Stack Mechanism Discussion with Time Complexity

2. *Is Empty*

```
     int isEmpty(Stack st)
{
    if (st.top == -1)
    {
        return 1;
    }
    return 0;
}
….
        case 5:

            if (isEmpty(stck))
            {
                cout << "Stack is Empty" << endl;
            }
            else
            {
                cout << "Stack is not Empty" << endl;
            }
            break;
```

if `top` of the stack is empty i.e. −1, return 1 and

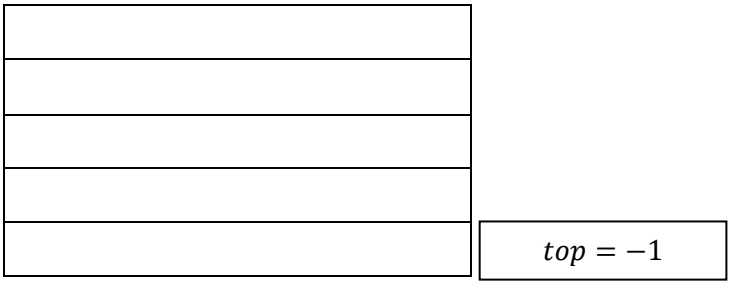
if $(isEmpty(stck))$ returned → `1` in the if condition

it will be stands as true and it will print ``Stack is Empty``.

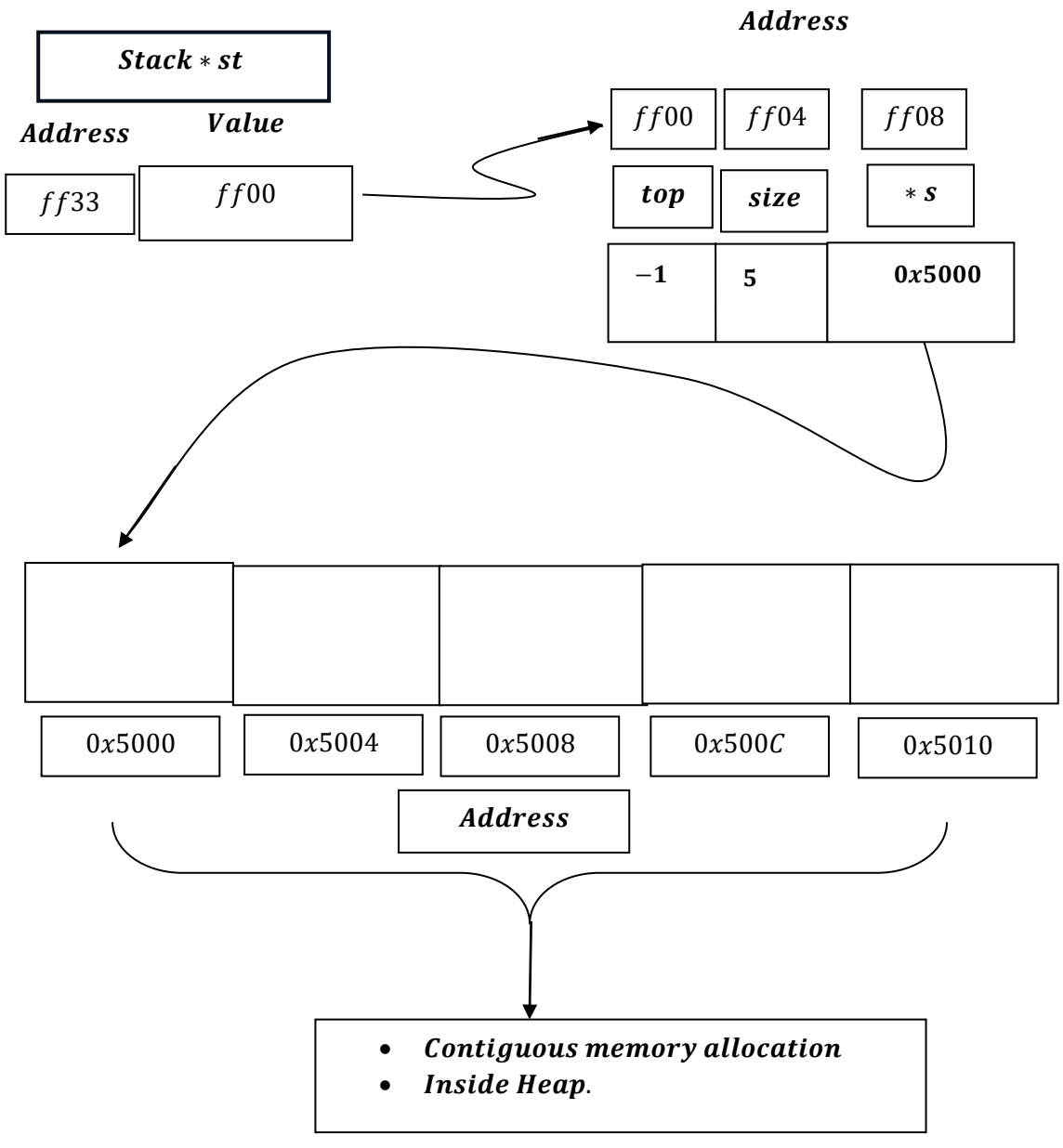
Similarly, if stack is not empty i.e. not −1, then return `0`.

if $(isEmpty(stck))$ returned → `0` in the if condition

it will be stands as false and it will print ``Stack is not Empty``.



Empty stack



- ***Contiguous memory allocation***
- ***Inside Heap***.

Time Complexity of IsEmpty function

```
     int isEmpty(Stack st)
{
    if (st.top == -1)
    {
        return 1;
    }
    return 0;
}
```

Step – by – step Time Analysis

1. ***Function call overhead*** $\rightarrow$ *takes constant time*
   $\rightarrow$ c *units of time* $\rightarrow$ $\boldsymbol{O(1)}$

2. ***Comparison operation***
   $if(st.top == -1)$
   $\rightarrow$ *One comparison*
   $\rightarrow$ *takes* c *units of time* $\rightarrow$ $\boldsymbol{O(1)}$

3. ***Return statement*** **(*either* 1 *or* 0)**
   $\rightarrow$ *Returning a value takes* c *units of time*
   $\rightarrow$ $\boldsymbol{O(1)}$

Total Time Complexity

$O(1) + O(1) + O(1)$
$= \boldsymbol{O(1)}$

What is Function Call Overhead?

Function call overhead means the small amount of extra work the system does when a function is called.

```
int isEmpty(Stack st)

{
    if (st.top == -1)
    {
        return 1;
    }
    return 0;
}
```

Function call:

```
 isEmpty(stck)
```

Even if our function does almost nothing, the system still has to:

1. Save the current execution state
2. Pass arguments to the function
3. Allocate space for local variables
4. Jump to the function's memory location
5. After execution, return back to the caller

All of this takes a **constant amount of time**.

Stack Data structure basically shows how call stack frame works i.e. in LIFO method , but in Stack Data Structure we manually do push() and pop() , where in call stack frame, it creates stack frame for entire function containing local variables , parameters , return address and saved registers and adds or push another stack frame if called again and pops whole stack frame automatically thus different from Stack datastructure , yet conceptually similar.
