NAME : B. Ravi Kumar

REG NO : 192311246

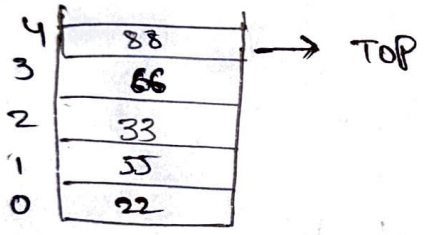
BRANCH : CSE

SUBJECT : DATA STRUCTURE FOR STACK OVERFLOW

CODE : CSA0389.

① Perform the following operations using stack. Assume the size of the stack is 5 and having a value of 22, 55, 33, 66, 88 in the stack from 0 position to size − 1. Now Perform the following operation: 1) Insert the elements in the stack. 2) POP() 3) POP() 4) Push (90) 5) Push (36) 6) Push (11) 7) Push (88) 8) POP(). Draw the diagram of stack and illustrate the above operations and identify where the top is?

Initial stack



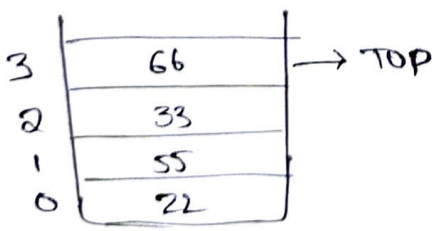
OPERANGE

1. Insert the elements in the stack:

The stack is already initialized with the elements [22, 55, 33, 66, 88]

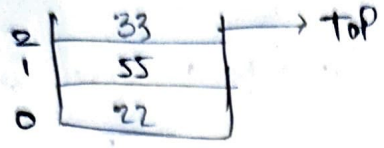
2. POP(): Remove the top element (88)

Stack after POP():



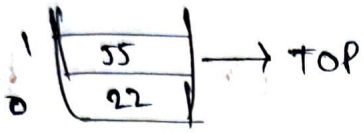
3. POP() : Remove the next top element (66)

Stack after POP():



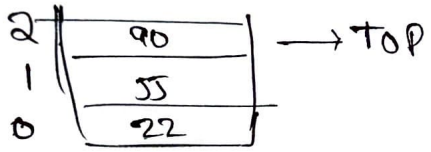
4. POP(): Remove the next top element (33).

Stack after POP():

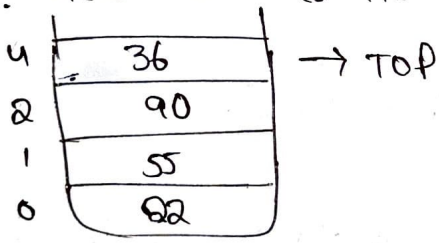


5. PUSH (90) : Add 90 to the Stack.

Stack after Push(90):

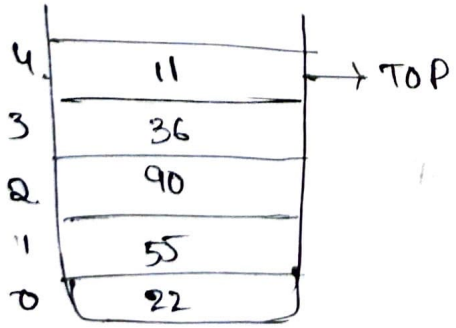


6. Push (36) : Add 36 to the stack.

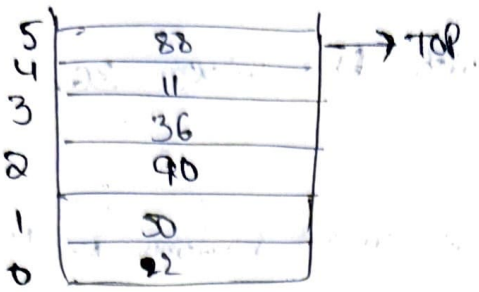


7. Push(11) : Add 11 to the Stack.

Stack after Push(11):

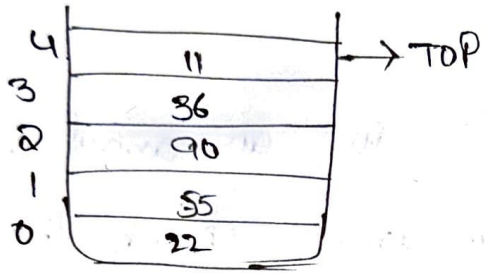


8. Push (88) : Add 88 to the stack.



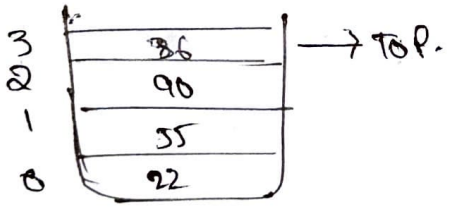
9. POP() : Remove the top element (88)

Stack after POP() :

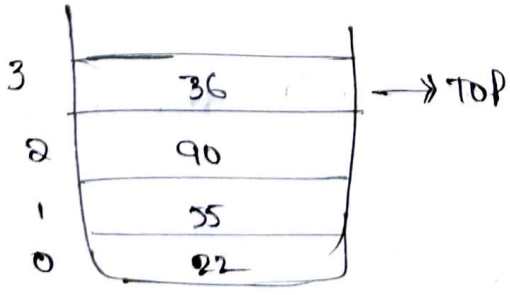


10. POP() : Remove top element (11).

Stack after POP() :



Final stack state.



IDENTIFICATION OF THE TOP : The 'TOP' of the stack is currently at index 3, with the value 36.

CONCLUCION :

* the operations on the stack were performed as specified and the current top element is 36 at index.

(2) Develop an algorithm to detect duplicate element in an unsorted array using linear search, determine the time complexity and discuss.

To detect duplicate elements in an array using linear search, you can use a brute-force approach that involves comparing each element with every other element in the array.

PSEUDO CODE :

```
function function duction(arr):
    duplicate = []
    n = length (arr).
    for i = 0 to n-1:
        for j = i+1 to n-1:
            if array[i] == arr[j] and arr[i] not in
duplicates.
    return duplicates.
```

1). create an empty list duplicates to store duplicate elements.

2) Iterate through each element arr[i] in the array.

3) for each arr[i], Compare it with every subsequent element arr[j].

4) After both both loops complete, return the list of duplicates.

TIME COMPLEXITY:

the time complexity of this brute-force approach is $(O(n^2))$, where (n) is the no. of elements in the array. this is because of each element, the element, algorithm compares it with every other element in the array.

1. USING A HASH SET:

we can use a hash set to keep track of elements we have seen as we iterate through the array. This method reduces the time complexity to $(O(n))$. for insertions and lookups in a hash set.

PSEUDO CODE:

```
function findduplicates (arr) :
    seen = set()
    duplicates = []
    for element in arr :
        if element in seen :
duplicates.append (element)
        else :
            seen.add (element)
    return duplicates.
```

EARLY EXIT ON DETECTION :

the current approach can be optimized to exit early if finds a duplicates is the only requirement. As soon as a duplicate is found, the function can return immediately.

In conclusion, using a set is an efficient way to find duplicates with O(n) time complexity and O(n) space complexity.