CHANDIGARH COLLEGE OF ENGINEERING & TECHNOLOGY (DEGREE WING)

Government institute under Chandigarh (UT) Administration, affiliated to Punjab University, Chandigarh

Department of Computer Science & Engineering

Semester: CSE 3rd

SUBJECT: Data Structures Practical (CS351)

<u>Problem 5: Case Study of Stack and Queue</u>

Submitted by:	**Submitted to:**
Bhavyam Dhand	Dr. R.B. Patel
(CO23316)	(Professor)
Date of Practical:23-Sep-24	**Date of Submission:**7-Oct-24

INDEX

S.No	Content	Page no.
I.	Objective	
II.	Discussion & Modelling of Problem	
III.	Implementation of Code	
1.	Linear Search	
2.	Binary Search	
3.	Insertion Sort	
4.	Selection Sort	
5.	Bubble Sort	
6.	Radix Sort	
7.	Quick Sort	
IV.	CODE	
V.	Output	
VI.	Log File as Machine Code	

CODE

```
#include <iostream>
#include <string>
#include <fstream>
#include <cmath>
#include <ctime>
#include <bitset>
#include <algorithm> // For std::swap
using namespace std;
// Function to write in Log file
void LogFile(const string& event) {
    ofstream BinFile("Machine_Code_Stack.txt", ios_base::app);
    if (BinFile.is_open()) {
        for (char c : event) {
            BinFile << bitset<8>(c) << " ";
        }
        BinFile << endl;
        BinFile.close();
    }
}
// Function to Create Array
int *CreateArray(ifstream &A, int n) {
    int x, i = 0, *arr = new int[n];
    while (A >> x && i < n) {
        *(arr + i) = x;
        i++;
    }
    return arr;
}
// Function for Linear Search
int LinSearch(int *arr, int n, int x) {
    LogFile("Linear Search initiated for element: " + to_string(x));
    for (int i = 0; i < n; i++) {
        if (*(arr + i) == x) {
            LogFile("Element found at index: " + to_string(i));
            return i;
        }
    }
    LogFile("Element not found");
    return -1;
}
// Function for Binary Search
int BinSearch(int *arr, int n, int x) {
    LogFile("Binary Search initiated for element: " + to_string(x));
    if (n == 0) {
        cout << "Array Underflow!" << endl;
        return -1;
    } else {
        int low = 0, high = n - 1;
        while (low <= high) {
            int mid = (low + high) / 2;
            if (*(arr + mid) < x) {
```

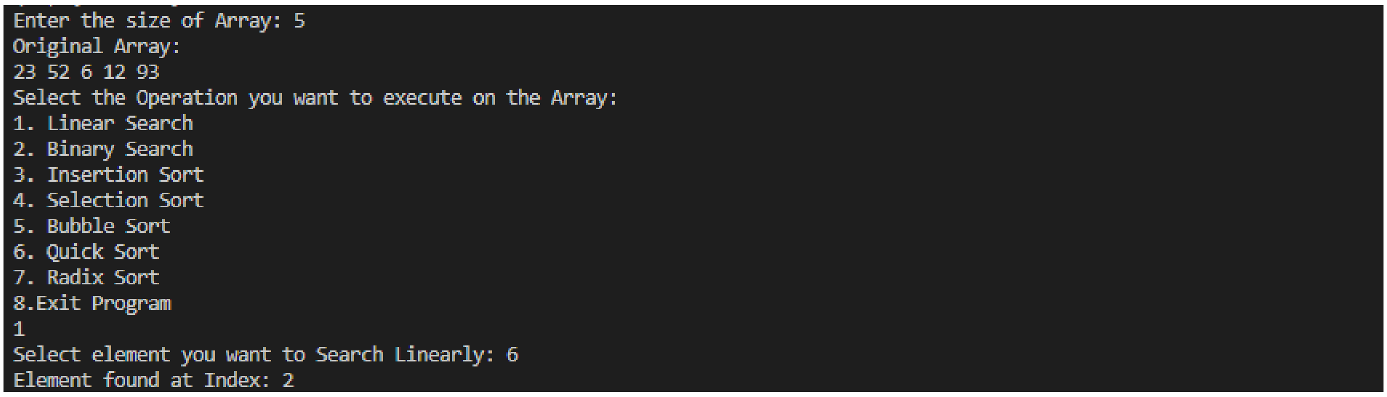
```cpp
                low = mid + 1;
            } else if (*(arr + mid) > x) {
                high = mid - 1;
            } else {
                LogFile("Element found at index: " + to_string(mid));
                return mid;
            }
        }
        LogFile("Element not found");
        cout << "Element not found!" << endl;
        return -1;
    }
}
// Function for Insertion Sort
void InsertionSort(int *arr, int n) {
    LogFile("Insertion Sort initiated");
    for (int i = 1; i < n; i++) {
        int x = *(arr + i);
        int j = i - 1;
        while (j >= 0 && *(arr + j) > x) {
            *(arr + j + 1) = *(arr + j);
            j--;
        }
        *(arr + j + 1) = x;
    }
    LogFile("Array sorted using Insertion Sort");
}
// Function for Selection Sort
void SelectionSort(int *arr, int n) {
    LogFile("Selection Sort initiated");
    for (int i = 0; i < n - 1; i++) {
        int min = i;
        for (int j = i + 1; j < n; j++) {
            if (*(arr + j) < *(arr + min))
                min = j;
        }
        if (min != i) {
            swap(*(arr + i), *(arr + min));
        }
    }
    LogFile("Array sorted using Selection Sort");
}
// Function for Bubble Sort
void BubbleSort(int *arr, int n) {
    LogFile("Bubble Sort initiated");
    for (int i = 0; i < n; i++) {
        bool swapped = false;
        for (int j = 0; j < n - 1 - i; j++) {
            if (*(arr + j) > *(arr + j + 1))
                swap(*(arr + j), *(arr + j + 1));
            swapped = true;
        }
        if (!swapped)
            break;
    }
    LogFile("Array sorted using Bubble Sort");
}
// Function for Quick Sort
void QuickSort(int *arr, int low, int high) {
    if (low < high) {
        LogFile("Quick Sort initiated");
        int pivot = *(arr + high);
        int i = low - 1;
        for (int j = low; j < high; j++) {
            if (*(arr + j) <= pivot) {
                i++;
                swap(*(arr + i), *(arr + j));
            }
        }
        swap(*(arr + i + 1), *(arr + high));
        int pi = i + 1;

        QuickSort(arr, low, pi - 1);
        QuickSort(arr, pi + 1, high);
    }
    LogFile("Array sorted using Quick Sort");
}
// Function for Radix Sort
void CountSort(int *arr, int n, int exp) {
    int *output = new int[n];
    int count[10] = {0};
    for (int i = 0; i < n; i++)
        count[(*(arr + i) / exp) % 10]++;
    for (int i = 1; i < 10; i++)
        count[i] += count[i - 1];
    for (int i = n - 1; i >= 0; i--) {
        output[count[(*(arr + i) / exp) % 10] - 1] = *(arr + i);
        count[(*(arr + i) / exp) % 10]--;
    }
    for (int i = 0; i < n; i++)
        *(arr + i) = output[i];
    delete[] output;
}
void RadixSort(int *arr, int n) {
    LogFile("Radix Sort initiated");
    int max = *max_element(arr, arr + n);
    for (int exp = 1; max / exp > 0; exp *= 10)
        CountSort(arr, n, exp);
    LogFile("Array sorted using Radix Sort");
}
int main() {
    int n, choice, x;
    cout << "Enter the size of Array: ";
    cin >> n;

    ifstream infile("array.txt");
    if (!infile.is_open()) {
        cout << "Error: Unable to open the file!" << endl;
        return -1;
    }
    int *array = CreateArray(infile, n);
    infile.close();  // Close the file after reading the data
    cout << "Original Array:" << endl;
    for (int i = 0; i < n; i++) {
        cout << *(array + i) << " ";
    }
    cout << endl;
    LogFile("Original Array: ");
    for (int i = 0; i < n; i++) {
        LogFile(to_string(*(array + i)) + " ");
    }
    ofstream OutFile("NewArray.txt");
    bool Sorted=false;
    while (Sorted!=true)
    {
        cout << "Select the Operation you want to execute on the Array:" << endl;
        cout << "1. Linear Search\n2. Binary Search\n3. Insertion Sort\n4. Bubble Sort\n5.
Quick Sort\n6. Radix Sort\n7.Exit Program" << endl;
        cin >> choice;
        switch (choice) {
        case 1://Linear Search
            cout << "Select element you want to Search Linearly: "; cin >> x;
            if (LinSearch(array, n, x)!=-1)
                cout << "Element lies in Index: " << LinSearch(array, n, x) << endl;
            break;
        case 2://Binary Search
            cout << "Select element you want to Search Binarily: "; cin >> x;
            if (BinSearch(array, n, x)!=-1)
                cout << "Element lies in Index: " << BinSearch(array, n, x) << endl;
            break;
        case 3://Insertion Sort
            InsertionSort(array, n);
            cout << "Sorted Array:" << endl;
            for (int i = 0; i < n; i++) {
                cout << *(array + i) << " ";
            }
            cout << endl;
            for (int i = 0; i < n; i++) {
                OutFile << *(array + i) << " ";
            }
            OutFile << endl;
            Sorted=true;
            break;
        case 4://Bubble Sort
            BubbleSort(array, n);
            cout << "Sorted Array:" << endl;
            for (int i = 0; i < n; i++) {
                cout << *(array + i) << " ";
            }
            cout << endl;
            for (int i = 0; i < n; i++) {
                OutFile << *(array + i) << " ";
            }
            OutFile << endl;
            Sorted=true;
            break;
        case 5://Quick Sort
            QuickSort(array, 0, n - 1);
            cout << "Sorted Array using Quick Sort:" << endl;
            for (int i = 0; i < n; i++) {
                cout << *(array + i) << " ";
            }
            cout << endl;
            for (int i = 0; i < n; i++) {
                OutFile << *(array + i) << " ";
            }
            OutFile << endl;
            Sorted=true;
            break;
        case 6://Radix Sort
            RadixSort(array, n);
            cout << "Sorted Array using Radix Sort:" << endl;
            for (int i = 0; i < n; i++) {
                cout << *(array + i) << " ";
            }
            cout << endl;
            for (int i = 0; i < n; i++) {
                OutFile << *(array + i) << " ";
            }
            OutFile << endl;
            Sorted=true;
            break;
        case 7://Exit Program
            OutFile.close();
            delete[] array;
            return 0;
        default:
            cout << "Invalid choice!" << endl;
            break;
        }
    }
    OutFile.close();
    delete[] array;
    return 0;
}
```

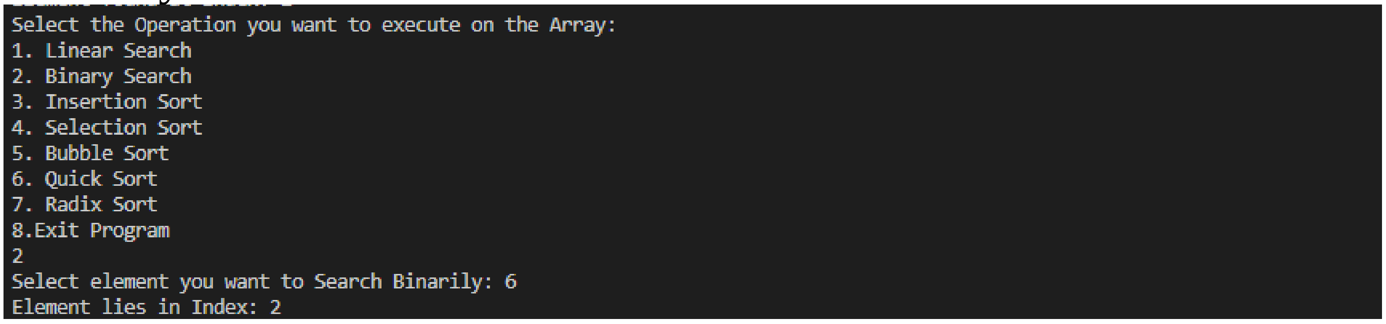
Output:

1. Linear Search:



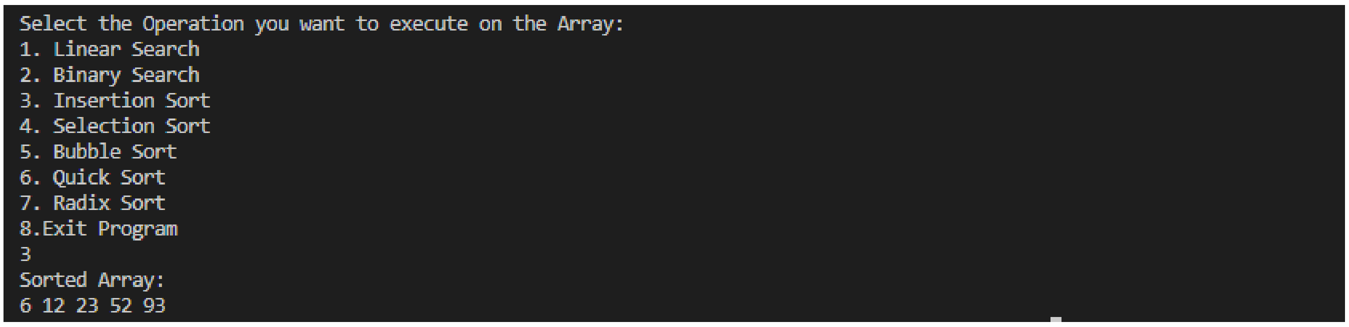
```
Enter the size of Array: 5
Original Array:
23 52 6 12 93
Select the Operation you want to execute on the Array:
1. Linear Search
2. Binary Search
3. Insertion Sort
4. Selection Sort
5. Bubble Sort
6. Quick Sort
7. Radix Sort
8.Exit Program
1
Select element you want to Search Linearly: 6
Element found at Index: 2
```

2. Binary Search:



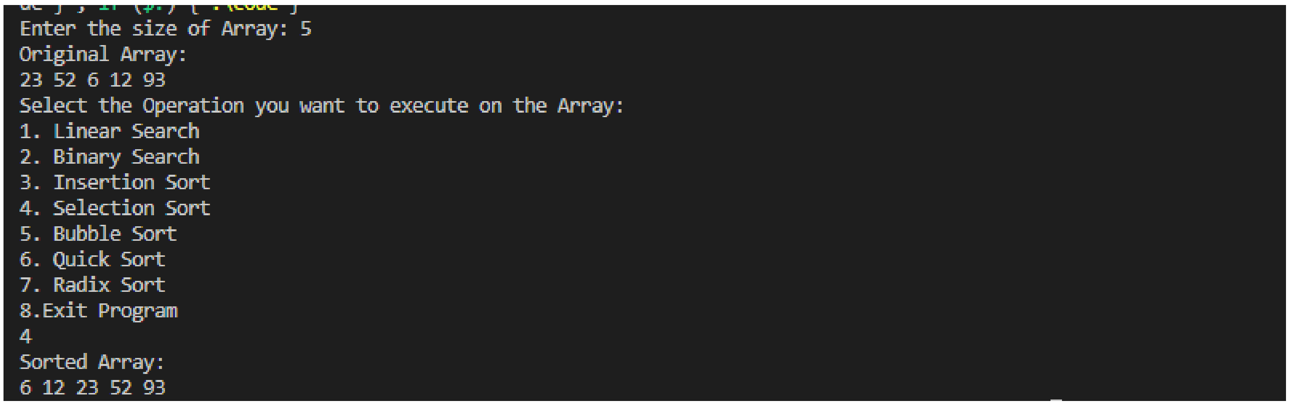
```
Select the Operation you want to execute on the Array:
1. Linear Search
2. Binary Search
3. Insertion Sort
4. Selection Sort
5. Bubble Sort
6. Quick Sort
7. Radix Sort
8.Exit Program
2
Select element you want to Search Binarily: 6
Element lies in Index: 2
```

3. Insertion Sort:



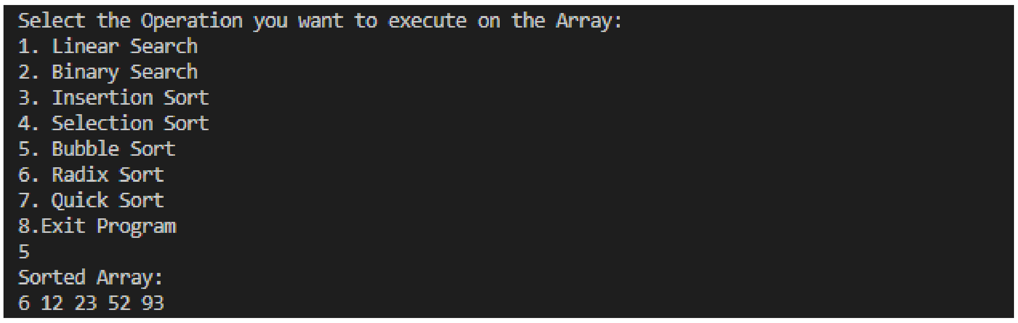
```
Select the Operation you want to execute on the Array:
1. Linear Search
2. Binary Search
3. Insertion Sort
4. Selection Sort
5. Bubble Sort
6. Quick Sort
7. Radix Sort
8.Exit Program
3
Sorted Array:
6 12 23 52 93
```

4. Selection Sort:



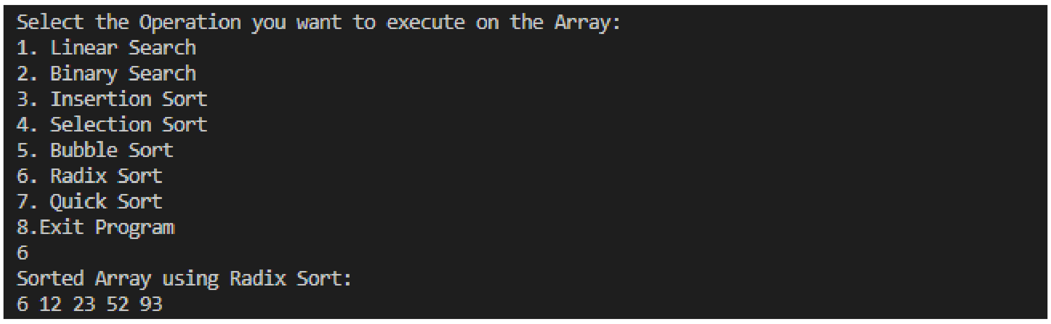
```
Enter the size of Array: 5
Original Array:
23 52 6 12 93
Select the Operation you want to execute on the Array:
1. Linear Search
2. Binary Search
3. Insertion Sort
4. Selection Sort
5. Bubble Sort
6. Quick Sort
7. Radix Sort
8.Exit Program
4
Sorted Array:
6 12 23 52 93
```

5. Bubble Sort:



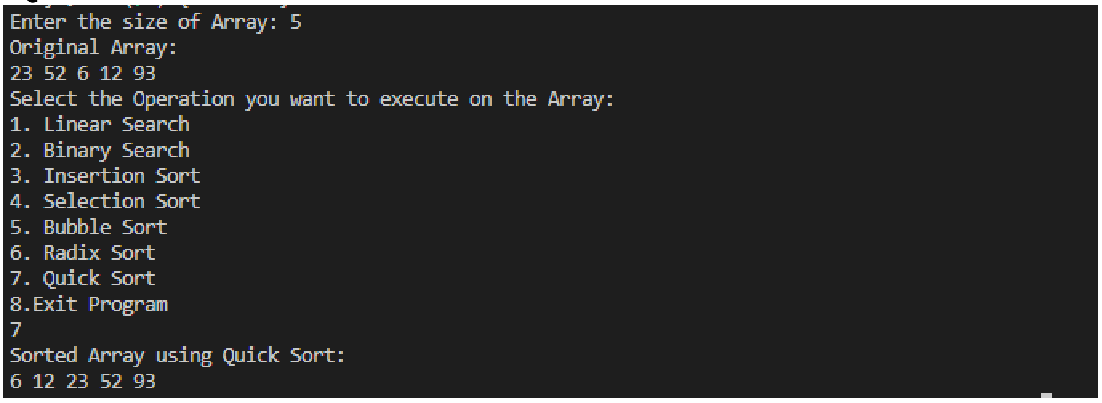
```
Select the Operation you want to execute on the Array:
1. Linear Search
2. Binary Search
3. Insertion Sort
4. Selection Sort
5. Bubble Sort
6. Radix Sort
7. Quick Sort
8.Exit Program
5
Sorted Array:
6 12 23 52 93
```

6. Radix Sort:



```
Select the Operation you want to execute on the Array:
1. Linear Search
2. Binary Search
3. Insertion Sort
4. Selection Sort
5. Bubble Sort
6. Radix Sort
7. Quick Sort
8.Exit Program
6
Sorted Array using Radix Sort:
6 12 23 52 93
```

7. Quick Sort:



```
Enter the size of Array: 5
Original Array:
23 52 6 12 93
Select the Operation you want to execute on the Array:
1. Linear Search
2. Binary Search
3. Insertion Sort
4. Selection Sort
5. Bubble Sort
6. Radix Sort
7. Quick Sort
8.Exit Program
7
Sorted Array using Quick Sort:
6 12 23 52 93
```

Log File as Machine Code:

```
01001111 01110010 01101001 01100111 01101001 01101110 01100001 01101100 00100000 01000001
01110010 01110010 01100001 01111001 00111010 00100000
00110010 00110011 00100000
00110101 00110010 00100000
00110110 00100000
00110001 00110010 00100000
00111001 00110011 00100000
01001100 01101001 01101110 01100101 01100001 01110010 00100000 01010011 01100101 01100001
01110010 01100011 01101000 00100000 01101001 01101110 01101001 01110100 01101001 01100001
01110100 01100101 01100100 00100000 01100110 01101111 01110010 00100000 01100101 01101100
01100101 01101101 01100101 01101110 01110100 00111010 00100000 00110110
01000101 01101100 01100101 01101101 01100101 01101110 01110100 00100000 01100110 01101111
01110101 01101110 01100100 00100000 01100001 01110100 00100000 01101001 01101110 01100100
01100101 01111000 00111010 00100000 00110010
01001100 01101001 01101110 01100101 01100001 01110010 00100000 01010011 01100101 01100001
01110010 01100011 01101000 00100000 01101001 01101110 01101001 01110100 01101001 01100001
01110100 01100101 01100100 00100000 01100110 01101111 01110010 00100000 01100101 01101100
01100101 01101101 01100101 01101110 01110100 00111010 00100000 00110110
01000101 01101100 01100101 01101101 01100101 01101110 01110100 00100000 01100110 01101111
01110101 01101110 01100100 00100000 01100001 01110100 00100000 01101001 01101110 01100100
01100101 01111000 00111010 00100000 00110010
01000010 01101001 01101110 01100001 01110010 01111001 00100000 01010011 01100101 01100001
01110010 01100011 01101000 00100000 01101001 01101110 01101001 01110100 01101001 01100001
01110100 01100101 01100100 00100000 01100110 01101111 01110010 00100000 01100101 01101100
01100101 01101101 01100101 01101110 01110100 00111010 00100000 00110100 00110011
01000101 01101100 01100101 01101101 01100101 01101110 01110100 00100000 01101110 01101111
01110100 00100000 01100110 01101111 01110101 01101110 01100100
01001001 01101110 01110011 01100101 01110010 01110100 01101001 01101111 01101110 00100000
01010011 01101111 01110010 01110100 00100000 01101001 01101110 01101001 01110100 01101001
01100001 01110100 01100101 01100100
01000001 01110010 01110010 01100001 01111001 00100000 01110011 01101111 01110010 01110100
01100101 01100100 00100000 01110101 01110011 01101001 01101110 01100111 00100000 01001001
01101110 01110011 01100101 01110010 01110100 01101001 01101111 01101110 00100000 01010011
01101111 01110010 01110100
01001111 01110010 01101001 01100111 01101001 01101110 01100001 01101100 00100000 01000001
01110010 01110010 01100001 01111001 00111010 00100000
00110010 00110011 00100000
00110101 00110010 00100000
00110110 00100000
00110001 00110010 00100000
00111001 00110011 00100000
01001001 01101110 01110011 01100101 01110010 01110100 01101001 01101111 01101110 00100000
01010011 01101111 01110010 01110100 00100000 01101001 01101110 01101001 01110100 01101001
01100001 01110100 01100101 01100100
01000001 01110010 01110010 01100001 01111001 00100000 01110011 01101111 01110010 01110100
01100101 01100100 00100000 01110101 01110011 01101001 01101110 01100111 00100000 01001001
01101110 01110011 01100101 01110010 01110100 01101001 01101111 01101110 00100000 01010011
01101111 01110010 01110100
01001111 01110010 01101001 01100111 01101001 01101110 01100001 01101100 00100000 01000001
01110010 01110010 01100001 01111001 00111010 00100000
00110010 00110011 00100000
```