Lab Experiments

Data Structures and Algorithms (24IC203P)

|  |
| --- |
| SCHOOL OF TECHNOLOGY  PANDIT DEENDAYAL ENERGY UNIVERSITY  GANDHINAGAR, GUJARAT, INDIA          Information and Communication  Technology LAB File (2025-26)  Data Structures and Algorithms LAB  (24IC203P)    **Student Name: Ronit Kundnani**  **Enrollment No.: 24BIT100**  **Course with Semester: 3rd**  **Division: 2**  **Group: H5**    **Lab Instructors: Dr. Rutvij H. Jhaveri**  **Dr. Rajendra Choudhary**  **Dr. Davinder Singh** |

**Table of Contents**

**Course Outcomes (COs):**

On completion of the course, student will be able to

**CO1** - Differentiate linear and non-linear data structures.

**CO2** - Enhance logical reasoning and programming skills.

**CO3** - Implement linear and non-linear data structures.

**CO4** - Identify suitable data structures to solve complex computing problems.

**CO5** - Apply the algorithms on the small and large data sets.

**CO6** - Design and implement an appropriate hashing function for an application.

**List of Experiments:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No.** | **Experiments Performed** | **Mapped CO** | **Mapped PO** |
| 1 | **Revision of Arrays and Structures** | CO-1 | PO - 1 |
| 2 | **Revision of Pointers and Structures** | CO-1 | PO - 1 |
| 3 | **Stack** | CO-2, CO-3 | PO – 1, 2, 4, 5 |
| 4 | **Stack Applications** | CO-2, CO-3 | PO – 1, 2, 3, 5 |
| 5 | **Queue** | CO-2, CO-3 | PO – 1, 2, 4, 5 |
| 6 | **Linked List** | CO-2, CO-3 | PO – 1, 2, 4, 5 |
| 7 | **Trees** | CO-2, CO-4 | PO – 1, 2, 4, 5 |
| 8 | **Graphs** | CO-2, CO-5 | PO – 1, 2, 4, 5 |
| 9 | **Sorting** | CO-2, CO-6 | PO – 1, 2, 4, 5 |
| 10 | **Searching** | CO-2, CO-6 | PO – 1, 2, 4, 5 |
| 11 | **Mini Project** | CO - 1, 2, 3, 4, 5, 6 | PO - 1, 2, 3,  5, 8, 9, 10, 11 |
| 12 |
| 13 |

**Pre-requisites:**

**1.** Basic knowledge of Computer Programming Fundamentals: Understanding of Arrays, Strings, Structure, Pointers and other fundamentals.

|  |  |
| --- | --- |
|  | **Instructions** |
| i. | Make all the programs using C language only. |
| ii. | You are allowed to use gcc compiler and command prompt for running the programming. Also, you are allowed to use any IDE (like, CODEBLOCKS) for implementation. |
| iii. | Each program must print your name and enrollment number in the starting of your each C program. |
| iv. | In each lab after making the programs, paste the code (in text format) with the output (snapshot of the output) in this file. |
| v. | Evaluation will be carried out based on Correctness of Code, Code Efficiency (Optimization), Use of Appropriate Data Structures, Code Readability and Structure, Documentation and Comments, Modularity and Reusability, Test Cases and Output,  Handling of Errors, Memory Management, Innovative Approach and Viva Performance,. |
| vi. | You must submit only a soft copy of the lab manual. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Exp. No.** | **Experiment Title** | **Date** | **Signature** |
| 1  **CODE**  **OUTPUT**  **CODE**  **OUTPUT**  **CODE**  **OUTPUT** | **[Revision of Arrays and Structures]**  Programs covering basics of Arrays and Structure.   1. Write a program in C to perform addition of two 3x3 matrices.   #include <stdio.h>  void add(int arr1[3][3],int arr2[3][3],int res[3][3]){  for(int i=0;i<3;i++){  for(int j=0;j<3;j++){  res[i][j]=arr1[i][j]+arr2[i][j];  }  }  }  int main(){  printf("Student Name: Ronit Kundnani\n");  printf("Student RollNo: 24BIT100\n");  int arr1[3][3];  int arr2[3][3];  int res[3][3];  printf("Enter the Elements of the matrix 1 \n");  for(int i=0;i<3;i++){  for(int j=0;j<3;j++){  printf("Enter %d %d th element:",i,j);  scanf("%d",&arr1[i][j]);  }  printf("\n");  }  printf("Enter the Elements of the matrix 2: \n");  for(int i=0;i<3;i++){  for(int j=0;j<3;j++){  printf("Enter %d %d th element:",i,j);  scanf("%d",&arr2[i][j]);  }  printf("\n");  }    add(arr1,arr2,res);  for(int i=0;i<3;i++){  for(int j=0;j<3;j++){  printf("%d \t",res[i][j]);  }  printf("\n");  }  return 0;  }       1. Create a structure Student in C with student name, student roll number and student address as its data members. Create the array of type student and print their values.   #include <stdio.h>  struct Student{  char name[10];  int rollno;  char address[20];  };  int main(){  printf("Student Name: Ronit Kundnani\n");  printf("Student RollNo: 24BIT100\n");  int n;  printf("Enter the number of students:");  scanf("%d",&n);  struct Student s1[n];  for(int i=0;i<n;i++){  printf("Enter The details of the %d th student \n",(i+1));  printf("Enter student name:");  scanf("%s",&s1[i].name);  printf("Enter student RollNo:");  scanf("%d",&s1[i].rollno);  printf("Enter student address:");  scanf("%s",&s1[i].address);  printf("\n");  }  printf("\n \n");  for(int i=0;i<n;i++){  printf("Student %d :\n",i+1);  printf("student name: %s\n",s1[i].name);  printf("student RollNo:%d\n",s1[i].rollno);  printf("student address:%s\n",s1[i].address);  }  return 0;  }    **Practice Problem:**   1. Write a program in C that obtains the minimum and maximum element from the array. Modify this program to give the second largest and second smallest element of the array.   #include <stdio.h>  int main(){  printf("Student Name: Ronit Kundnani\n");  printf("Student RollNo: 24BIT100\n");  int n;  printf("Enter the size of array: ");  scanf("%d",&n);  int arr[n];  printf("Enter the elements of the array: ");  for(int i=0;i<n;i++){  scanf("%d",&arr[i]);  }  int largest=arr[0] , secondLarge=arr[0];  int smallest=arr[0] , secondSmall=arr[0];  for(int i=0;i<n;i++){  if(arr[i]>largest){  secondLarge=largest;  largest=arr[i];  }  else if(arr[i]>secondLarge && arr[i]<largest){  secondLarge=arr[i];  }  if(arr[i]<smallest){  secondSmall=smallest;  smallest=arr[i];  }  else if(arr[i]<secondSmall && arr[i]>smallest){  secondSmall=arr[i];  }  }  printf("The smallest and Second smallest value of the array are %d and %d \n",smallest,secondSmall);  printf("The largest and Second largest value of the array are %d and %d",largest,secondLarge);  return 0;  } |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 2  **CODE**  **OUTPUT**  **CODE**  OUTPUT    **CODE**  **OUTPUT** | **[Revision of Pointers and Structures]**  Programs covering structure definition, array of structure, nested structures etc.     1. Write a program in C to implement arrays of pointers and pointers to arrays.   #include <stdio.h>  int main(){      printf("Student Name: Ronit Kundnani\n");      printf("Student RollNo: 24BIT100\n");      int a=10,b=20,c=30;      int \*ptr[]={&a,&b,&c};      printf("Array Of Pointers\n");      for(int i=0;i<3;i++){          printf("%d\n",\*ptr[i]);      }      printf("Pointers Of Array \n");      int arr[]={1,2,3,4,5,6,7,8,9,10};      int \*p=arr;      for(int i=0;i<10;i++){          printf("%d\n",\*(p+i));      }      return 0;  }     1. Write a program in C to implement pointers to structures.   #include <stdio.h>  #include <string.h>  struct Student{  char name[10];  int rollno;  char address[20];  };  int main(){  printf("Student Name: Ronit Kundnani\n");  printf("Student RollNo: 24BIT100\n");  struct Student s1;  struct Student \*ptr;  ptr=&s1;  strcpy(ptr->name,"ronit");  ptr->rollno=10;  strcpy(ptr->address,"BlahBlah");  printf("name:%s\n",ptr->name);  printf("rollno:%d\n",ptr->rollno);  printf("address:%s\n",ptr->address);  return 0;  }    **Practice Problem:**   1. Write a program in C to perform swapping of two numbers by passing addresses of the variables to the functions.   #include <stdio.h>  void swap(int \*a,int \*b){  int temp=\*a;  \*a=\*b;  \*b=temp;  }  int main(){  printf("Student Name: Ronit Kundnani\n");  printf("Student RollNo: 24BIT100\n");  int a=10,b=20;  printf("Values Before swapping a:%d and b:%d\n",a,b);  swap(&a,&b);  printf("swapped values: a:%d b:%d",a,b);  return 0;  } |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 3  **CODE**  **OUTPUT**  **CODE**  **OUTPUT** | **[Stack]**  Programs covering stack and applications.   1. Implement a stack using an array. Implement the following functions:    1. isEmpty(): Tests whether stack is empty or not.    2. push(): Adds new element to the stack.    3. pop(): Removes top element from the stack.    4. top(): Returns value of the top element.   #include<stdio.h>  #define MAX 20  struct stack{  int s[MAX];  int top;  };  struct stack s1;  int isEmpty(){  if(s1.top == -1){  printf("Stack is Empty or Underflow\n");  return 1;  }  return 0;  }  int isFull(){  if(s1.top == MAX-1){  printf("Stack is Full or Overflow\n");  return 1;  }  return 0;  }  void push(int ele){  if(isFull()){  printf("You can't push an element\n");  }else{  s1.top += 1;  s1.s[s1.top]=ele;  printf("%d is pushed into stack\n", ele);  }  }  void pop(){  if(isEmpty()){  printf("You can't pop an element\n");  }else{  printf("%d is popped from the stack\n" ,s1.s[s1.top]);  s1.top -= 1;  }  }  void Top(){  printf("Top element of stack is %d\n" ,s1.s[s1.top]);  }  int main(){  printf("Student Name: Ronit Kundnani\n");  printf("Student RollNo: 24BIT100\n");  s1.top = -1;  push(1);  push(3);  push(5);  pop();  push(2);  push(4);  pop();  pop();  Top();  pop();  pop();  pop();  return 0;  }    **Practice Problems:**   1. Write a program to reverse a given string using a stack.   #include<stdio.h>  #include<string.h>  #define MAX 20  struct stack{  char s[MAX];  int top;  };  struct stack s1;  int isEmpty(){  if(s1.top == -1){  printf("Stack is Empty or Underflow\n");  return 1;  }  return 0;  }  int isFull(){  if(s1.top == MAX-1){  printf("Stack is Full or Overflow\n");  return 1;  }  return 0;  }  void push(char ele){  if(isFull()){  printf("You can't push an element\n");  }else{  s1.top += 1;  s1.s[s1.top]=ele;  //printf("%c is pushed into stack\n", ele);  }  }  char pop(){  if(isEmpty()){  printf("You can't pop an element\n");  }else{  char popped = s1.s[s1.top];  //printf("%c is popped from the stack\n" ,s1.s[s1.top]);  s1.top -= 1;  return popped;  }  }  void Top(){  printf("Top element of stack is %c\n" ,s1.s[s1.top]);  }  int main(){  printf("Student Name: Rudresh Monpara\n");  printf("Student RollNo: 24BIT136\n");  s1.top = -1;  int i;  char A[MAX];  gets(A);  for(i=0;i<=strlen(A);i++){  push(A[i]);  }  printf("Reverse of String :\n");  for(i=0;i<=strlen(A);i++){  printf("%c",pop());  }  return 0;  }     1. Given an expression, write a program to examine whether the pairs and the orders of “{“, “}”, “(“, “)”, “[“, “]” are correct in the expression or not.   Example:  Input: exp = “[( )]{}{[( )( )]( )}” Output: Balanced  Input: exp = “” Output: Not Balanced  #include <stdio.h>  #include <stdlib.h>  struct stack  {      int size;      int top;      char \*arr;  };  int **isEmpty**(struct stack \**s*)  {      return *s*->top == -1;  }  int **isFull**(struct stack \**s*)  {      return *s*->top == *s*->size - 1;  }  char **push**(struct stack \**s*, char *value*)  {      if (**isFull**(*s*))      {  **printf**("Stack overflow\n");          return 0;      }      else      {  *s*->top++;  *s*->arr[*s*->top] = *value*;          return *value*;      }  }  char **pop**(struct stack \**s*)  {      if (**isEmpty**(*s*))      {  **printf**("Stack underflown\n");          return 0;      }      else      {          return *s*->arr[*s*->top--];      }  }  int **match**(char *f*, char *b*)  {      if (*f* == '{' && *b* == '}')      {          return 1;      }      if (*f* == '[' && *b* == ']')      {          return 1;      }      if (*f* == '(' && *b* == ')')      {          return 1;      }      return 0;  }  int **parenthesisMatch**(char \**exp*)  {      struct stack \*sp;      sp->size = 100;      sp->top = -1;      sp->arr = (char \*)**malloc**(sp->size \* sizeof(char));      char popped\_ch;      for (int i = 0; *exp*[i] != '\0'; i++)      {          if (*exp*[i] == '(' || *exp*[i] == '{' || *exp*[i] == '[')          {  **push**(sp, *exp*[i]);          }          else if (*exp*[i] == ')' || *exp*[i] == '}' || *exp*[i] == ']')          {              if (**isEmpty**(sp))              {                  return 0;              }              popped\_ch = **pop**(sp);              if (!**match**(popped\_ch, *exp*[i]))              {                  return 0;              }          }      }      if (**isEmpty**(sp))      {          return 1;      }      else      {          return 0;      }  }  int **main**()  {  **printf**("Student Name: Ronit Kundnani\n");  **printf**("Student RollNo: 24BIT100\n");      char \*exp = "[( )]{}{[( )(  )]( )}";      if (**parenthesisMatch**(exp))      {  **printf**("Parenthesis Matching\n");      }      else      {  **printf**("Parenthesis not Matching \n");      }      return 0;  } | | | |  |  |
|  |  |  |  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 4  Code | **[Stack]**   1. Convert the given infix expression into postfix expression using stack.   #include <stdio.h>  #include <string.h>  #include <stdlib.h>  struct stack {  int size;  int top;  char \*arr;  };  int isEmpty(struct stack \*s) {  return s->top == -1;  }  int isFull(struct stack \*s) {  return s->top == s->size - 1;  }  char pop(struct stack \*s) {  if (isEmpty(s)) {  printf("stack underflow\n");  return 0;  }  return s->arr[s->top--];  }  char push(struct stack \*s, char value) {  if (isFull(s)) {  printf("stack overflow\n");  return -1;  }  s->arr[++s->top] = value;  return value;  }  char top(struct stack \*s) {  if (isEmpty(s)) return '\0';  return s->arr[s->top];  }  int isOperator(char a) {  return (a == '+' || a == '-' || a == '\*' || a == '/');  }  int precedence(char a) {  if (a == '\*' || a == '/')  return 3;  else if (a == '+' || a == '-')  return 2;  else if (a == '(')  return 1; // lowest precedence  else  return 0;  }  char \*inftopost(char \*infix) {  struct stack \*sp = (struct stack \*)malloc(sizeof(struct stack));  sp->size = 100;  sp->top = -1;  sp->arr = (char \*)malloc(sp->size \* sizeof(char));  char \*postfix = (char \*)malloc((strlen(infix) + 1) \* sizeof(char));  int i = 0, j = 0;  while (infix[i] != '\0') {  if (infix[i] == '(') {  push(sp, infix[i]);  i++;  }  else if (infix[i] == ')') {  while (!isEmpty(sp) && top(sp) != '(') {  postfix[j++] = pop(sp);  }  pop(sp); // remove '('  i++;  }  else if (!isOperator(infix[i])) {  postfix[j++] = infix[i++];  }  else {  while (!isEmpty(sp) && precedence(top(sp)) >= precedence(infix[i])) {  postfix[j++] = pop(sp);  }  push(sp, infix[i]);  i++;  }  }  while (!isEmpty(sp)) {  postfix[j++] = pop(sp);  }  postfix[j] = '\0';  return postfix;  }  int main()  {  printf("Student Name: Ronit Kundnani\n");  printf("Student RollNo: 24BIT100\n");  char \*infix = "a+b+c";  printf("%s\n", inftopost(infix));  return 0;  }      **Practice Problems:**   1. Write a program to evaluate the following given postfix expressions:   e.g. Input: 231 ∗ +9– Output: -4  #include <stdio.h>  #include <stdlib.h>  struct stack  {      int top;      int size;      char \*arr;  };  int **isEmpty**(struct stack \**s*)  {      return *s*->top == -1;  }  int **isFull**(struct stack \**s*)  {      return *s*->top == *s*->size - 1;  }  char **pop**(struct stack \**s*)  {      if (**isEmpty**(*s*))      {  **printf**("stack underflow\n");          return 0;      }      else      {          return *s*->arr[*s*->top--];      }  }  char **push**(struct stack \**s*, char *value*)  {      if (**isFull**(*s*))      {  **printf**("stack overflow\n");          return -1;      }      else      {  *s*->top++;  *s*->arr[*s*->top] = *value*;          return *value*;      }  }  char **top**(struct stack \**s*)  {      return *s*->arr[*s*->top];  }  int **isOperant**(char *a*)  {      return (*a* >= '0' && *a* <= '9');  }  int **inftopost**(char \**exp*)  {      struct stack \*s = (struct stack \*)**malloc**(sizeof(struct stack));      s->top = -1;      s->size = 100;      s->arr = (char \*)**malloc**(s->size \* sizeof(char));      int i = 0;      while (*exp*[i] != '\0')      {          if (**isOperant**(*exp*[i]))          {  **push**(s, *exp*[i]);          }          else          {              if (*exp*[i] == '+')              {                  int second = **pop**(s) - '0';                  int first = **pop**(s) - '0';                  char c = first + second + '0';  **push**(s, c);              }              else if (*exp*[i] == '-')              {                  int second = **pop**(s) - '0';                  int first = **pop**(s) - '0';                  char c = first - second + '0';  **push**(s, c);              }              else if (*exp*[i] == '\*')              {                  int second = **pop**(s) - '0';                  int first = **pop**(s) - '0';                  char c = first \* second + '0';  **push**(s, c);              }              else if (*exp*[i] == '/')              {                  int second = **pop**(s) - '0';                  int first = **pop**(s) - '0';                  char c = first / second + '0';  **push**(s, c);              }          }          i++;      }      return **pop**(s) - '0';  }  int **main**()  {  **printf**("Student Name: Ronit Kundnani\n");  **printf**("Student RollNo: 24BIT100\n");      char \* exp="23+54\*+";  *// char \*exp = "22/";*  **printf**("Result:%d", **inftopost**(exp));      return 0;  }       1. Convert the given infix expression into prefix expression using stack.       #include <stdio.h>  #include <string.h>  #include <stdlib.h>  struct stack {  int size;  int top;  char \*arr;  };  int isEmpty(struct stack \*s) {  return s->top == -1;  }  int isFull(struct stack \*s) {  return s->top == s->size - 1;  }  char pop(struct stack \*s) {  if (isEmpty(s)) {  return 0;  }  return s->arr[s->top--];  }  char push(struct stack \*s, char value) {  if (isFull(s)) {  return -1;  }  s->arr[++s->top] = value;  return value;  }  char top(struct stack \*s) {  if (isEmpty(s)) return '\0';  return s->arr[s->top];  }  int isOperator(char a) {  return (a == '+' || a == '-' || a == '\*' || a == '/');  }  int precedence(char a) {  if (a == '\*' || a == '/')  return 2;  else if (a == '+' || a == '-')  return 1;  else  return 0;  }  void reverse(char \*exp) {  int n = strlen(exp);  for (int i = 0; i < n / 2; i++) {  char temp = exp[i];  exp[i] = exp[n - i - 1];  exp[n - i - 1] = temp;  }  }  char \*inftopost(char \*infix) {  struct stack \*sp = (struct stack \*)malloc(sizeof(struct stack));  sp->size = 100;  sp->top = -1;  sp->arr = (char \*)malloc(sp->size \* sizeof(char));  char \*postfix = (char \*)malloc((strlen(infix) + 1) \* sizeof(char));  int i = 0, j = 0;  while (infix[i] != '\0') {  if (infix[i] == '(') {  push(sp, infix[i]);  }  else if (infix[i] == ')') {  while (!isEmpty(sp) && top(sp) != '(') {  postfix[j++] = pop(sp);  }  pop(sp); // remove '('  }  else if (!isOperator(infix[i])) {  postfix[j++] = infix[i];  }  else {  while (!isEmpty(sp) && precedence(top(sp)) >= precedence(infix[i])) {  postfix[j++] = pop(sp);  }  push(sp, infix[i]);  }  i++;  }  while (!isEmpty(sp)) {  postfix[j++] = pop(sp);  }  postfix[j] = '\0';  free(sp->arr);  free(sp);  return postfix;  }  char \*inftopre(char \*infix) {  int n = strlen(infix);  char \*rev = (char \*)malloc((n + 1) \* sizeof(char));  strcpy(rev, infix);  reverse(rev);  // Swap brackets  for (int i = 0; i < n; i++) {  if (rev[i] == '(') rev[i] = ')';  else if (rev[i] == ')') rev[i] = '(';  }  char \*post = inftopost(rev);  reverse(post); // reverse postfix to get prefix  free(rev);  return post;  }  int main() {  printf("Student Name: Ronit Kundnani\n");  printf("Student RollNo: 24BIT100\n");  char exp[] = "a+b+c";  printf("Infix: %s\n", exp);  printf("Prefix: %s\n", inftopre(exp));    return 0;  } |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 5 | **[Queue]**   1. Implement various functionalities of Queue using Arrays. For example: insertion, deletion, front element, rear element etc.   **Practice Problems:**   1. Implement Priority Queue, where every element has a priority associated with it. Perform operations like Insertion and Deletion in a priority queue. 2. Implement Double Ended Queue that supports following operation:    1. insertFront(): Adds an item at the front of Deque.    2. insertLast(): Adds an item at the rear of Deque.    3. deleteFront(): Deletes an item from the front of Deque.    4. deleteLast(): Deletes an item from the rear of Deque. |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 6 | **[Linked List]**   1. Implement doubly linked list with following functions:    1. insertAtBeginning(): Inserts a node at the end of the list    2. insertAtEnd(): Inserts a node at the end of the list.    3. insertAtPosition() – Inserts a node at a specific index.    4. deleteFromBeginning(): Deletes the first node.    5. deleteFromEnd(): Deletes the last node.    6. deleteFromPosition(position) : Deletes a node from a specific position.    7. forwardTraverse(): Traverses and displays the list from head to tail.    8. backwardTraverse(): Traverses and displays the list from tail to head    9. search: search(): Searches for a node with given data and returns its position (or index).   **Practice Problems:**   1. Implement Stack and Queue using linked list with suitable functions (e.g. insert, delete, checking stack/queue is empty or full). 2. Implement Circular and Double ended queues using linked list with suitable functions. 3. Write a program that takes ***two sorted lists*** as inputs and merge them into one sorted list.   For example, if the first linked list *A* is 5 =>10 =>15, and the other linked list *B* is 2 => 3 => 20, then output should be 2 => 3 => 5 => 10 => 15 => 20. |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 7 | **[Trees]**   1. Implement binary search tree with   insert(),inorderTraverse(), preorderTraverse(), search() and findHeight().  **Practice Problems:**   1. Implement binary search tree with noninorderTraverse(), preorderTraverse(), countNodes(), and countLeafNodes(). | recursive functions:  postorderTraverse(),  recursive functions: insert(), postorderTraverse(), |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 8 | **[Graphs]**   1. Implement Minimum Spanning Tree (MST) using the greedy   Kruskal’s algorithm **Practice Problems:**   1. For a given graph 𝐺 = (𝑉, 𝐸), study and implement the Breadth First   Search and Depth First Search.   1. Given a directed graph, check whether the graph contains a cycle or not. Your function should return true if the given graph contains at least one cycle, else return false. Perform same task for undirected graph as well. 2. Implement Minimum Spanning Tree (MST) using the Prim’s algorithm. |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 9 | **[Sorting]**   1. Implement Insertion sort to arrange the numbers in ascending order. 2. Implement Quick sort to arrange the numbers in ascending order. **Practice Problems:** 3. Implement Bubble sort to arrange the numbers in ascending order. 4. Implement Selection sort to arrange the numbers in descending order. 5. Implement Merge sort to arrange the numbers in ascending order |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 10 | **[Searching]**   1. Implement Linear search to search a number from a sorted array. 2. Implement Binary search to search a number from a sorted array. **Practice Problem:** 3. Given a sorted array of integers (in descending order), writea C program to find the first and last occurrence of a given number x. If the element is not found, print suitable message. |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 11 | **Experiment No. 11, 12, 13 [Mini Project and Problem Solving]**  These laboratory sessions are dedicated for doing a mini project and doubt solving sessions. Students are expected to do a mini project by forming a group of 5 on the concepts they have learned in data structures. They are free to select any mini project as per their choice. If they are not able to decide, they can select any one from the following:   1. **Hospital Patient Record System:** This system manages patient information in a hospital. It stores patient records, including ID, name, age, disease, date of admission, and status   (admitted/discharged). You can implement: Add patient details, Search patient by ID, Discharge patient, View all current admitted patients  Data Structures Used: Linked Lists (for dynamic record handling), Queues (for patient appointments)   1. **Bank Management System:** This project simulates basic banking operations: Account creation, Deposit/withdrawal, Balance check, Displaying account details. Each account can be stored in a hash table or binary search tree to allow fast searching and updates.      1. **College Admission Management System:** Simulate the admission process of students into different courses based on merit and availability. Features: Student registration, Merit list sorting, Course seat allocation, Waiting list generation Data Structures Used: Arrays and Structures (for sorting using marks), Queues (for waitlist), Priority Queue (optional for merit) 2. **Online Food Ordering System:** Simulate an online food delivery system. Functions include: Display menu, Place order, Cancel order, Show bill, Maintain order history Data Structures Used: Stacks (for order undo), Queues or Linked Lists (for order processing), Arrays (for menu) |  |  |