DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

NAAC GRADE A+
Accredited University

UNIVERSITY INSTITUTE OF ENGINEERING

Department of Computer Science & Engineering

(BE-CSE/IT-5th Sem)

Design and Analysis of Algorithms

Subject Code: 23CSH-301/ITH-301

Submitted to:

Faculty name: MD. Shaqlain

Submitted by:

Name: HarshVardhan

UID: 23BCS10363
Section: KRG-1

Group: A

Name: Aaditya Bansal **UID: 23BCS11115**

INDEX

Ex. No	List of Experiments	Date	Conduct (MM: 12)	Viva (MM: 10)	Worksheet (MM: 8)	Total (MM:30)	Remarks/Signature
1.1	Analyze if stack Isempty, Isfull and if elements are present then return top element in stacks using templates and also perform push and pop operation in stack.						
1.2	Develop a program for implementation of power function and determine that complexity should be O(log n).						
1.3	Evaluate the complexity of the developed program to find frequency of elements in a given array.						
1.4	i. Apply the concept of Linked list and write code to Insert and Delete an element at the beginning and end of Singly Linked List. ii. Apply the concept of Linked list and write code to Insert and Delete an element at the beginning and at end in Doubly and Circular Linked List.						
2.1	Sort a given set of elements using the Quick sort method and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be						

	sorted. The elements can be read from a file or can be generated using the random number generator.						
2.2	Develop a program and analyze complexity to implement subset-sum problem using Dynamic Programming.						
2.3	Develop a program and analyze complexity to implement 0-1 Knapsack using Dynamic Programming.						
3.1	Develop a program and analyze complexity to find shortest paths in a graph with positive edge weights using Dijkstra's algorithm.						
3.2	Develop a program and analyze complexity to find all occurrences of a pattern P in a given string S.						
3.3	Lab Based Mini Project.						

Experiment No: 1.1

Student Name: HarshVardhan
UID: 23BCS10363
Branch: CSE
Section/Group: Krg-1 A
Semester: 5th
Date of Performance: 21/07/25
Subject Name: Design analysis and algorithm
Subject Code: 23CSH-301

Aim:

Analyze if stack Isempty, Isfull and if elements are present then return top element in stacks using templates and also perform push and pop operation in stack.

Procedure/Algorithm/Pseudocode:

```
Step 1: Initialization
    o Input max_size
    o Set top ← -1
    o Create array arr[max_size] to store elements
Step 2: push(element)
    o If top == max_size - 1, print "Stack is full" and return
    o Else, increment top ← top + 1
    o Set arr[top] ← element
    o Print "Pushed element: " + element
Step 3: pop()
    o If top == -1, print "Stack is empty" and return
    o Else, print "Popped element: " + arr[top]
    o Decrement top ← top - 1
Step 4: peek()
    o If top == -1, print "Stack is empty" and return default value
    o Else, return arr[top]
Step 5: isEmpty()
    o If top == -1, return true
```

- Else, return false

Step 6: isFull()

- If top == max_size - 1, return true
- Else, return false

Step 7: Main Function

- Create a stack s of type string with size 5
- Call push() five times with different values
- Call peek() and print the top element
- Call pop() five times to remove all elements

Code:

```
#include <iostream>
using namespace std;

template <typename T>
class Stack {
private:
    T* arr;
    int top;
    int max_size;

public:

    Stack(int max_size) {
        this->max_size = max_size;
        arr = new T[max_size];
        top = -1;
    }
    ~Stack() {
        delete[] arr;
    }
    void push(T element) {
        if (isFull()) {
            cout << "Stack is full" <<
endl;
            return;
        }
        arr[++top] = element;
        cout << "Pushed element: "
<< element << endl;
    }

    void pop() {
        if (isEmpty()) {
            cout << "Stack is empty"
<< endl;
            return;
        }
        cout << "Popped element: "
<< arr[top--] << endl;
    }

    T peek() {
        if (isEmpty()) {
            cout << "Stack is empty"
<< endl;
            return T();
        }
        return arr[top];
    }

    bool isEmpty() {
        return top == -1;
    }
```

```
        bool isFull() {
            return top == max_size - 1;
        }
    };

    int main() {
        Stack<string> s(5);  // Pass an
    int for size

        s.push("23BCS11111");
        s.push("23BCS11112");
        s.push("23BCS11113");
        s.push("23BCS11114");
        s.push("23BCS11115");

        cout << "Top element: " <<
    s.peek() << endl;

        s.pop();
        s.pop();
        s.pop();
        s.pop();
        s.pop();

        return 0;
    }
```

Observations/Outcome:

1. **Stack behaves as expected (LIFO - Last In First Out):**
   The last pushed element is the first one to be popped.
   Example: After pushing "23BCS11115" last, it's the first to be popped.
2. **Push Operation:**
   Elements are successfully added until the stack reaches its max_size.
   After 5 elements, pushing more would trigger a "Stack is full" message (overflow condition is handled).

3. **Pop Operation:**
   Elements are removed in reverse order of insertion.
   After popping all elements, further pop attempts trigger a "Stack is empty" message (underflow condition is handled).
4. **Peek Operation:**
   Correctly returns the top element without removing it.
   Returns default value and prints a warning when the stack is empty.
5. **isEmpty and isFull Functions:**
   Correctly determine the state of the stack.
   Used internally in push() and pop() to manage boundary conditions.
6. **Memory Management:**
   Dynamic memory is allocated using new and properly deallocated in the destructor using delete[].
7. **Type Flexibility via Templates:**
   The stack works with any data type (like string, int, etc.) due to the use of templates.

Time Complexity:

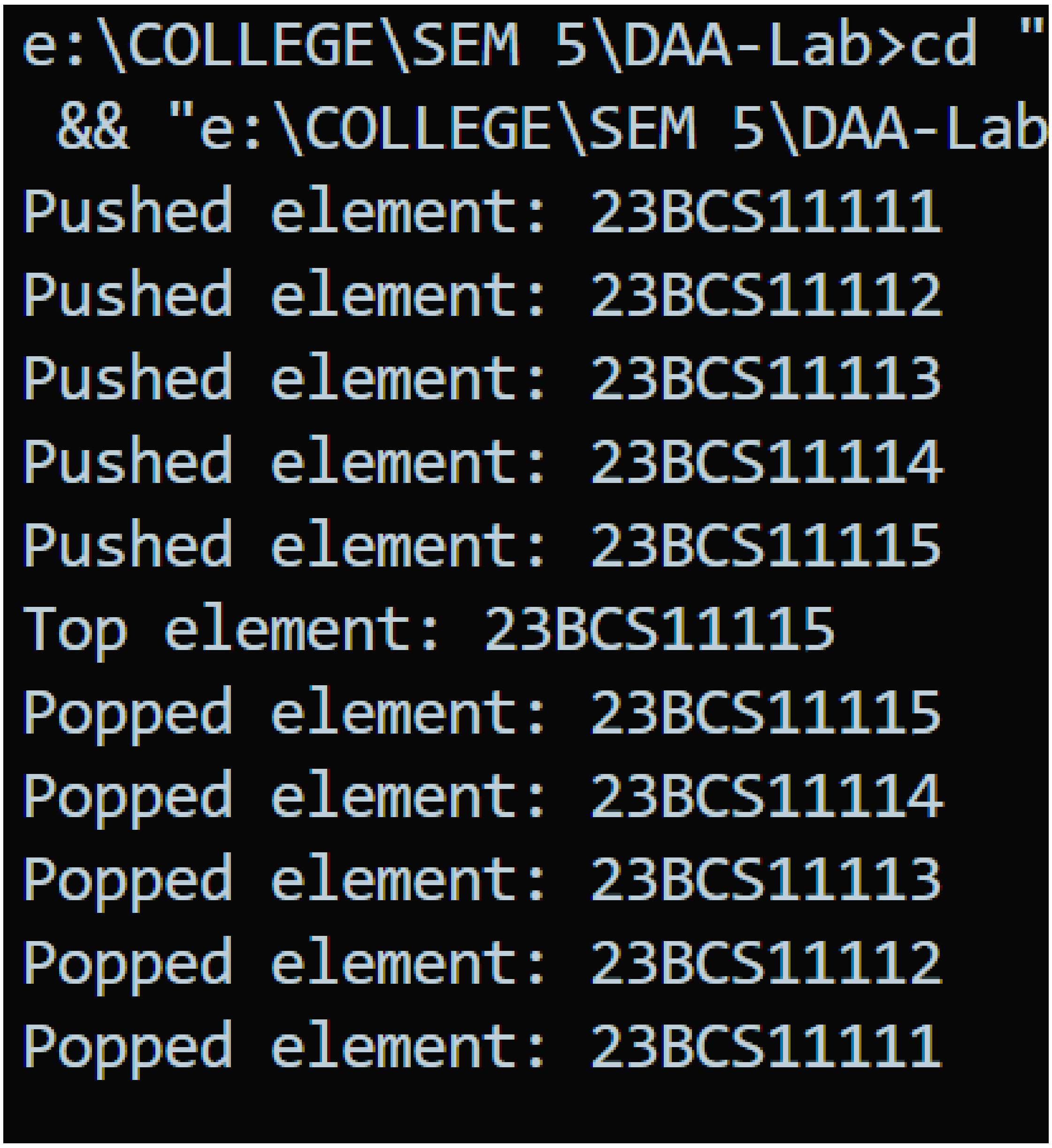
- push(T element): O(1) in the average case, as it involves a simple array assignment and implementing the top index.
- pop(): O(1), as it involves decrementing the top index and accessing the array element.
- peek(): O(1), as it directly accesses the top element.
- isEmpty() and isFull(): O(1), simple comparisons.

Overall, each operation (push, pop, peek, isEmpty, isFull) runs in constant time, O(1).

Learning outcomes:

1. Implementation of Stack Using Arrays and Templates:
2. Understanding Stack Operations and LIFO Behavior:
3. Handling Stack Overflow and Underflow Conditions:
4. Memory Management in C++:

Output:



```
e:\COLLEGE\SEM 5\DAA-Lab>cd "
 && "e:\COLLEGE\SEM 5\DAA-Lab
Pushed element: 23BCS11111
Pushed element: 23BCS11112
Pushed element: 23BCS11113
Pushed element: 23BCS11114
Pushed element: 23BCS11115
Top element: 23BCS11115
Popped element: 23BCS11115
Popped element: 23BCS11114
Popped element: 23BCS11113
Popped element: 23BCS11112
Popped element: 23BCS11111
```