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: Reetesh Sahu
UID: 23BCS10018
Section: KRG-1 A

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.						

DEPARTMENT OF
COMPUTER SCIENCE & ENGINEERING
Discover. Learn. Empower.

Experiment No: 1.1

Student Name: Reetesh Sahu
UID: 23BCS10018
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.

Algorithm: Stack Implementation using Array & Templates

Step 1: Initialization

- Read maximum size of stack max_size.
- Set top = -1.
- Create an array arr[max_size] to hold elements.

Step 2: push(element)

- If top == max_size - 1 → print **"Stack Overflow"** and exit.
- Otherwise:
  - Increment top ← top + 1.
  - Insert the element at arr[top].
  - Print **"Element inserted: element"**.

Step 3: pop()

- If top == -1 → print **"Stack Underflow"** and exit.
- Otherwise:
  - Print **"Element removed: arr[top]"**.
  - Decrease top ← top - 1.

Step 4: peek()

- If top == -1 → print **"Stack is empty"** and return a default value.
- Otherwise:
  - Return the element stored at arr[top].

Step 5: isEmpty()

- 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

1. Create a stack s of type **string** with capacity 5.
2. Call push() five times with different roll numbers (e.g., "23BCS201", "23BCS202", …).
3. Call peek() to display the top element.
4. Call pop() five times to delete all elements.

Code:

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

template <class T>
class Stack {
private:
    T* data;
    int topIndex;
    int capacity;

public:
    // Constructor
    Stack(int size) {
        capacity = size;
        data = new T[capacity];
        topIndex = -1;
    }

    // Destructor
    ~Stack() {
        delete[] data;
    }

    // Insert element into stack
    void pushElement(T value) {
        if (isFull()) {
            cout << "⚠ Stack Overflow! Cannot insert " << value << endl;
            return;
        }
```

```
        data[++topIndex] = value;
        cout << "Inserted: " << value << endl;
    }

    // Remove element from stack
    void popElement() {
        if (isEmpty()) {
            cout << " ⚠ Stack Underflow! No element to remove." << endl;
            return;
        }
        cout << "Removed: " << data[topIndex--] << endl;
    }

    // Show top element
    T topElement() {
        if (isEmpty()) {
            cout << "Stack has no elements." << endl;
            return T();
        }
        return data[topIndex];
    }

    // Check if empty
    bool isEmpty() {
        return topIndex == -1;
    }

    // Check if full
    bool isFull() {
        return topIndex == capacity - 1;
    }
};

int main() {
    Stack<string> st(5);

    // Insertion
    st.pushElement("23BCS201");
    st.pushElement("23BCS202");
    st.pushElement("23BCS203");
    st.pushElement("23BCS204");
    st.pushElement("23BCS205");
```

```
    cout << "Element at Top: " << st.topElement() << endl;

    // Deletion
    st.popElement();
    st.popElement();
    st.popElement();
    st.popElement();
    st.popElement();

    return 0;
}
```

Refined Observations / Results:

1. **LIFO Property Verified:**
   The stack strictly follows *Last In First Out*. Example: 23BCS205 entered last, so it is the first to be deleted.
2. **Insertion (pushElement):**
   Values are added till the stack is full. Once capacity is reached, further attempts give an overflow warning.
3. **Deletion (popElement):**
   Values are removed in reverse order of insertion. Trying to delete beyond empty state gives an underflow message.
4. **Top Element Access (topElement):**
   Correctly shows the element at the top without removing it. If stack is empty, it returns a default value.
5. **Empty & Full Checkers:**
   Both conditions are correctly handled with isEmpty() and isFull().
6. **Memory Handling:**
   Dynamic memory allocation is done in the constructor and safely released in the destructor.
7. **Generic Nature:**
   The template class allows this stack to work with different data types (string, int, char, etc.).

Time Complexity

- **Insertion (pushElement):** O(1) → just index increment + assignment.
- **Deletion (popElement):** O(1) → index decrement.
- **Top Access (topElement):** O(1).
- **isEmpty / isFull:** O(1).
-

  Every operation executes in constant time.

Output:

```
Inserted: 23BCS201
Inserted: 23BCS202
Inserted: 23BCS203
Inserted: 23BCS204
Inserted: 23BCS205
Element at Top: 23BCS205
Removed: 23BCS205
Removed: 23BCS204
Removed: 23BCS203
Removed: 23BCS202
Removed: 23BCS201
```