**Data Structures And Algorithms Practical File**

**(IT-663)**

***Submitted in partial fulfillment of the requirements for the award of the degree***

***Of***

**Masters of Computer Application (SE)**

**Submitted to: Submitted By:**

**Dr. Reena Gupta Name : Thakur Rudraksha Singh**

**Associate Professor Enrollment No : 1616404524**

**Semester - I**

****

**University School of Information , Communication & Technology**

**Guru Gobind Singh Indraprastha University**

**Sector-16C , Dwarka New Delhi - 110078**

**(2024-2026)**

**INDEX**

|  |  |  |
| --- | --- | --- |
| **S No.** | **Practical** | **Page No.** |
| 1 | Implement operations (traverse, insert (beg, loc, end), delete (beg, loc, end), linear search, binary search using recursion, insertion sort) on an array. |  |
| 2 | Implement Sparse Array. |  |
| 3 | Implement insertion (at the beginning, at specified location, at the end) and deletion (at the beginning, at specified location, at the end) on single linked list and count the number of nodes & reverse the single linked list. |  |
| 4 | Implement insertion (at the beginning, at specified location, at the end) and deletion (at the beginning, at specified location, at the end) on circular single linked list. |  |
| 5 | Implement insertion (at the beginning, at specified location, at the end) and deletion (at the beginning, at specified location, at the end) on double linked list. |  |
| 6 | Implement insertion (at the beginning, at specified location, at the end) and deletion (at the beginning, at specified location, at the end) on circular double linked list. |  |
| 7 | Write a program to merge two sorted linked list and display the final sorted linked list. |  |
| 8 | Implement operations (push, pop) on a stack using arrays. Check the status of the stack whether there is underflow or overflow. |  |
| 9 | Implement the conversion of infix notation to postfix notation and evaluation of postfix notation using stacks. |  |
| 10 | Implement operations (enqueue, dequeue) on a queue using arrays. Check the status of the queue whether it is empty or full. |  |
| 11 | Implement circular queue using arrays and linked list. |  |
| 12 | Implement stacks and queues using linked list. |  |
| 13 | Implement operations on Binary Search Tree (Insertion, Deletion, Search, Traversals (using recursion)- Inorder, Preorder, Postorder). |  |
| 14 | Implement traversals on Binary Search Tree (using stacks) - Inorder, Preorder, Postorder. |  |
| 15 | Implement AVL tree. |  |
| 16 | Implement graph traversals (DFS & BFS). |  |
| 17 | Implement dijkstra’s shortest path greedy algorithm and Kruskal’s  minimum spanning tree greedy algorithm. |  |
| 18 | Implement Longest Common Subsequence dynamic programming algorithm. |  |
| 19 | Perform various sorting techniques (heap, count, radix, bucket). |  |
| 20 | Perform divide and conquer sorting techniques (merge, quick). |  |

1. **Implement operations (traverse, insert (beg, loc, end), delete (beg, loc, end), linear search, binary search using recursion, insertion sort) on an array.**

#include <iostream>

using namespace std;

void traverse(int arr[], int n){

    for (int i = 0; i < n; i++)

        cout << arr[i] << " ";

    cout << endl;

}

void insertAtBeginning(int arr[], int &n, int value, int capacity){

    if (n >= capacity){

        cout << "Array is full." << endl;

        return;

    }

    for (int i = n; i > 0; i--)

        arr[i] = arr[i - 1];

    arr[0] = value;

    n++;

    cout << "Value " << value << " is inserted at the beginning of the array." << endl;

    cout << "Your updated array is: " << endl;

    traverse(arr, n);

}

void insertAtEnd(int arr[], int &n, int value, int capacity){

    if (n >= capacity){

        cout << "Array is full." << endl;

        return;

    }

    arr[n] = value;

    n++;

    cout << "Value " << value << " is inserted at the end of the array." << endl;

    cout << "Your updated array is: " << endl;

    traverse(arr, n);

}

void insertAtPosition(int arr[], int &n, int value, int position, int capacity){

    if (position < 0 || position > n){

        cout << "Invalid position." << endl;

        return;

    }

    if (n >= capacity){

        cout << "Array is full." << endl;

        return;

    }

    for (int i = n; i > position; i--)

        arr[i] = arr[i - 1];

    arr[position] = value;

    n++;

    cout << "Value " << value << " is inserted at position " << position << "." << endl;

    cout << "Your updated array is: " << endl;

    traverse(arr, n);

}

void deleteFromBeginning(int arr[], int &n){

    if (n <= 0){

        cout << "Array is empty." << endl;

        return;

    }

    for (int i = 0; i < n - 1; i++)

        arr[i] = arr[i + 1];

    n--;

    cout << "Element deleted from the beginning of the array." << endl;

    cout << "Your updated array is: " << endl;

    traverse(arr, n);

}

void deleteFromEnd(int arr[], int &n){

    if (n <= 0){

        cout << "Array is empty." << endl;

        return;

    }

    n--;

    cout << "Element deleted from the end of the array." << endl;

    cout << "Your updated array is: " << endl;

    traverse(arr, n);

}

void deleteFromPosition(int arr[], int &n, int position){

    if (position < 0 || position >= n){

        cout << "Invalid position." << endl;

        return;

    }

    for (int i = position; i < n - 1; i++)

        arr[i] = arr[i + 1];

    n--;

    cout << "Element deleted from position " << position << "." << endl;

    cout << "Your updated array is: " << endl;

    traverse(arr, n);

}

int linearSearch(int arr[], int n, int key){

    for (int i = 0; i < n; i++){

        if (arr[i] == key)

            return i;

    }

    return -1;

}

int binarySearch(int arr[], int left, int right, int key){

    if (right >= left){

        int mid = left + (right - left) / 2;

        if (arr[mid] == key)

            return mid;

        if (arr[mid] > key)

            return binarySearch(arr, left, mid - 1, key);

        return binarySearch(arr, mid + 1, right, key);

    }

    return -1;

}

void insertionSort(int arr[], int n){

    for (int i = 1; i < n; i++)   //j is initialized to i - 1, which points to the last element of the sorted portion.

    {

        int key = arr[i];

        int j = i - 1;

        while (j >= 0 && arr[j] > key){

            arr[j + 1] = arr[j];

            j--;

        }

        arr[j + 1] = key;

    }

}

int main(){

     int capacity = 100; // Set the maximum capacity of the array

    int arr[capacity];

    int n = 0; // Initialize the size of the array to 0

    cout << "Enter the number of elements in your array (max: " << capacity << "): ";

    int numElements;

    cin >> numElements;

    if (numElements > capacity) {

        cout << "Number of elements exceeds capacity. Setting size to " << capacity << "." << endl;

        cout << "as you have given larger than capacity we are taking number of elemts as 5 by default" << endl;

        numElements = 5;

    }

    cout << "Enter the elements of the array: ";

    for (int i = 0; i < numElements; i++) {

        cin >> arr[i];

    }

    n = numElements; // Update the size of the array

    bool continueProgram = true;

    while (continueProgram){

        cout << "\nMenu:\n";

        cout << "1. Traverse Array        ";

        cout << "2. Insert at Beginning     ";

        cout << "3. Insert at End\n";

        cout << "4. Insert at Position    ";

        cout << "5. Delete from Beginning   ";

        cout << "6. Delete from End\n";

        cout << "7. Delete from Position  ";

        cout << "8. Linear Search           ";

        cout << "9. Binary Search (Array should be sorted)\n";

        cout << "10. Insertion Sort\n";

        cout << "Enter your choice: ";

        int choice, value, position;

        cin >> choice;

        switch (choice){

        case 1:

            traverse(arr, n);

            break;

        case 2:

            cout << "Enter value to insert: ";

            cin >> value;

            insertAtBeginning(arr, n, value, capacity);

            break;

        case 3:

            cout << "Enter value to insert: ";

            cin >> value;

            insertAtEnd(arr, n, value, capacity);

            break;

        case 4:

            cout << "Enter value to insert : ";

            cin >> value;

            cout << "Enter position to insert (0-based index) from 0 to "<<n<<" : ";

            cin >> position;

            insertAtPosition(arr, n, value, position, capacity);

            break;

        case 5:

            deleteFromBeginning(arr, n);

            break;

        case 6:

            deleteFromEnd(arr, n);

            break;

        case 7:

            cout << "Enter position to delete (0-based index) 0 to "<<n-1<<": ";

            cin >> position;

            deleteFromPosition(arr, n, position);

            break;

        case 8:

            cout << "Enter value to search: ";

            cin >> value;

            position = linearSearch(arr, n, value);

            if (position != -1)

                cout << "Element found at position: " << position << endl;

            else

                cout << "Element not found." << endl;

            break;

        case 9:

            cout << "Enter value to search: ";

            cin >> value;

            position = binarySearch(arr, 0, n - 1, value);

            if (position != -1)

                cout << "Element found at position : " << position << endl;

            else

                cout << "Element not found." << endl;

            break;

        case 10:

            insertionSort(arr, n);

            cout << "Array sorted successfully." << endl;

            break;

        default:

            cout << "Invalid choice. Try again." << endl;

        }

        // Ask if the user wants to continue

        char cont;

        cout << "\nDo you want to continue? (y/n): ";

        cin >> cont;

        if (cont != 'y' && cont != 'Y'){

            continueProgram = false;

            cout << "Exiting program." << endl;

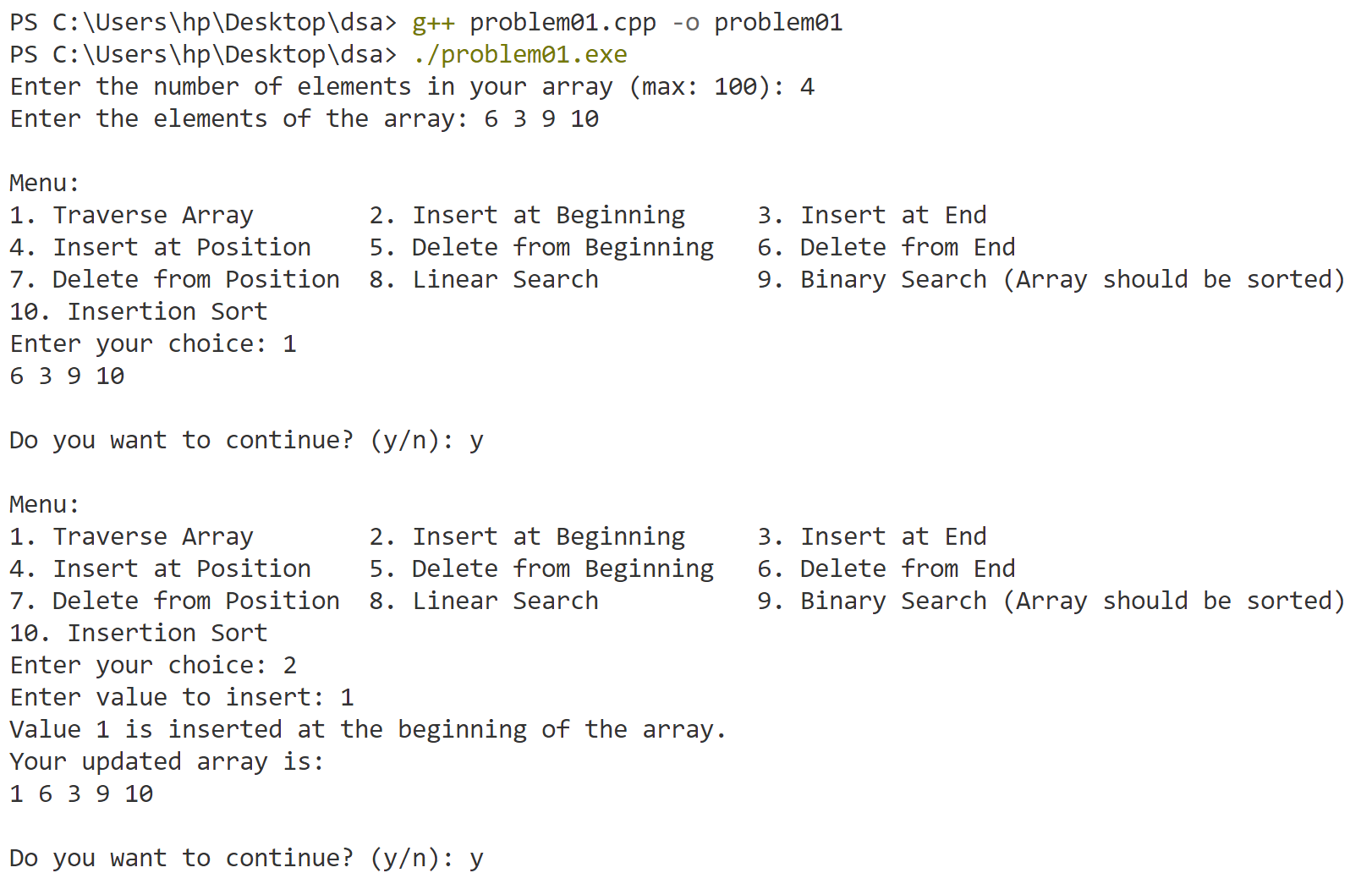
        }

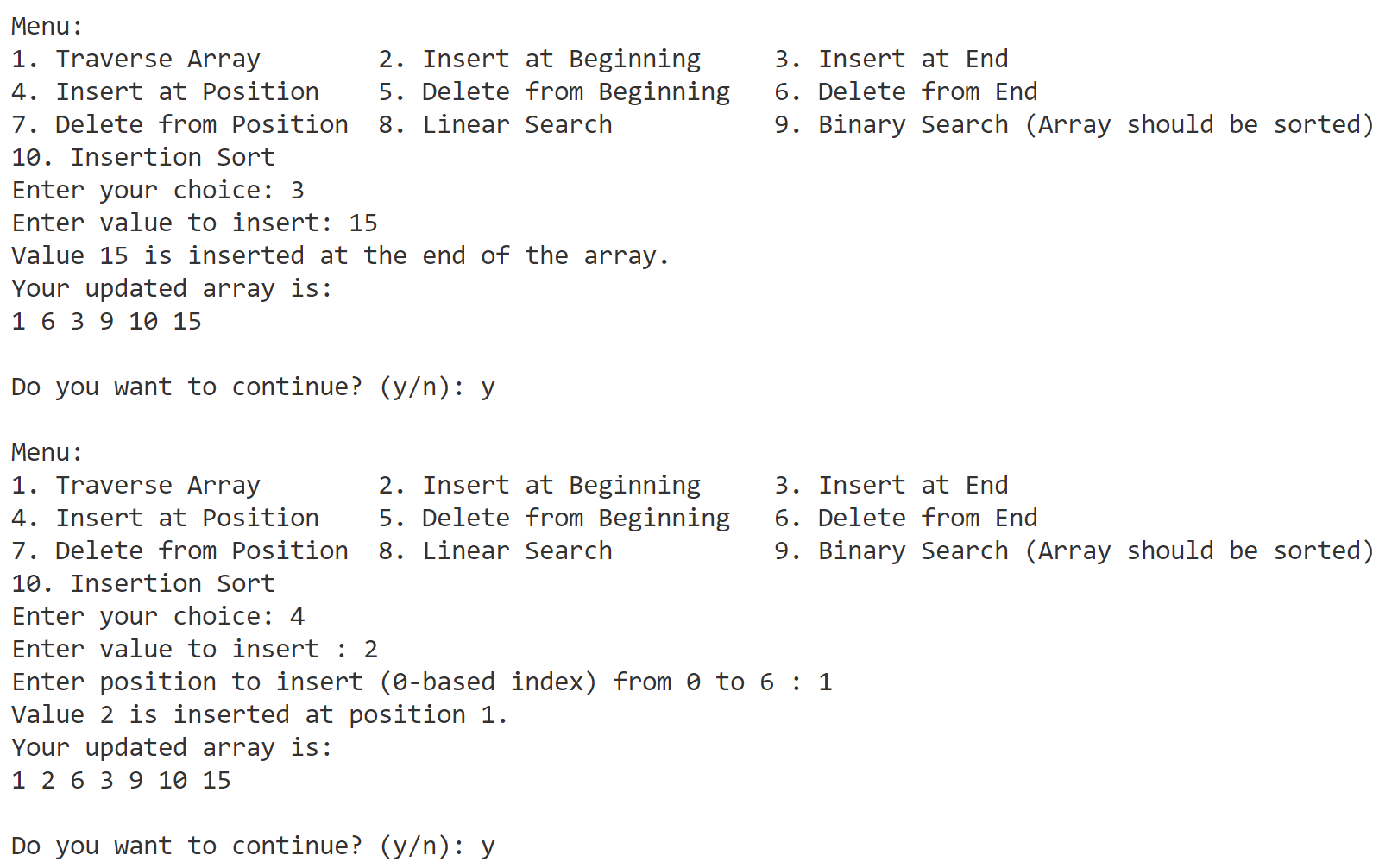
    }

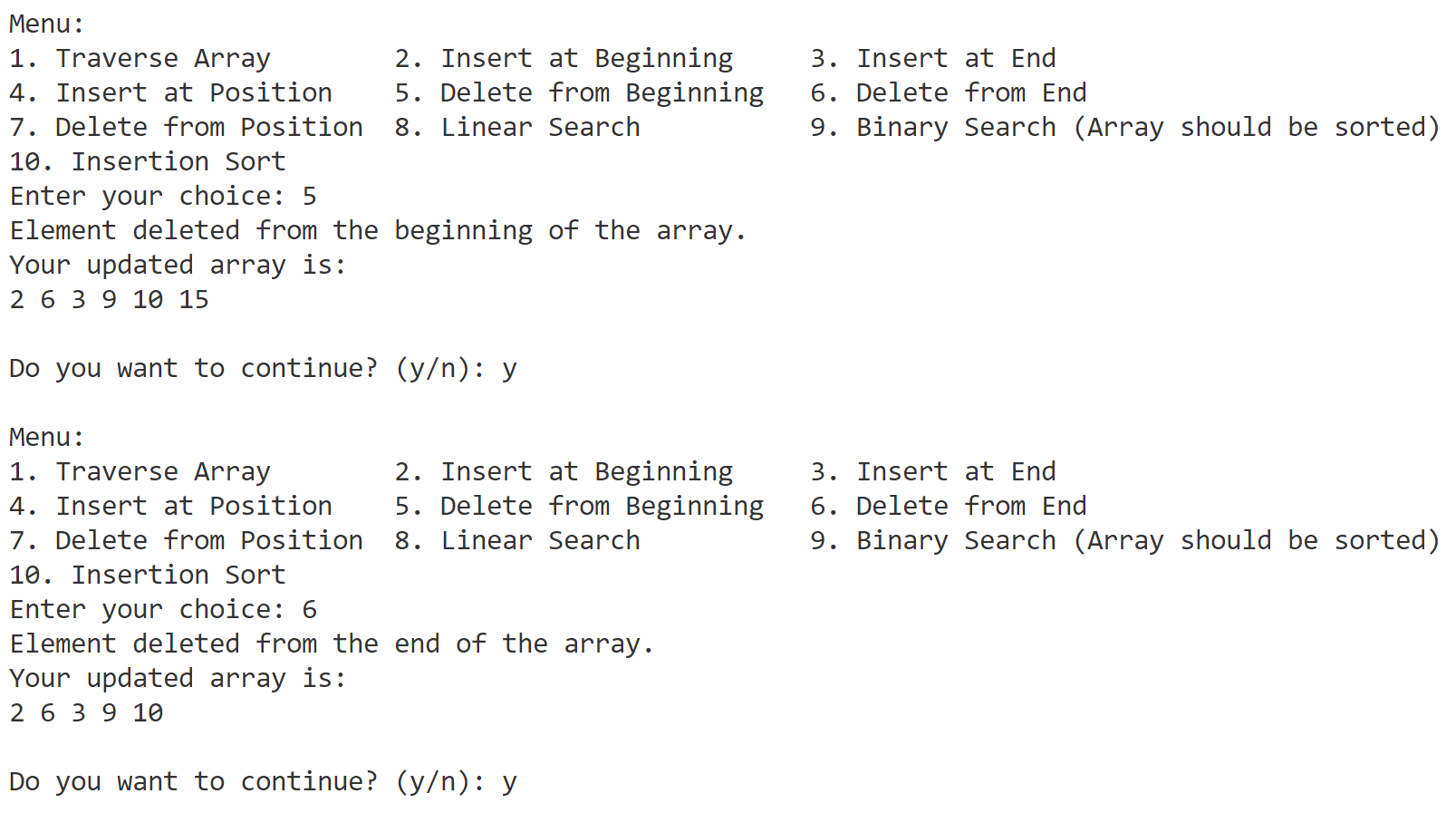
return 0;

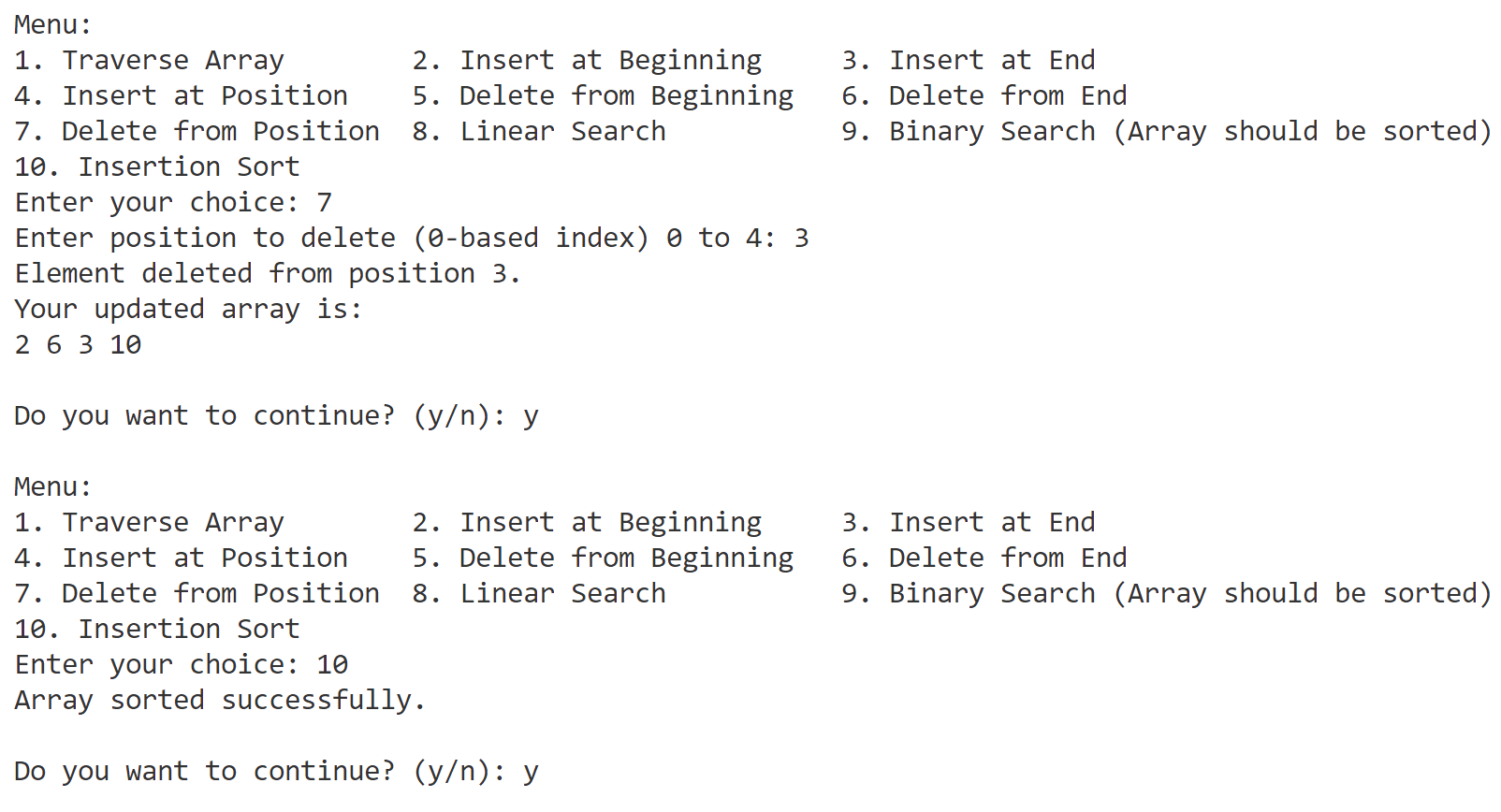
}

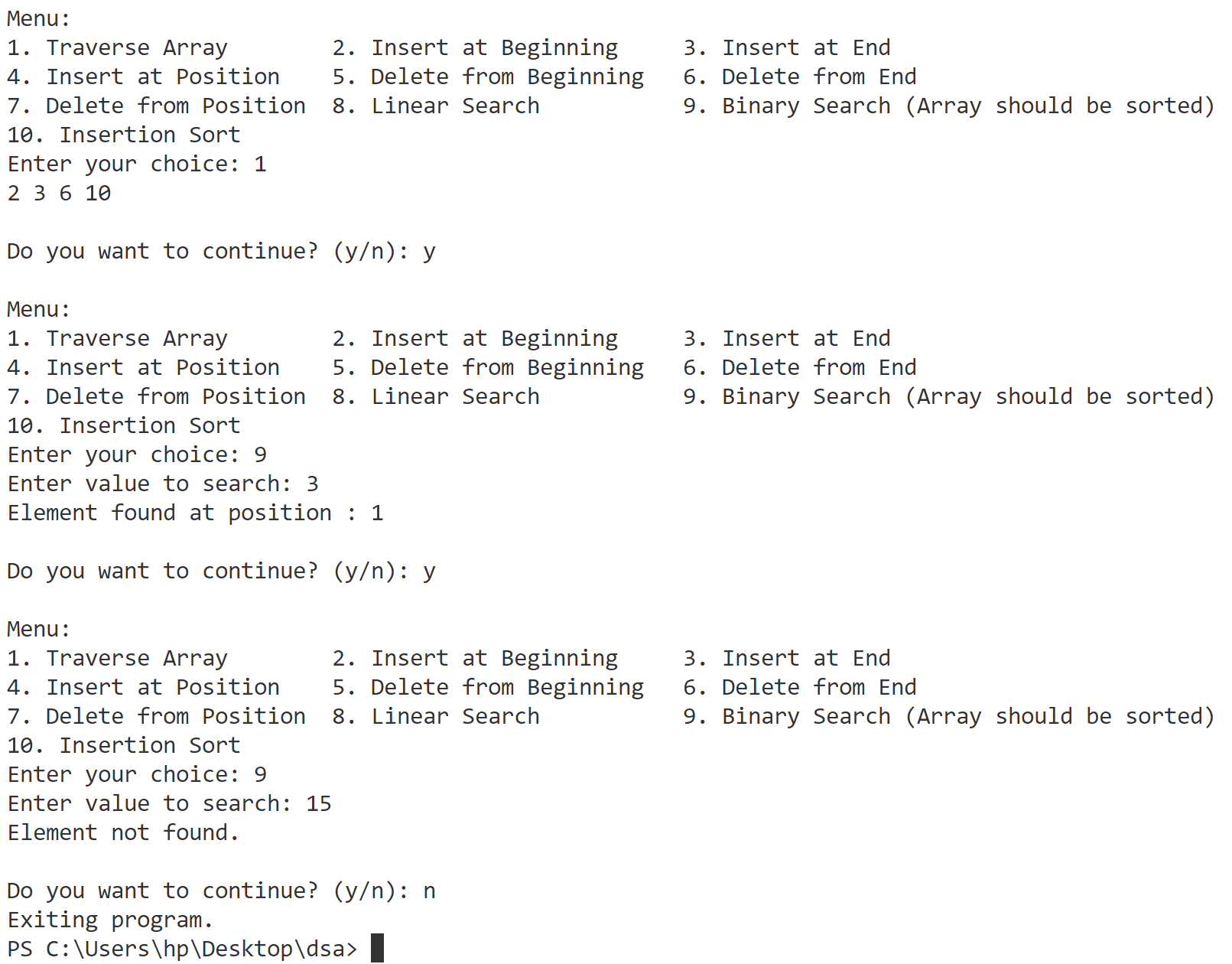
**Output**



****

****

****

****

1. **Implement Sparse Array.**

//sparse matrix

#include <bits/stdc++.h>

using namespace std;

int main() {

    ///////////////////////////Array////////////////////////////////////

    int rows, columns;

    cout << "Enter Rows(max:10) and Columns(max:10) :" << endl;

    cin >> rows >> columns;

    if (rows == 0 && columns == 0) {

        cout << "both rows and columns cannot be zero" << endl;

        return 0;

    } else if (rows > 10 || columns > 10) {

        cout << "rows and columns cannot be greater than 10" << endl;

        return 0;

    }

    int arr[rows][columns];

    cout << "Enter elements of matrix:" << endl;

    for (int i = 0; i < rows; i++) {

        for (int j = 0; j < columns; j++) {

            cin >> arr[i][j];

        }

    }

    cout << "Input Matrix: " << endl;

    for (int i = 0; i < rows; i++) {

        for (int j = 0; j < columns; j++) {

            cout << arr[i][j] << " ";

        }

        cout << endl;

    }

    ///////////////////////////Sparse Matrix////////////////////////////////////

    int cnt = 0;

    for (int i = 0; i < rows; i++) {

        for (int j = 0; j < columns; j++) {

            if (arr[i][j] == 0)

                cnt++;

        }

    }

    // total elements in array

    int totalElements = rows \* columns;

    if (totalElements - cnt > (totalElements / 4)) {

        cout << "not sparse matrix" << endl;

        cout << "exiting program." << endl;

        return 0;

    }

    cout << "Sparse Matrix" << endl;

    int nonZeroElement = totalElements - cnt;

    int sparseMatrix[nonZeroElement + 1][3];

    int row = 0;

    sparseMatrix[row][0] = rows;

    sparseMatrix[row][1] = columns;

    sparseMatrix[row][2] = nonZeroElement;

    row++;

    for (int i = 0; i < rows; i++) {

        for (int j = 0; j < columns; j++) {

            if (arr[i][j] != 0) {

                sparseMatrix[row][0] = i;

                sparseMatrix[row][1] = j;

                sparseMatrix[row][2] = arr[i][j];

                row++;

            }

        }

    }

    for (int i = 0; i < nonZeroElement + 1; i++) {

        for (int j = 0; j < 3; j++) {

            cout << sparseMatrix[i][j] << " ";

        }

        cout << endl;

    }

    cout << "used zero based indexing" << endl;

    return 0;

}

**Output**

****

1. **j**