KG REDDY College of Engineering & Technology
AN AUTONOMOUS INSTITUTION

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

DATA STRUCTURES LAB MANUAL

Subject Code : KGR23ACD218

Regulation : KGR23

Academic Year : 2025-2026

II B. TECH I SEMESTER
COMPUTER SCIENCE AND ENGINEERING

KG REDDY COLLEGE OF ENGINEERING AND TECHNOLOGY

Affiliated o JNTUH, Chilkur,(V), Moinabad(M) R. R Dist, TS-501504

VISION AND MISSION OF THE INSTITUTION

VISION:

To become an institution which is internationally recognized for its holistic approach to engineering, innovative teaching and learning culture, research and entrepreneurial ecosystem, and sustainable social impact in the community.

MISSION:

• To offer undergraduate and post-graduate programs which are supported through industry relevant curriculum and innovative teaching and learning processes that would help students succeed intheir professional careers.
• To provide faculty and students with an ecosystem that fosters innovation, research, entrepreneurship, and international exposure through strategic partnerships with governmentorganizations and collaboration with industries.
• To provide holistic learning environment to students which will contribute to their personal and professional growth and enable them to become leaders in their respective fields.
• To contribute to the development of the region by using our technological expertise to work with nearby communities and support them in their social and economic development.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VISION:

To be recognized as a department of excellence by stimulating a learning environment inwhich students and faculty will thrive and grow to achieve their professional, institutional and societal goals.

MISSION:

• To provide high quality technical education to students that will enable life-long learning and build expertise in advanced technologies in Computer Science and Engineering.
• To promote research and development by providing opportunities to solve complex engineering problems in collaboration with industry and government agencies.
• To encourage professional development of students that will inculcate ethical values and leadership skills through entrepreneurship while working with the community to address societalissues.

PROGRAM EDUCATIONAL OBJECTIVES

PEO 1: Graduates will provide solutions to difficult and challenging issues in their profession by applying computer science and engineering theory and principles.
PEO 2: Graduates have successful careers in computer science and engineering fields or will be able to successfully pursue advanced degrees.
PEO 3: Graduates will communicate effectively, work collaboratively and exhibit high levels of professionalism, moral and ethical responsibility.
PEO 4: Graduates will develop the ability to understand and analyze engineering issues in abroader perspective with ethical responsibility towards sustainable development.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PROGRAM OUTCOMES

PO I: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO II: Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO III: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO IV: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO V: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
PO VI: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO VII: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of and need for sustainable development.
PO VIII: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO IX: Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO X: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO XI: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO XII: Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PROGRAM SPECIFIC OUTCOMES

PSO1: The Computer Science and Engineering graduates are able to analyze, design, develop, test and apply management principles, mathematical foundations in the development of intelligentsystems with computational solutions, make them to expert in designing the secure application and hardware prototype
PSO2: The graduating student will be analyze the contemporary research issues in different areasof computer science & engineering and explore research gaps, analyze and carry out research in the specialized/emerging areas.
PSO3: Develop their skills to solve problems in the broad area of programming concepts and appraise environmental and social issues with ethics and manage different projects in multi- disciplinary field to conducive in cultivating skills for successful career, entrepreneurship and higher studies.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DATA STRUCTURES LAB

Prerequisites: A Course on "Python Programming".

Course Objectives: The course will help students

1. To impart the basic concepts of data structures and algorithms, searching and sorting techniques.
2. To gain the knowledge of about stacks, queues, lists, trees and graphs
3. To understand about writing algorithms and step by step approach in solving problems with the help of fundamental data structures.
4. To strengthen the ability to the students to identify and apply the suitable data structure for the given real world problem.
5. To enable them to gain knowledge in practical applications of data structures.

Course Outcomes: Upon completion of this course, the students will be able to

CO1: Implement linear data structures such as arrays, linked lists, stacks, queues for efficient data organization and manipulation.

CO2: Develop solutions using data structures such as trees, graphs, heaps, hash tables for efficient search and retrieval of data.

CO3: Select and apply appropriate techniques for searching trees and graph Problems.

CO4: Select and apply appropriate techniques for sorting problems.

CO5: Work independently and communicate effectively in oral and written forms.

List of Experiments:

1. A college has N number of students and the following details of all the students are maintained – register number, name, branch, phone number. Write a program to store the details of the students using a singly linked list. Develop functions to perform the following operations on the data.

   a. Insert new student's details

   b. Display the details of the students

   c. Display the total number of students

   d. Delete a given student's information

2. A company has N number of employees and it maintains the following details of each of its employees: ID, department, salary, phone number. Develop a menu driven program using doubly linked list to store the employees' data. Develop functions to perform the following operations on the data.

   a. Add and delete employees

   b. Display total number of employees

   c. Display details of employees with salary more than Rs. 50,000

   d. Display the phone number of the employee given the ID

3. Write a program that uses functions to perform the following operations on circular linked list:

   i) Creation ii) Insertion iii) Deletion iv) Traversal

4. Write a program that implement stack (its operations) using Arrays

5. Write a program that implement Queue (its operations) using Arrays

6. Write a program that implements the following sorting methods to sort a given list of integers in ascending order

   i) Bubble sort ii) Selection sort iii) Quick Sort

7. a) Write a program to sort a given set of integers using merge sort.

   b) Write a program to read the marks obtained by students in a mathematics examination and store the data using a heap data structure. Find out the maximum and minimum marks obtained by the students.

8. Write a program that use both recursive and non-recursive functions to perform the following searching operations for a Key value in a given list of integers:

   i) Linear search ii) Binary search

9. Write a program to implement the tree traversal methods.

10. There are train paths between cities. If there is a train between city A and city B then there is a route between the cities. The cost of the route is the distance between city A and city B. Represent the train travel route information as a graph. The node can be represented by the name of the city. Write a program to perform the following operations.

    a. Traverse the graph and display the details of all trains between the cities along with the cost using breadth-first method.

    b. Traverse the graph and display the details of all

Additional Experiments:

1. a) Write a Program to Sort the set of elements: (i.e., numbers or strings)

   i). Quick Sort ii). Heap Sort. iii). Merge Sort

   b) Write a Program to Sort the set of elements by using External sorting algorithms

   i). Shell Sort ii). Linear Sort. iii). Binary Sort iv) Fibonacci sort

2. Write C programs for implementing the following graph traversal algorithms:

   a)Depth first traversal b)Breadth first traversal.

3. Write a Program to implement all functions of a Dictionary by using Hashing.

4. Write a program to Implement Insertion and Deletion Operations on AVL Trees.

5. Write a Program to Implement Insertion, Deletion and Search Operations on SPLAY Trees.

Note: Use Classes and Objects to implement the above programs.

TEXT BOOKS:

1. Data structures and algorithms in python by Michael T. Goodrich.
2. Data Structures and Algorithmic Thinking with Python by Narasimha Karumanchi.

REFERENCE BOOKS:

1. Hands-On Data Structures and Algorithms with Python: Write complex and powerful code using the latest features of Python 3.7, 2nd Edition by Dr. Basant Agarwal, Benjamin Baka.
2. Data Structures and Algorithms with Python by Kent D. Lee and Steve Hubbard.
3. Problem Solving with Algorithms and Data Structures Using Python by Bradley N Miller and David L. Ranum.
4. Core Python Programming -Second Edition, R. Nageswara Rao, Dreamtech Press.

1. A college has N number of students and the following details of all the students are maintained – register number, name, branch, phone number. Write a program to store the details of the students using a singly linked list. Develop functions to perform the following operations on the data.

A. Insert new student's details

B. Display the details of the students

C. Display the total number of students

D. Delete a given student's information

Source Code:

```
// Define the structure for a student
typedef struct Student {
    int register_number;
    char* name;
    char* branch;
    char* phone_number; struct
    Student* next;
} Student;

// Function to create a new student node
Student* createStudent(int register_number, char* name, char* branch, char* phone_number) { Student*
    new_student = (Student*) malloc(sizeof(Student));
    new_student->register_number = register_number;
    new_student->name = (char*) malloc(strlen(name) + 1);
    strcpy(new_student->name, name);
    new_student->branch = (char*) malloc(strlen(branch) + 1);
    strcpy(new_student->branch, branch);
    new_student->phone_number = (char*) malloc(strlen(phone_number) + 1);
    strcpy(new_student->phone_number, phone_number);
    new_student->next = NULL; return
    new_student;
```

```
}

// Function to insert a new student into the linked list
void insertStudent(Student** head, int register_number, char* name, char* branch, char* phone_number) {
    Student* new_student = createStudent(register_number, name, branch, phone_number); if (*head
    == NULL) {
        *head = new_student;
    } else {
        Student* current = *head;
        while (current->next!= NULL) { current =
            current->next;
        }
        current->next = new_student;
    }
}

// Function to display the details of all students void
displayStudents(Student* head) {
    Student* current = head;
    while (current!= NULL) {
        printf("Register Number: %d\n", current->register_number); printf("Name:
        %s\n", current->name);
        printf("Branch: %s\n", current->branch);
        printf("Phone Number: %s\n\n", current->phone_number); current =
        current->next;
    }
}

// Function to display the total number of students int
countStudents(Student* head) {
    int count = 0;
```

```
    Student* current = head;
    while (current!= NULL) {
        count++;
        current = current->next;
    }
    return count;
}

// Function to delete a student with a given register number void
deleteStudent(Student** head, int register_number) {
    if (*head == NULL) return;

    if ((*head)->register_number == register_number) { Student*
        temp = *head;
        *head = (*head)->next;
        free(temp->name); free(temp-
        >branch); free(temp-
        >phone_number); free(temp);
        return;
    }

    Student* current = *head;
    while (current->next!= NULL) {
        if (current->next->register_number == register_number) { Student*
            temp = current->next;
            current->next = current->next->next;
            free(temp->name);
            free(temp->branch); free(temp-
            >phone_number); free(temp);
```

```
            return;
        }
        current = current->next;
    }
}

int main() { int
    N;
    printf("Enter the number of students: ");
    scanf("%d", &N);

    Student* head = NULL;

    int i;
    for (i = 0; i < N; i++) { int
        register_number;
        char name[100], branch[100], phone_number[100];

        printf("Enter details of student %d:\n", i + 1);
        printf("Register Number: ");
        scanf("%d", &register_number);
        printf("Name: ");
        scanf("%s", name);
        printf("Branch: ");
        scanf("%s", branch);
        printf("Phone Number: ");
        scanf("%s", phone_number);

        insertStudent(&head, register_number, name, branch, phone_number);
    }
```

```
    // Display all students
    printf("All Students:\n");
    displayStudents(head);

    // Display total number of students
    printf("Total Students: %d\n", countStudents(head));

    int delete_register_number;
    printf("Enter the register number of the student to delete: "); scanf("%d",
    &delete_register_number);

    // Delete a student
    deleteStudent(&head, delete_register_number);

    // Display all students again
    printf("All Students after deletion:\n"); displayStudents(head);

    return 0;
}
```

OUTPUT:

```
Enter the number of students: 3
Enter details of student 1:
Register Number: 501
Name: ram
Branch: cse
Phone Number: 3324534
Enter details of student 2:
Register Number: 502
Name: shayam
```

Branch: cse

Phone Number: 455667 Enter details of student 3:
Register Number: 503
Name: sai
Branch: cse

Phone Number: 688780 All Students:
Register Number: 501
Name: ram
Branch: cse

Phone Number: 3324534

Register Number: 502
Name: shayam Branch: cse
Phone Number: 455667

Register Number: 503
Name: sai
Branch: cse

Phone Number: 688780

Total Students: 3

Enter the register number of the student to delete: 503

2.A company has N number of employees and it maintains the following details of each of its employees: ID, department, salary, phone number. Develop a menu driven program using doubly linked list to store the employees' data. Develop functions to perform the following operations on the data.

E. Add and delete employees

F. Display total number of employees

G. Display details of employees with salary more than Rs. 50,000

H. Display the phone number of the employee given the ID

SOURCE CODE:

```
// Define the structure for an employee typedef
struct Employee {
    int id;
    char    department[100];
    int salary;
    char    phone_number[100];
    struct    Employee*    next;
    struct Employee* prev;
} Employee;


// Function to create a new employee node
Employee* createEmployee(int id, char* department, int salary, char* phone_number) {
    Employee* new_employee = (Employee*) malloc(sizeof(Employee)); new_employee->id =
    id;
    strcpy(new_employee->department,       department);
    new_employee->salary = salary;
    strcpy(new_employee->phone_number,         phone_number);
    new_employee->next = NULL;
    new_employee->prev    =    NULL;
    return new_employee;
}
```

```
// Function to add an employee to the doubly linked list
void addEmployee(Employee** head, Employee** tail, int id, char* department, int salary, char* phone_number) {
    Employee* new_employee = createEmployee(id, department, salary, phone_number); if (*head
    == NULL) {
        *head = new_employee;
        *tail = new_employee;
    } else {
        (*tail)->next = new_employee;
        new_employee->prev = *tail;
        *tail = new_employee;
    }
}

// Function to delete an employee from the doubly linked list
void deleteEmployee(Employee** head, Employee** tail, int id) {
    Employee* current = *head;
    while (current != NULL) { if
        (current->id == id) {
            if (current->prev != NULL) {
                current->prev->next = current->next;
            } else {
                *head = current->next;
            }
            if (current->next != NULL) {
                current->next->prev = current->prev;
            } else {
                *tail = current->prev;
            }
            free(current);
            return;
        }
```

```
        current = current->next;
    }
}

// Function to display the total number of employees int
countEmployees(Employee* head) {
    int count = 0;
    Employee* current = head; while
    (current != NULL) {
        count++;
        current = current->next;
    }
    return count;
}

// Function to display the details of employees with salary more than Rs. 50,000 void
displayHighSalaryEmployees(Employee* head) {
    Employee* current = head; while
    (current != NULL) {
        if (current->salary > 50000) { printf("ID:
            %d\n", current->id);
            printf("Department:     %s\n",     current->department);
            printf("Salary: %d\n", current->salary);
            printf("Phone Number: %s\n\n", current->phone_number);
        }
        current = current->next;
    }
}

// Function to display the phone number of the employee given the ID void
displayPhoneNumber(Employee* head, int id) {
```

```
    Employee* current = head;
    while (current != NULL) { if
    (current->id == id) {
            printf("Phone Number: %s\n", current->phone_number); return;
        }
        current = current->next;
    }
    printf("Employee not found.\n");
}

int main() {
    Employee* head = NULL;
    Employee* tail = NULL;

    int choice;
    while (1) {
        printf("Menu:\n");
        printf("1. Add Employee\n"); printf("2.
        Delete Employee\n");
        printf("3. Display Total Number of Employees\n");
        printf("4. Display Employees with Salary more than Rs. 50,000\n"); printf("5.
        Display Phone Number of an Employee\n");
        printf("6. Exit\n"); printf("Enter
        your choice: "); scanf("%d",
        &choice);

        switch (choice) {
            case 1: {
                int id;
                char department[100], phone_number[100];
```

```
    int                salary;
    printf("Enter ID: ");
    scanf("%d", &id); printf("Enter
    Department: "); scanf("%s",
    department);          printf("Enter
    Salary: "); scanf("%d", &salary);
    printf("Enter Phone Number: ");
    scanf("%s", phone_number);
    addEmployee(&head, &tail, id, department, salary, phone_number); break;
}
case 2: {
    int id;
    printf("Enter ID of the employee to delete: ");
    scanf("%d", &id);
    deleteEmployee(&head, &tail, id); break;
}
case 3: {
    printf("Total Number of Employees: %d\n", countEmployees(head)); break;
}
case 4: { displayHighSalaryEmployees(head);
    break;
}
case 5: {
    int id;
    printf("Enter ID of the employee to display phone number: "); scanf("%d",
    &id);
```

```
            displayPhoneNumber(head,      id);
            break;
        }
        case 6: { printf("Exiting...\n");
            return 0;
        }
        default: {
            printf("Invalid choice. Please try again.\n"); break;
        }
}


}
}
```

OUTPUT:

```
Menu:
1. Add Employee
2. Delete Employee
3. Display Total Number of Employees
4. Display Employees with Salary more than Rs. 50,000
5. Display Phone Number of an Employee
6. Exit
Enter your choice: 1
Enter ID: 101
Enter Department: EEE
Enter Salary: 34355
Enter Phone Number: 789998 Menu:
1. Add Employee
2. Delete Employee
3. Display Total Number of Employees
4. Display Employees with Salary more than Rs. 50,000
5. Display Phone Number of an Employee
6. Exit
Enter your choice: 1
Enter ID: 102
Enter Department: EEE
Enter Salary: 67768
Enter Phone Number: 7697698878
Menu:
1. Add Employee
```

2. Delete Employee
3. Display Total Number of Employees
4. Display Employees with Salary more than Rs. 50,000
5. Display Phone Number of an Employee
6. Exit

Enter your choice: 2
Enter ID of the employee to delete: 101 Menu:
1. Add Employee
2. Delete Employee
3. Display Total Number of Employees
4. Display Employees with Salary more than Rs. 50,000
5. Display Phone Number of an Employee
6. Exit

Enter your choice: 3
Total Number of Employees: 1 Menu:
1. Add Employee
2. Delete Employee
3. Display Total Number of Employees
4. Display Employees with Salary more than Rs. 50,000
5. Display Phone Number of an Employee
6. Exit

Enter your choice: 4 ID:
102
Department: EEE
Salary: 67768
Phone Number: 7697698878

Menu:
1. Add Employee
2. Delete Employee
3. Display Total Number of Employees
4. Display Employees with Salary more than Rs. 50,000
5. Display Phone Number of an Employee
6. Exit

Enter your choice: 5
Enter ID of the employee to display phone number: 101
Employee not found.
Menu:
1. Add Employee
2. Delete Employee
3. Display Total Number of Employees
4. Display Employees with Salary more than Rs. 50,000
5. Display Phone Number of an Employee
6. Exit

2. Write a program that uses functions to perform the following operations on circular

linked list:

i) Creation ii) Insertion iii) Deletion iv) Traversal

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

// Data structure definitions
typedef struct Node {
    int element;
    struct Node* next;
} Node;

typedef Node* List; typedef
Node* Position;

// Function prototypes
void Insert(int element, List list, Position position); void
Delete(int element, List list);
Position Find(int element, List list); Position
FindPrevious(int element, List list); void
Display(List list);

// Function implementations
void Insert(int element, List list, Position position) {
    Node* newNode = (Node*) malloc(sizeof(Node)); if
    (!newNode) {
        printf("Memory out of space\n"); exit(1);
    }
    newNode->element = element;
    newNode->next = position->next;
    position->next = newNode;
```

```
}

void Delete(int element, List list) {
    Position previous = FindPrevious(element, list); if
    (!previous) {
        printf("Element does not exist!!!\n");
        return;
    }
    Node* nodeToDelete = previous->next;
    previous->next = nodeToDelete->next;
    free(nodeToDelete);
}

Position Find(int element, List list) { Position
    current = list->next;
    while (current != list && current->element != element) { current
        = current->next;
    }
    return current;
}

Position FindPrevious(int element, List list) { Position
    current = list;
    while (current->next != list && current->next->element != element) { current =
        current->next;
    }
    return current;
}

void Display(List list) { printf("The
    list elements are: ");
```

```
    Position current = list->next; while
    (current != list) {
        printf("%d -> ", current->element);
        current = current->next;
    }
    printf("\n");
}

// Main program logic int
main() {
    List list = (List) malloc(sizeof(Node)); list-
    >next = list;

    int choice;
    do {
        printf("1. INSERT\t 2. DELETE\t 3. FIND\t 4. PRINT\t 5. QUIT\n");
        scanf("%d", &choice);

        switch (choice) {
            case 1: {
                int element;
                printf("Enter the element to be inserted: ");
                scanf("%d", &element);
                Insert(element, list, list);
                break;
            }

            case 2: {
                int element;
                printf("Enter the element to be deleted: ");
                scanf("%d", &element);
```

```
            Delete(element,    list);
            break;
        }

        case 3: {
            int element;
            printf("Enter the element to be searched: ");
            scanf("%d", &element);
            Position found = Find(element, list); if
            (found == list) {
                printf("Element does not exist!!!\n");
            } else {
                printf("Element exists!!!\n");
            }
            break;
        }

        case 4: { Display(list);
            break;
        }
      }
   } while (choice < 5);

   return 0;
}
```

OUTPUT:

1. INSERT 2. DELETE 3. FIND 4. PRINT 5. QUIT

1

Enter the element to be inserted: 20

1. INSERT 2. DELETE 3. FIND 4. PRINT 5. QUIT

1

Enter the element to be inserted: 30

1. INSERT 2. DELETE 3. FIND 4. PRINT 5. QUIT

1

Enter the element to be inserted: 40

1. INSERT 2. DELETE 3. FIND 4. PRINT 5. QUIT

3

Enter the element to be searched: 20 Element

exists!!!

1. INSERT 2. DELETE 3. FIND 4. PRINT 5. QUIT

4

The list elements are: 40 -> 30 -> 20 ->

1. INSERT 2. DELETE 3. FIND 4. PRINT 5. QUIT

2

Enter the element to be deleted: 30

1. INSERT 2. DELETE 3. FIND 4. PRINT 5. QUIT

4

The list elements are: 40 -> 20 ->

1. INSERT 2. DELETE 3. FIND 4. PRINT 5. QUIT

3. Write a program that implement stack (its operations) using Arrays

```
#include<stdio.h>

int     stack[100],choice,n,top,x,i;
void push(void);
void pop(void); void
display(void);      int
main()
{
   //clrscr();
```

```
    top=-1;
    printf("\n Enter the size of STACK[MAX=100]:");
    scanf("%d",&n);
    printf("\n\t STACK OPERATIONS USING ARRAY");
    printf("\n\t ----------------------------------");
    printf("\n\t 1.PUSH\n\t 2.POP\n\t 3.DISPLAY\n\t 4.EXIT"); do
    {
       printf("\n Enter the Choice:");
       scanf("%d",&choice);
       switch(choice)
       {
          case 1:
          {
             push();
             break;
          }
          case 2:
          {
             pop();
             break;
          }
          case 3:
          {
             display();
             break;
          }
          case 4:
          {
             printf("\n\t EXIT POINT "); break;
          }
          default:
          {
             printf ("\n\t Please Enter a Valid Choice(1/2/3/4)");
          }

       }
    }
    while(choice!=4);
    return 0;
}
void push()
{
    if(top>=n-1)
    {
       printf("\n\tSTACK is over flow");
```

```
    }
    else
    {
        printf(" Enter a value to be pushed:");
        scanf("%d",&x);
        top++;
        stack[top]=x;
    }
}
void pop()
{
    if(top<=-1)
    {
        printf("\n\t Stack is under flow");
    }
    else
    {
        printf("\n\t The popped elements is %d",stack[top]); top--
        ;
    }
}
void display()
{
    if(top>=0)
    {
        printf("\n The elements in STACK \n"); for(i=top;
        i>=0; i--)
            printf("\n%d",stack[i]);
        printf("\n Press Next Choice");
    }
    else
    {
        printf("\n The STACK is empty");
    }

}
```

OUTPUT:

```
Enter the size of STACK[MAX=100]:50
```

STACK OPERATIONS USING ARRAY

---

1. PUSH
2. POP
3. DISPLAY
4. EXIT

Enter the Choice:1
Enter a value to be pushed:3 Enter

the Choice:2

The popped elements is 3 Enter
the Choice:3

The STACK is empty
Enter the Choice:

4. Write a program that implement Queue (its operations) using Arrays

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

#define N 5

void main() {
   int queue[N], ch = 1, front = 0, rear = 0, i, j = 1, x = N;

   printf("Queue using Array\n");
   printf("1. Insertion\n2. Deletion\n3. Display\n4. Exit\n");

   while (ch) {
      printf("\nEnter the Choice: ");
      scanf("%d", &ch);

      switch (ch) {
         case 1:
            if (rear == x) { printf("\nQueue is
               Full\n");
            } else {
               printf("\nEnter no %d: ", j++);
               scanf("%d", &queue[rear++]);
            }
            break;

         case 2:
            if (front == rear) { printf("\nQueue is
               empty\n");
            } else {
```

```
            printf("\nDeleted Element is %d\n", queue[front++]); x++;
        }
        break;

    case 3:
        printf("\nQueue Elements are:\n"); if
        (front == rear) {
            printf("\nQueue is Empty\n");
        } else {
            for (i = front; i < rear; i++) {
                printf("%d\n", queue[i]);
            }
        }
        break;

    case 4:
        exit(0);

    default:
        printf("Wrong Choice: please see the options\n");
    }
  }
  getch();
}
```

OUTPUT:

Queue using Array

1. Insertion
2. Deletion
3. Display
4. Exit

Enter the Choice: 1

Enter no 1: 1

Enter the Choice: 2

Deleted Element is 1

Enter the Choice: 3

Queue Elements are:

Queue is Empty

Enter the Choice:

5. Write a program that implements the following sorting methods to sort a given list of integers in ascending order

i) Bubble sort ii) Selection sort iii) Quick Sort

Bubble sort

```
#include <stdio.h>

void swap(int* xp, int* yp) { int
    temp = *xp;
    *xp = *yp;
    *yp = temp;
}

void bubbleSort(int arr[], int n) { int
    i, j;
    for (i = 0; i < n - 1; i++) {
for (j = 0; j < n - i - 1; j++) {
if (arr[j] > arr[j + 1]) {
swap(&arr[j], &arr[j + 1]);
            }
        }
    }
}

void printArray(int arr[], int size) { int i;
    for (i = 0; i < size; i++) { printf("%d
        ", arr[i]);
    }
    printf("\n");
}

int main() { int
    n;
    printf("Enter the number of elements: ");
    scanf("%d", &n);

    int arr[n];
    printf("Enter %d elements: ", n); int
    i;
    for (i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    bubbleSort(arr, n);
```

```
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}
```

OUTPUT:

INPUT:
arr = [64, 34, 25, 12, 22, 11, 90]

OUTPUT:
Sorted array is: [11, 12, 22, 25, 34, 64, 90]

Selection sort

```
#include <stdio.h>

void selectionSort(int arr[], int n) { int i,
    j, min_idx;

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

        for (j = i + 1; j < n; j++) {
            if (arr[j] < arr[min_idx]) {
                min_idx = j;
            }
        }

        if (min_idx != i) {
            int temp = arr[min_idx];
            arr[min_idx] = arr[i]; arr[i]
            = temp;
        }
    }
}

void printArray(int arr[], int size) { int i;
    for (i = 0; i < size; i++) { printf("%d
        ", arr[i]);
    }
    printf("\n");
}

int main() { int
    n;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
```

```
    int arr[n];
    printf("Enter %d elements: ", n); int
    i;
    for (i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    selectionSort(arr, n); printf("Sorted
    array: \n"); printArray(arr, n);
    return 0;
}
```

OUTPUT:

INPUT:

arr = [64, 25, 12, 22, 11]

OUTPUT:

Sorted array is: [11, 12, 22, 25, 64]

Quick Sort

```
#include <stdio.h>

int partition(int arr[], int low, int high) { int
    pivot = arr[high];
    int i = low - 1; int
    j;

    for (j = low; j < high; j++) { if
        (arr[j] < pivot) {
            i++;
            int temp = arr[i];
            arr[i] = arr[j]; arr[j]
            = temp;
        }
    }

    int temp = arr[i + 1];
    arr[i + 1] = arr[high];
    arr[high] = temp;

    return i + 1;
}

void quickSort(int arr[], int low, int high) { if
    (low < high) {
        int pivot = partition(arr, low, high);

        quickSort(arr, low, pivot - 1);
```

```
        quickSort(arr, pivot + 1, high);
    }
}

void printArray(int arr[], int size) { int i;
    for (i = 0; i < size; i++) { printf("%d
        ", arr[i]);
    }
    printf("\n");
}

int main() { int
    n;
    printf("Enter the number of elements: ");
    scanf("%d", &n);

    int arr[n];
    printf("Enter %d elements: ", n); int
    i;
    for (i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    quickSort(arr, 0, n - 1);
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}
```

OUTPUT:

INPUT:
arr = [33, 10, 55, 71, 29, 3, 18]
OUTPUT:
Sorted array is: [3, 10, 18, 29, 33, 55, 71]

6. a) Write a program to sort a given set of integers using merge sort.

```
#include <stdio.h>

// Function to merge two sorted subarrays
void merge(int arr[], int left, int mid, int right) { int n1
    = mid - left + 1;
    int n2 = right - mid;

    // Create temporary arrays int
    leftArr[n1], rightArr[n2];

    // Copy data to temporary arrays int i;
    for (i = 0; i < n1; i++) {
        leftArr[i] = arr[left + i];
```

```
    }
    int j;
    for (j = 0; j < n2; j++) { rightArr[j] =
        arr[mid + 1 + j];
    }

    // Merge the temporary arrays
    i = 0;
    j = 0;
    int k = left;
    while (i < n1 && j < n2) {
        if (leftArr[i] <= rightArr[j]) {
            arr[k] = leftArr[i];
i++;
} else {
            arr[k] = rightArr[j];
            j++;
        }
        k++;
    }

    // Copy remaining elements of leftArr[], if any while
    (i < n1) {
        arr[k] = leftArr[i]; i++;
        k++;
    }

    // Copy remaining elements of rightArr[], if any while
    (j < n2) {
        arr[k] = rightArr[j];
        j++;
        k++;
    }
}

// Function to perform merge sort
void mergeSort(int arr[], int left, int right) { if
    (left < right) {
        int mid = left + (right - left) / 2;

        // Sort first and second halves
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);

        // Merge the sorted halves
        merge(arr, left, mid, right);
    }
}

// Function to print an array
void printArray(int arr[], int size) { int i;
    for (i = 0; i < size; i++) {
```

```
        printf("%d ", arr[i]);
    }
    printf("\n");
}

// Driver code
int main() {
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr) / sizeof(arr[0]);

    printf("Original array: \n");
    printArray(arr, n);

    mergeSort(arr, 0, n - 1);

    printf("Sorted array: \n");
    printArray(arr, n);

    return 0;
}
```

OUTPUT:

INPUT:
[38, 27, 43, 3, 9, 82, 10]
OUTPUT:
Sorted array is: [3, 9, 10, 27, 38, 43, 82]

b) Write a program to read the marks obtained by students in a mathematics examination and store the data using a heap data structure. Find out the maximum and minimum marks obtained by the students.

```
#include <stdio.h>

// Function to heapify a subtree rooted at index void
heapify(int arr[], int n, int i) {
    int largest = i;
    int left = 2 * i + 1; int
    right = 2 * i + 2;

    if (left < n && arr[left] > arr[largest]) {
        largest = left;
    }

    if (right < n && arr[right] > arr[largest]) { largest =
        right;
    }

    if (largest!= i) {
        int temp = arr[i]; arr[i]
        = arr[largest];
```

```
        arr[largest] = temp;

        heapify(arr, n, largest);
    }
}

// Function to build a max heap
void buildMaxHeap(int arr[], int n) { int i;
    for (i = n / 2 - 1; i >= 0; i--) {
        heapify(arr, n, i);
    }
}

// Function to extract the maximum element (root) from the heap int
extractMax(int arr[], int n) {
    int max = arr[0];
    arr[0] = arr[n - 1]; n-
    -;

    heapify(arr, n, 0);

    return max;
}

// Function to extract the minimum element from the heap int
extractMin(int arr[], int n) {
    int min = arr[n - 1];
    return min;
}

// Driver code
int main() {
    int marks[] = {85, 90, 78, 92, 88, 76, 95};
    int n = sizeof(marks) / sizeof(marks[0]);

    buildMaxHeap(marks, n);

    printf("Maximum marks: %d\n", extractMax(marks, n));
    printf("Minimum marks: %d\n", extractMin(marks, n - 1));

    return 0;
}
```

OUTPUT:

INPUT:
Enter number of students: 5
Enter the marks obtained by students:

78
92
67
85
90

OUTPUT:

Minimum mark obtained: 67

Maximum mark obtained: 92

7. Write a program that use both recursive and non-recursive functions to perform the following searching operations for a Key value in a given list of integers:

A)

```
#include <stdio.h>

int main() {
   int  array[100];
   int search;
   int  c;
   int n;

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

   printf("Enter %d integer(s)\n", n); for

   (c = 0; c < n; c++) {
      scanf("%d", &array[c]);
   }

   printf("Enter a number to search\n");
   scanf("%d", &search);
   int found = 0;
   for (c = 0; c < n; c++) {
      if (array[c] == search) { /* If required element is found */
         printf("%d is present at location %d.\n", search, c + 1); found
         = 1;
         break;
      }}
   if (!found) {
      printf("%d isn't present in the array.\n", search)

      return 0;
}
```

OUTPUT:

INPUT:
arr = [2, 5, 8, 12, 16, 23, 38, 56, 72, 91]
key = 23
OUTPUT:
Recursive Linear Search: 5 Recursive Binary Search: 5

B) #include <stdio.h> int

```
main() {
    int c, first, last, middle, n, search; int
    array[100];

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

    printf("Enter %d integers: ", n); for (c

    = 0; c < n; c++) {
        scanf("%d", &array[c]);
    }

    printf("Enter value to find: ");
    scanf("%d", &search);

    first = 0; last
    = n - 1;

    while (first <= last) { middle =
        (first + last) / 2;

        if (array[middle] < search) { first
            = middle + 1;
        } else if (array[middle] == search) {
            printf("%d found at location %d.\n", search, middle + 1); break;
        } else {
            last = middle - 1;
        }
    }

    if (first > last) {
        printf("Not found! %d isn't present in the list.\n", search);
    }

    return 0; }
```

OUTPUT:

INPUT:
arr = [2, 5, 8, 12, 16, 23, 38, 56, 72, 91]
key = 23
OUTPUT:
Non-recursive Linear Search: 5
Non-recursive Binary Search: 5

8. Write a program to implement the tree traversal methods.

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

struct node {
  int data;

  struct node *leftChild; struct
  node *rightChild;
};

struct node *root = NULL; void

insert(int data) {
  struct node *tempNode = (struct node*) malloc(sizeof(struct node)); struct
  node *current;
  struct node *parent;

  tempNode->data = data; tempNode-
  >leftChild = NULL; tempNode-
  >rightChild = NULL;

  //if tree is empty
  if(root == NULL) {
    root = tempNode;
  } else {
    current = root; parent
    = NULL;

    while(1) {
      parent = current;

      //go to left of the tree if(data
      < parent->data) {
        current = current->leftChild;

        //insert to the left if(current
        == NULL) {
          parent->leftChild = tempNode;
          return;
        }
```

```
        } //go to right of the tree else {
          current = current->rightChild;

          //insert to the right if(current
          == NULL) {
            parent->rightChild = tempNode;
            return;
          }
        }
      }
    }
}

struct node* search(int data) { struct
  node *current = root;
  printf("Visiting elements: ");

  while(current->data != data) {
    if(current != NULL)
      printf("%d ",current->data);

    //go to left tree
    if(current->data > data) { current
      = current->leftChild;
    }
    //else go to right tree
    else {
      current = current->rightChild;
    }

    //not found
    if(current == NULL) {
      return NULL;
    }
  }

  return current;
}

void pre_order_traversal(struct node* root) { if(root !=
  NULL) {
    printf("%d ",root->data);
    pre_order_traversal(root->leftChild);
    pre_order_traversal(root->rightChild);
  }
}

void inorder_traversal(struct node* root) {
  if(root != NULL) {
    inorder_traversal(root->leftChild);
    printf("%d ",root->data);
    inorder_traversal(root->rightChild);
  }
```

```
}

void post_order_traversal(struct node* root) { if(root
  != NULL) {
    post_order_traversal(root->leftChild);
    post_order_traversal(root->rightChild);
    printf("%d ", root->data);
  }
}

int main() { int
  i;
  int array[7] = { 27, 14, 35, 10, 19, 31, 42 };

  for(i = 0; i < 7; i++) insert(array[i]);

  i = 31;
  struct node * temp = search(i);

  if(temp != NULL) {
    printf("[%d] Element found.", temp->data);
    printf("\n");
  }else {
    printf("[ x ] Element not found (%d).\n", i);
  }

  i = 15;
  temp = search(i);

  if(temp != NULL) {
    printf("[%d] Element found.", temp->data);
    printf("\n");
  }else {
    printf("[ x ] Element not found (%d).\n", i);
  }

  printf("\nPreorder traversal: ");
  pre_order_traversal(root);

  printf("\nInorder traversal: ");
  inorder_traversal(root);

  printf("\nPost order traversal: ");
  post_order_traversal(root);

  return 0;
}
```

OUTPUT:

```
Inorder traversal:
10 14 19 27 31 35 42
Preorder traversal:
```

27 14 10 19 35 31 42
Postorder traversal:
10 19 14 31 42 35 27

9. There are train paths between cities. If there is a train between city A and city B then there is a route between the cities. The cost of the route is the distance between city A and city B. Represent the train travel route information as a graph. The node can be represented by the name of the city. Write a program to perform the following operations.

1. **Traverse the graph and display the details of all trains between the cities along with the cost using breadth-first method.**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_CITIES 10
#define MAX_NAME_LENGTH 50

// Structure to represent a train route (edge)
typedef struct TrainRoute {
    int destination;
    int cost; // distance between the cities struct
    TrainRoute* next;
} TrainRoute;

// Structure to represent a graph (adjacency list) typedef
struct Graph {
    char cities[MAX_CITIES][MAX_NAME_LENGTH]; // City names TrainRoute*
    adjList[MAX_CITIES]; // Array of adjacency lists for cities int numCities; //
    Number of cities in the graph
} Graph;

// Queue structure for BFS
typedef struct Queue {
    int items[MAX_CITIES];
    int front, rear;
} Queue;

// Function prototypes
void initGraph(Graph* graph);
void addRoute(Graph* graph, char* cityA, char* cityB, int cost); int
getCityIndex(Graph* graph, char* cityName);
void bfs(Graph* graph, int startIndex);
void dfs(Graph* graph, int startIndex, int visited[]); void
enqueue(Queue* q, int value);
int dequeue(Queue* q);
int isQueueEmpty(Queue* q);

int main() {
    Graph graph;
    initGraph(&graph);
```

```
    // Example adding some routes (cities and their distances)
    addRoute(&graph, "CityA", "CityB", 100); addRoute(&graph,
    "CityA", "CityC", 200); addRoute(&graph, "CityB", "CityC",
    150);      addRoute(&graph,      "CityB",      "CityD",      300);
    addRoute(&graph, "CityC", "CityD", 50);

    // Display all train routes using BFS
    printf("BFS Traversal:\n"); bfs(&graph,
    0);

    // Display all train routes using DFS
    printf("\nDFS Traversal:\n");
    int visited[MAX_CITIES] = {0};
    dfs(&graph, 0, visited);

    return 0;
}

void initGraph(Graph* graph) {
    graph->numCities = 0;
    int i;
    for (i = 0; i < MAX_CITIES; i++) {
        graph->adjList[i] = NULL;
    }
}

void addRoute(Graph* graph, char* cityA, char* cityB, int cost) { int
    indexA = getCityIndex(graph, cityA);
    int indexB = getCityIndex(graph, cityB);

    if (indexA == -1) {
        strcpy(graph->cities[graph->numCities],       cityA);
        indexA = graph->numCities++;
    }

    if (indexB == -1) {
        strcpy(graph->cities[graph->numCities], cityB); indexB =
        graph->numCities++;
    }

    // Add route from cityA to cityB
    TrainRoute* newRouteA = (TrainRoute*)malloc(sizeof(TrainRoute));
    newRouteA->destination = indexB;
    newRouteA->cost = cost;
    newRouteA->next = graph->adjList[indexA];
    graph->adjList[indexA] = newRouteA;

    // Add route from cityB to cityA (since train routes are bidirectional)
    TrainRoute* newRouteB = (TrainRoute*)malloc(sizeof(TrainRoute));
    newRouteB->destination = indexA;
    newRouteB->cost = cost;
    newRouteB->next = graph->adjList[indexB];
```

```
    graph->adjList[indexB] = newRouteB;
}

int getCityIndex(Graph* graph, char* cityName) { int i;
    for (i = 0; i < graph->numCities; i++) {
        if (strcmp(graph->cities[i], cityName) == 0) { return
            i;
        }
    }
    return -1; // City not found
}

void bfs(Graph* graph, int startIndex) { Queue
    q;
    q.front = q.rear = -1;
    int visited[MAX_CITIES] = {0};

    enqueue(&q,     startIndex);
    visited[startIndex] = 1;

    while (!isQueueEmpty(&q)) { int
        current = dequeue(&q);
        printf("City:       %s\n",       graph->cities[current]);
        TrainRoute* route = graph->adjList[current]; while
        (route != NULL) {
            if (!visited[route->destination]) {
                printf("Train: %s -> %s, Cost: %d\n", graph->cities[current], graph->cities[route-
>destination], route->cost);
                enqueue(&q,      route->destination);
                visited[route->destination] = 1;
            }
            route = route->next;
        }
    }
}

void    dfs(Graph*    graph,    int    startIndex,    int    visited[])    {
    visited[startIndex] = 1;
    printf("City:       %s\n",       graph->cities[startIndex]);
    TrainRoute* route = graph->adjList[startIndex]; while
    (route != NULL) {
        if (!visited[route->destination]) {
            printf("Train: %s -> %s, Cost: %d\n", graph->cities[startIndex], graph->cities[route-
>destination], route->cost);
            dfs(graph, route->destination, visited);
        }
        route = route->next;
    }
}

void enqueue(Queue* q, int value) { if (q-
    >rear == MAX_CITIES - 1) {
        printf("Queue overflow!\n");
```

```
        return;
    }
    if (q->front == -1) q->front = 0; q-
    >items[++q->rear] = value;
}

int dequeue(Queue* q) { if
    (q->front == -1) {
        printf("Queue     underflow!\n");
        return -1;
    }
    int item = q->items[q->front]; if
    (q->front == q->rear) {
        q->front = q->rear = -1;
    } else {
        q->front++;
    }
    return item;
}

int isQueueEmpty(Queue* q) { return
    q->front == -1;
}
```

OUTPUT:

BFS traversal starting from CityA:

Train from CityA to CityB with cost 10
Train from CityA to CityC with cost 15
Train from CityB to CityA with cost 10
Train from CityB to CityD with cost 12
Train from CityC to CityA with cost 15
Train from CityC to CityE with cost 10
Train from CityD to CityB with cost 12
Train from CityD to CityE with cost 2
Train from CityD to CityF with cost 1
Train fromCityE to CityC with cost 10
Train from CityE to CityD with cost 2
Train from CityE to CityF with cost 5
Train from CityF to CityD with cost 1
Train from CityF to CityE with cost 5