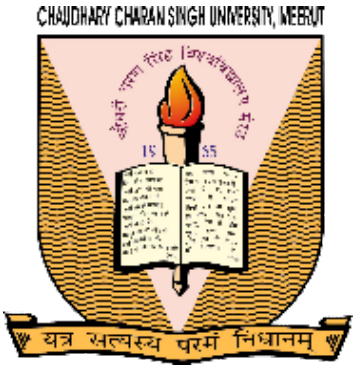
SIR CHHOTU RAM INSTITUTE OF ENGINEERING & TECHNOLOGY



Practical File

on

Data Structures

Branch – Computer Science / 3rd Semester

Session: 2023 - 2024

Submitted By:

Student's Name

Roll no. 100200300

Submitted To:

Teacher's Name

Table of Contents

S.no	Content
1.	Array Operations
2.	Stack Operations
3.	Queue Operations
4.	Linked List Insertion
5.	Linked List Deletion
6.	Searching (Linear & Binary)
7.	Bubble Sort
8.	Quick Sort
9.	Insertion Sort
10.	Selection Sort

Array Operations

1. **WAP to perform the following operations on Array.**
    a. **Traversing**
    b. **Insertion**
    c. **Deletion**

Program Source Code:

```
#include <stdio.h>

#define MAX_SIZE 100

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

int insertElement(int arr[], int size, int element, int position) {
    if (size >= MAX_SIZE) {
        printf("Array is full. Cannot insert element.\n");
        return size;
    }

    if (position < 0 || position > size) {
        printf("Invalid position. Cannot insert element.\n");
        return size;
    }

    for (int i = size - 1; i >= position; i--) {
        arr[i + 1] = arr[i];
    }

    arr[position] = element;

    return size + 1;
}

int deleteElement(int arr[], int size, int position) {
    if (size <= 0) {
        printf("Array is empty. Cannot delete element.\n");
        return size;
    }

    if (position < 0 || position >= size) {
        printf("Invalid position. Cannot delete element.\n");
        return size;
    }
```

```
    int deletedElement = arr[position];

    for (int i = position; i < size - 1; i++) {
        arr[i] = arr[i + 1];
    }

    return size - 1;
}

int main() {
    int arr[MAX_SIZE];
    int size = 0;
    int choice, element, position;

    traverseArray(arr, size);

    printf("Array Operations:\n");
    printf("1. Traverse\n");
    printf("2. Insert\n");
    printf("3. Delete\n");
    printf("4. Exit\n");

    do {
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                traverseArray(arr, size);
                break;
            case 2:
                printf("Enter the element to insert: ");
                scanf("%d", &element);
                printf("Enter the position to insert: scanf("%d",
                &position);
                size = insertElement(arr, size, element, position);
                break;
            case 3:
                printf("Enter the position to delete: ");
               scanf("%d", &position);
                size = deleteElement(arr, size, position);
                break;
            case 4:
                printf("Bye Bye 👋\n");
                break;
            default:
                printf("That option Don't Exist 😳\n");
        }
        if (choice!=1 && choice!= 4){
            traverseArray(arr, size);
        }

    } while (choice != 4);
```

```
    return 0;
}
```

Output:

Array elements:

Array Operations:
1. Traverse
2. Insert
3. Delete
4. Exit
Enter your choice: 2
Enter the element to insert: 12
Enter the position to insert: 0
Array elements: 12

Enter your choice: 2
Enter the element to insert: 31
Enter the position to insert: 1
Array elements: 12 31

Enter your choice: 2
Enter the element to insert: 2
Enter the position to insert: 2
Array elements: 12 31 2

Enter your choice: 3
Enter the position to delete: 0
Array elements: 31 2

Enter your choice: 1
Array elements: 31 2

Enter your choice: 4
Bye Bye 👋

Stack Operations

2. WAP to perform the following operations on Stack.

a. Push
b. Pop

Program Source Code:

```
#include <stdio.h>
#define MAX_SIZE 100

int stack[MAX_SIZE];
int top = -1;

void push(int value) {
    if (top == MAX_SIZE - 1) {
        printf("Stack Overflow\n");
    } else {
        top++;
        stack[top] = value;
        printf("Pushed %d onto the stack\n\n", value);
    }
}

void pop() {
    if (top == -1) {
        printf("Stack Underflow\n");
    } else {
        int value = stack[top];
        top--;
        printf("Popped %d from the stack\n\n", value);
    }
}

int main() {
    int choice, value;
        printf("Stack Operations\n");
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. Exit\n");
    do {

        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the value to push: ");
                scanf("%d", &value);
                push(value);
                break;
```

```
            case 2:
                pop();
                break;
            case 3:
                printf("Bye Bye 👋\n");
                break;
            default:
                printf("That option Don't Exist 😳\n");
        }
    } while (choice != 3);

    return 0;
}
```

Output:

Stack Operations
1. Push
2. Pop
3. Exit
Enter your choice: 1
Enter the value to push: 1
Pushed 1 onto the stack

Enter your choice: 1
Enter the value to push: 2
Pushed 2 onto the stack

Enter your choice: 1
Enter the value to push: 4
Pushed 4 onto the stack

Enter your choice: 2
Popped 4 from the stack

Enter your choice: 2
Popped 2 from the stack

Enter your choice: 2
Popped 1 from the stack

Enter your choice: 2
Stack Underflow
Enter your choice: 3
Bye Bye 👋

Queue Operations

3. WAP to perform the following operations on Queue.

a. Enqueue (Insertion)
b. Dequeue (Deletion)

Program Source Code:

```
#include <stdio.h>
#define MAX_SIZE 4

int queue[MAX_SIZE];
int front = -1;
int rear = -1;

void enqueue(int value) {
    if ((front == 0 && rear == MAX_SIZE - 1) || (rear == (front - 1)
% (MAX_SIZE - 1))) {
        printf("Queue Overflow\n");
    } else if (front == -1) {
        front = rear = 0;
        queue[rear] = value;
        printf("Enqueued %d into the queue\n\n", value);
    } else if (rear == MAX_SIZE - 1 && front != 0) {
        rear = 0;
        queue[rear] = value;
        printf("Enqueued %d into the queue\n\n", value);
    } else {
        rear++;
        queue[rear] = value;
        printf("Enqueued %d into the queue\n\n", value);
    }
}

void dequeue() {
    if (front == -1) {
        printf("Queue Underflow\n");
    } else if (front == rear) {
        int value = queue[front];
        front = rear = -1;
        printf("Dequeued %d from the queue\n\n", value);
    } else if (front == MAX_SIZE - 1) {
        int value = queue[front];
        front = 0;
        printf("Dequeued %d from the queue\n\n", value);
    } else {
        int value = queue[front];
        front++;
        printf("Dequeued %d from the queue\n\n", value);
    }
}
```

```
int main() {
    int choice, value;
    printf("Queue Operations\n");
    printf("1. Enqueue\n");
    printf("2. Dequeue\n");
    printf("3. Exit\n");
    do {
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter the value to enqueue: ");
                scanf("%d", &value);
                enqueue(value);
                break;
            case 2:
                dequeue();
                break;
            case 3:
                printf("Bye Bye 👋\n");
                break;
            default:
                printf("That option doesn't exist 😳\n");
        }
    } while (choice != 3);
    return 0;
}
```

Output:

Queue Operations
1. Enqueue
2. Dequeue
3. Exit
Enter your choice: 1
Enter the value to enqueue: 2
Enqueued 2 into the queue

Enter your choice: 1
Enter the value to enqueue: 3
Enqueued 3 into the queue

Enter your choice: 1
Enter the value to enqueue: 5
Enqueued 5 into the queue

Enter your choice: 1
Enter the value to enqueue: 6
Enqueued 6 into the queue

Enter your choice: 1

Enter the value to enqueue: 7
Queue Overflow
Enter your choice: 2
Dequeued 2 from the queue

Enter your choice: 2
Dequeued 3 from the queue

Enter your choice: 1
Enter the value to enqueue: 4
Enqueued 4 into the queue

Enter your choice: 3
Bye Bye 👋

Linked List Insertion Operation

4. WAP to perform the Insertion operation on Linked List.

a. At the Beginning
b. At the End
c. After a specified node (value)

Program Source Code:

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

struct Node {
    int data;
    struct Node* next;
};

struct Node* createNode(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct
Node));
    if (newNode == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }
    newNode->data = value;
    newNode->next = NULL;
    return newNode;
}

void insertAtBeginning(struct Node** head, int value) {
    struct Node* newNode = createNode(value);
    newNode->next = *head;
    *head = newNode;
}

void insertAtEnd(struct Node** head, int value) {
    struct Node* newNode = createNode(value);
    if (*head == NULL) {
        *head = newNode;
    } else {
        struct Node* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}

void insertAfterNode(struct Node* prevNode, int value) {
```

```
    if (prevNode == NULL) {
        printf("Previous node cannot be NULL\n");
        return;
    }
    struct Node* newNode = createNode(value);
    newNode->next = prevNode->next;
    prevNode->next = newNode;
}

void printLinkedList(struct Node* head) {
    struct Node* temp = head;
    printf("Linked List Elements: ");
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    struct Node* head = NULL;
    int choice, value;
    printf("Linked List Insertion Operations\n");
    printf("1. Insert at Beginning\n");
    printf("2. Insert at End\n");
    printf("3. Insert after a specified node\n");
    printf("4. Exit\n");
    do {
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter the value to insert at the beginning:
");
                scanf("%d", &value);
                insertAtBeginning(&head, value);
                break;
            case 2:
                printf("Enter the value to insert at the end: ");
                scanf("%d", &value);
                insertAtEnd(&head, value);
                break;
            case 3:
                printf("Enter the value to insert: ");
                scanf("%d", &value);
                printf("Enter the value of the node after which to
insert: ");
                int prevValue;
                scanf("%d", &prevValue);
                struct Node* temp = head;
                while (temp != NULL && temp->data != prevValue) {
                    temp = temp->next;
                }
```

```
                if (temp == NULL) {
                    printf("Node with value %d not found\n",
prevValue);
                } else {
                    insertAfterNode(temp, value);
                }
                break;
            case 4:
                printf("Bye Bye 👋\n");
                break;
            default:
                printf("That option doesn't exist 😳\n");

        }
        if (choice!= 4){
            printLinkedList(head);
        }
    } while (choice != 4);
    return 0;
}
```

Output:

Linked List Insertion Operations
1. Insert at Beginning
2. Insert at End
3. Insert after a specified node
4. Exit
Enter your choice: 1
Enter the value to insert at the beginning: 23
Linked List Elements: 23
Enter your choice: 1
Enter the value to insert at the beginning: 34
Linked List Elements: 34 23
Enter your choice: 2
Enter the value to insert at the end: 45
Linked List Elements: 34 23 45
Enter your choice: 3
Enter the value to insert: 12
Enter the value of the node after which to insert: 18
Node with value 18 not found
Linked List Elements: 34 23 45
Enter your choice: 3
Enter the value to insert: 22
Enter the value of the node after which to insert: 23
Linked List Elements: 34 23 22 45
Enter your choice: 4
Bye Bye 👋

Linked List Deletion Operation

5. WAP to perform the Deletion operation on Linked List.

a. At the Beginning
b. At the End
c. After a specified node (value)

Program Source Code:

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

struct Node {
    int data;
    struct Node* next;
};

struct Node* createNode(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct
Node));
    if (newNode == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }
    newNode->data = value;
    newNode->next = NULL;
    return newNode;
}

void insertAtBeginning(struct Node** head, int value) {
    struct Node* newNode = createNode(value);
    newNode->next = *head;
    *head = newNode;
}

void deleteAtBeginning(struct Node** head) {
    if (*head == NULL) {
        printf("Linked list is empty\n");
        return;
    }
    struct Node* temp = *head;
    *head = (*head)->next;
    free(temp);
}

void deleteAtEnd(struct Node** head) {
    if (*head == NULL) {
        printf("Linked list is empty\n");
        return;
    }
```

```
    if ((*head)->next == NULL) {
        free(*head);
        *head = NULL;
        return;
    }
    struct Node* temp = *head;
    while (temp->next->next != NULL) {
        temp = temp->next;
    }
    free(temp->next);
    temp->next = NULL;
}

void deleteNode(struct Node** head, int value) {
    if (*head == NULL) {
        printf("Linked list is empty\n");
        return;
    }
    if ((*head)->data == value) {
        struct Node* temp = *head;
        *head = (*head)->next;
        free(temp);
        return;
    }
    struct Node* temp = *head;
    while (temp->next != NULL && temp->next->data != value) {
        temp = temp->next;
    }
    if (temp->next == NULL) {
        printf("Node with value %d not found\n", value);
    } else {
        struct Node* nodeToDelete = temp->next;
        temp->next = temp->next->next;
        free(nodeToDelete);
    }
}

void printLinkedList(struct Node* head) {
    struct Node* temp = head;
    printf("Linked List Elements: ");
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    struct Node* head = NULL;
    int choice, value;

    insertAtBeginning(&head, 10);
    insertAtBeginning(&head, 20);
```

```
    insertAtBeginning(&head, 30);
    insertAtBeginning(&head, 40);
    insertAtBeginning(&head, 40);
    insertAtBeginning(&head, 40);

    printf("Linked List Deletion Operations\n");
    printf("1. Delete at Beginning\n");
    printf("2. Delete at End\n");
    printf("3. Delete a specified node\n");
    printf("4. Exit\n");

    do {
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                deleteAtBeginning(&head);
                break;
            case 2:
                deleteAtEnd(&head);
                break;
            case 3:
                printf("Enter the value of the node to delete: ");
                scanf("%d", &value);
                deleteNode(&head, value);
                break;
            case 4:
                printf("Bye Bye 👋\n");
                break;
            default:
                printf("That option doesn't exist 😳\n");
        }

        if (choice != 4) {
            printLinkedList(head);
        }
    } while (choice != 5);

    return 0;
}
```

Output:

Linked List Deletion Operations
1. Delete at Beginning
2. Delete at End
3. Delete a specified node
4. Exit
Enter your choice: 1
Linked List Elements: 40 40 30 20 10
Enter your choice: 1

Linked List Elements: 40 30 20 10
Enter your choice: 3
Enter the value of the node to delete: 20
Linked List Elements: 40 30 10
Enter your choice: 2
Linked List Elements: 40 30
Enter your choice: 3
Enter the value of the node to delete: 45
Node with value 45 not found
Linked List Elements: 40 30
Enter your choice: 4
Bye Bye 👋

Searching

6. WAP to perform the Searching operation using following algorithms.
   a. Linear Search
   b. Binary Search

Program Source Code:

```
#include <stdio.h>

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

int linearSearch(int arr[], int size, int key) {
    for (int i = 0; i < size; i++) {
        if (arr[i] == key) {
            return i;
        }
    }
    return -1;
}

int binarySearch(int arr[], int size, int key) {
    int low = 0;
    int high = size - 1;

    while (low <= high) {
        int mid = (low + high) / 2;

        if (arr[mid] == key) {
            return mid;
        } else if (arr[mid] < key) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }

    return -1;
}

int main() {
    int arr[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    int size = 10;
    int choice, element;
```

```
    traverseArray(arr, size);

    printf("Array Operations:\n");
    printf("1. Linear Search\n");
    printf("2. Binary Search\n");
    printf("3. Exit\n");

    do {
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the key to search: "); scanf("%d",
&element);
                int linearSearchResult = linearSearch(arr, size,
element);
                if (linearSearchResult == -1) {
                    printf("Element not found.\n");
                } else {
                    printf("Element found at index %d.\n",
linearSearchResult);
                }
                break;
            case 2:
                printf("Enter the key to search: "); scanf("%d",
&element);
                int binarySearchResult = binarySearch(arr, size,
element);
                if (binarySearchResult == -1) {
                    printf("Element not found.\n");
                } else {
                    printf("Element found at index %d.\n",
binarySearchResult);
                }
                break;
            case 3:
                printf("Bye Bye 👋\n");
                break;
            default:
                printf("That option Don't Exist 😳\n");
        }

    } while (choice != 3);

    return 0;
}
```

Output:

Array elements: 1 2 3 4 5 6 7 8 9 10

Array Operations:
1. Linear Search
2. Binary Search
3. Exit
Enter your choice: 1
Enter the key to search: 5
Element found at index 4.
Enter your choice: 2
Enter the key to search: 9
Element found at index 8.
Enter your choice: 1
Enter the key to search: 11
Element not found.
Enter your choice: 2
Enter the key to search: 34
Element not found.
Enter your choice: 3
Bye Bye 👋

Sorting

7. WAP to perform sorting using the Bubble Sort Technique.

Program Source Code:

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

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

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

    int *arr = (int*)malloc(n * sizeof(int));
    if (arr == NULL) {
        printf("Memory allocation failed.");
        return 1;
    }

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

    bubbleSort(arr, n);

    printf("Array After Bubble Sort:\n");
    for (i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }

    return 0;
}
```

Output:

```
Enter the number of elements in the array: 6
Enter the elements of the array:
23
45
7
6
2
2
Array After Bubble Sort:
2 2 6 7 23 45
```

---

8. WAP to perform sorting using the Quick Sort Technique.

Program Source Code:

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

void swap(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = (low - 1);
    for (int j = low; j <= high - 1; j++) {
        if (arr[j] < pivot) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    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);
    }
}
```

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

    int *arr = (int*)malloc(n * sizeof(int));
    if (arr == NULL) {
        printf("Memory allocation failed.");
        return 1;
    }

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

    quickSort(arr, 0, n - 1);

    printf("Array After Quick Sort:\n");
    for (i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }

    return 0;
}
```

Output:

Enter the number of elements in the array: 7
Enter the elements of the array:
23
42
35
67
82
22
21
Array After Quick Sort:
21 2 23 35 42 67 82

9. WAP to perform sorting using the Insertion Sort Technique.

Program Source Code:

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

void insertionSort(int arr[], int n) {
    int i, key, j;
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;

        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}

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

    int *arr = (int*)malloc(n * sizeof(int));
    if (arr == NULL) {
        printf("Memory allocation failed.");
        return 1;
    }

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

    insertionSort(arr, n);

    printf("Array After Insertion Sort:\n");
    for (i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }

    return 0;
}
```

Output:

Enter the number of elements in the array: 7
Enter the elements of the array:
12
32
31
45
45
21
20
Array After Insertion Sort:
12 20 21 31 32 45 45

10.WAP to perform sorting using the Selection Sort Technique.

Program Source Code:

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

void selectionSort(int arr[], int n) {
    int i, j, minIndex, temp;
    for (i = 0; i < n - 1; i++) {
        minIndex = i;
        for (j = i + 1; j < n; j++) {
            if (arr[j] < arr[minIndex]) {
                minIndex = j;
            }
        }
        temp = arr[minIndex];
        arr[minIndex] = arr[i];
        arr[i] = temp;
    }
}

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

    int *arr = (int*)malloc(n * sizeof(int));
    if (arr == NULL) {
        printf("Memory allocation failed.");
        return 1;
    }

    printf("Enter the elements of the array:\n");
```

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

    selectionSort(arr, n);

    printf("Array After Selection Sort:\n");
    for (i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }

    return 0;
}
```

Output:

Enter the number of elements in the array: 7
Enter the elements of the array:
12
32
45
67
35
21
31
Array After Selection Sort:
12 21 31 32 35 45 67