap[], int size) {
   if (size > 0)
         return heap[0];
   return INT_MIN; // Return minimum integer value if heap is empty
}

// Function to find the minimum element in a max-heap
findMin(int heap[], int size) {
  int min,i;
   if (size <= 0)
         return INT_MAX; // Return maximum integer value if heap is empty
          min = heap[(size - 1) / 2 + 1];
   for (i = (size - 1) / 2 + 1; i < size; i++) {
         if (heap[i] < min)
            min = heap[i];
   }
   return min;
}

void main() {
   int n,heap[10],i;
   clrscr();
   printf("Enter the number of elements in the heap: ");
   scanf("%d", &n);
   printf("Enter the elements of the heap:\n");
   for (i = 0; i < n; i++) {
         printf("Element %d: ", i + 1);
         scanf("%d", &heap[i]);
   }
   printf("Maximum element in the max-heap: %d\n", findMax(heap, n));
   printf("Minimum element in the max-heap: %d\n",findMin(heap, n));
  getch();
}
```

// EX.NO.10 (A) Implementation of Shortest Path algorithms :Dijkstra's Algorithm

```
#include<stdio.h>
int cost[20][20], dis[20], visit[20];
void main() {
   int n, s, i, j, m, next, min;
   clrscr();
   printf("Enter the no of vertices: ");
   scanf("%d",&n);
   printf("Enter the cost matrix:\n");
   for(i=1; i<=n; i++) {
         for(j=1; j<=n; j++) {
            scanf("%d",&cost[i][j]);
            if(cost[i][j]==0)
                  cost[i][j]=999;
         }
   }
   printf("Enter the source vertex: ");
   scanf("%d",&s);

   for(i=1; i<=n; i++) {
         visit[i]=0;
         dis[i]=cost[s][i];
   }
   visit[s]=1;
   dis[s]=0;

   for(i=2; i<=n; i++) {
         min=999;
         for(j=1; j<=n; j++) {
            if(dis[j]<min && visit[j]==0) {
                  min=dis[j];
                  next=j;
            }
         }
         visit[next]=1;
         for(j=1; j<=n; j++) {
            if(dis[next]+cost[next][j] < dis[j]) {
                  dis[j]=dis[next]+cost[next][j];
            }
         }
   }
   for(i=1; i<=n; i++) {
         printf("Vertex %d to %d: Distance: %d\n", s, i, dis[i]);
   }
getch();
}
```

//EX.NO.11 (A) Implementation Of Minimum Spanning Tree Algorithms: Prim's Algorithm

```
#include<stdio.h>
int cost[20][20], visit[20];
void main() {
   int n,k, i, j, u, v, min, mincost = 0;
   clrscr();
   printf("Enter the no of vertices: ");
   scanf("%d",&n);
   printf("Enter the cost matrix:\n");
   for(i=1; i<=n; i++) {
         for(j=1; j<=n; j++) {
            scanf("%d",&cost[i][j]);
            if(cost[i][j]==0)
                  cost[i][j]=999;
         }
   }
   for(i=1; i<=n; i++) visit[i]=0;
   visit[1] = 1;
   for(k=1; k<n; k++) {
         min = 999;
         for(i=1; i<=n; i++) {
            if(visit[i]) {
                  for(j=1; j<=n; j++) {
                     if(!visit[j] && cost[i][j] < min) {
                           min = cost[i][j];
                           u = i;
                           v = j;
                     }
                  }
            }
         }
         visit[v] = 1;
         mincost += min;
         printf("Edge (%d, %d) = %d\n", u, v, min);
   }
   printf("Minimum cost = %d\n", mincost);
getch();
}
```

//EX.NO. 12 (A) IMPLEMENTATION OF LINEAR SEARCH

```
#include <stdio.h>
void main() {
   int a[10], i, n, x;
   clrscr();
   printf("Enter the number of elements: ");
   scanf("%d", &n);
   printf("Enter %d elements:\n", n);
   for(i = 0; i < n; i++)
         scanf("%d", &a[i]);
   printf("Enter the element to be searched: ");
   scanf("%d", &x);
    for(i = 0; i < n; i++)
         if(a[i] == x) {
            printf("%d is found at location %d\n", x, i);
            getch();
            return;
         }
    printf("%d is not found in the array\n", x);
getch();
}
```

//12 (B) IMPLEMENTATION OF BINARY SEARCH

```
#include <stdio.h>
void main() {
   int a[10],i, n, x, low, mid, high;
   clrscr();
   printf("Enter the number of elements: ");
   scanf("%d", &n);
   printf("Enter %d elements:\n", n);
   for (i = 0; i < n; i++)
   scanf("%d", &a[i]);
   printf("Enter the element to be searched: ");
   scanf("%d", &x);
   low = 0;
   high = n - 1;
   while (low <= high) {
         mid = (low + high) / 2;
         if (a[mid] == x) {
            printf("%d is found at location %d\n", x, mid);
            getch();
            return ;
         }
         else if (a[mid] < x) low = mid + 1;
         else high = mid - 1;
   }
   printf("%d is not found in the array\n", x);
   getch();
}
```

//EX.NO.11(C) IMPLEMENTATION OF PATTERN SEARCH

```
#include <stdio.h>
#include <string.h>
void main() {
   char T[25], P[10];
   int i, j, n, m;
   clrscr();
   printf("Enter the Text: ");
   scanf("%s", T);
   printf("Enter the Pattern: ");
   scanf("%s", P);
   n = strlen(T);
   m = strlen(P);
   for (i = 0; i <= n - m; i++) {
         for (j = 0; j < m && P[j] == T[i + j]; j++);
         if (j == m) {
            printf("Pattern is available in Text at %d\n", i);
            getch();
            return ;
         }
   }
   printf("Pattern is not available in Text\n");
   getch();
}
```

//EX.NO.13 (A) IMPLEMENTATION OF INSERTION SORT

```
#include <stdio.h>
void main() {
    int arr[10], n, i, j, key;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    printf("Enter %d elements: ", n);
    for (i = 0; i < n; i++) scanf("%d", &arr[i]);
     for (i = 1; i < n; i++) {
            key = arr[i];
            j = i - 1;
            while (j >= 0 && arr[j] > key) {
                arr[j + 1] = arr[j];
                j--;
            }
            arr[j + 1] = key;
    }
    printf("Sorted array: ");
    for (i = 0; i < n; i++) printf("%d ", arr[i]);
    printf("\n");
    getch();
}
```

// 13 (B) IMPLEMENTATION OF SELECTION SORT

```
#include <stdio.h>
void main()
{
   int n, i, j, mi, temp,arr[10];
   printf("Enter the number of elements: ");
   scanf("%d", &n);
   printf("Enter %d elements: ", n);
   for (i = 0; i < n; i++)
   scanf("%d", &arr[i]);
   for (i = 0; i < n-1; i++) {
          mi = i;
          for (j = i+1; j < n; j++) {
             if (arr[j] < arr[mi]) {
                    mi = j;
             }
          }
          temp = arr[mi];
          arr[mi] = arr[i];
          arr[i] = temp;
   }
   printf("\nSorted array: ");
   for (i = 0; i < n; i++)
          printf("%d ", arr[i]);
    getch();
}
```

//EX.NO.13. IMPLEMENTATION OF MERGE SORT

```
#include <stdio.h>
void main() {
   int a[30], temp[30], n, i, j, k, mid, l, r,m;
   clrscr();
   printf("Enter number of elements: ");
   scanf("%d", &n);
   printf("Enter array elements: ");
   for (i = 0; i < n; i++)
   scanf("%d", &a[i]);
   for (mid = 1; mid < n; mid *= 2) {
         for (l = 0; l < n - mid; l += 2 * mid) {
            r = (l + 2 * mid - 1 < n) ? l + 2 * mid - 1 : n - 1;
             m = l + mid - 1;
            i = l, j = m + 1, k = l;
            while (i <= m && j <= r) {
                  if (a[i] < a[j])
                  temp[k++] = a[i++];
                  else
                  temp[k++] = a[j++];
            }
            while (i <= m)
            temp[k++] = a[i++];
            while (j <= r)
            temp[k++] = a[j++];
            for (i = l; i <= r; i++)
            a[i] = temp[i];
         }
   }
   printf("\nSorted array: ");
   for (i = 0; i < n; i++)
   printf("%d ", a[i]);
   getch();
}
```

//EX.NO.14(A) Implementation of Open Addressing - Linear Probing

```
#include <stdio.h>
void main() {
   int ht[10],i, key, ind;
   clrscr();
   printf("Enter key to insert: ");
   for (i = 0; i < 10; i++)
   ht[i] = -1;
   for (i = 0; i < 7; i++) {
          scanf("%d", &key);
         ind = key % 10;
         while (ht[ind] != -1)
            ind = (ind + 1) % 10;
         ht[ind] = key;
   }
   printf("\n Hash Table: ");
   for (i = 0; i < 10; i++)
         printf("%d ", ht[i]);
    getch();
}
```

//EX.NO.14(B) Implementation of Open Addressing - Quadratic Probing

```
#include <stdio.h>
void main() {
int ht[10],i, key, ind, count;
clrscr();
printf("Enter key to insert: ");
for (i = 0; i < 10; i++)
  ht[i] = -1;
 for (i = 0; i < 7; i++) {
   scanf("%d", &key);
   count = 0;
   ind = key % 10;
   while (ht[ind] != -1 && count < 10) {
      count++;
      ind = (key + count * count) % 10;
     }
   if (count < 10)
   ht[ind] = key;
  }
 printf("\nHash Table: ");
 for (i = 0; i < 10; i++)
      printf("%d ", ht[i]);
  getch();
 }
```