Assignment-03

Name : K. Sai Sahithi

Reg.No : 192311240

Sub.code : CSA0389

Sub.Name: Data structore

Faculty name: Dr. ashok kumar

Date : 05-08-2024.

perform following operations using stack. Assume the size of the stack is 5 and having a value of 25, 22,33,66,88 in the stack from o position to size -1. Now, perform the following operations.

1) Invert the elements in the stack, 2, POP[],3 POP[], 3) POP[], 4), push [90], 5) push [36], push, [11], 7], push [88], 8] POP[], 9] POP[].

Draw diagram of stack and intialize the above operations & identify where the top is?

Size of stock: 5

Element in stack (from bottom to top):

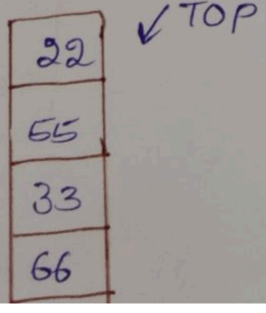
22, 55, 33, 66, 88.

Top of stack: 88

Operations:

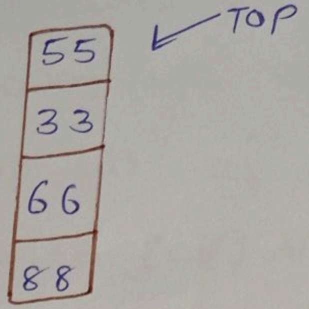
1) Invert the elements in stack.

- The operation will reverse the order of element in the stack.
- After invertion, the stack will look like:



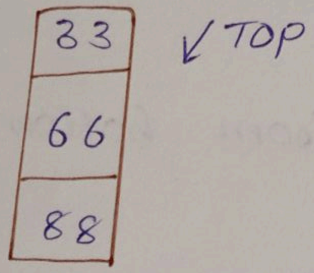
POP( )

- Remove the top element(22).



POP( )

- Remove the top element (55)



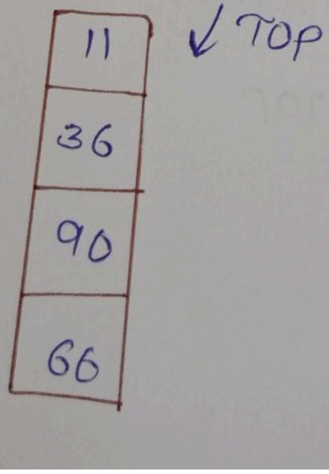
POP( )

- Remove the top element(33).

4) POP( ):

- Remove the top element (88).

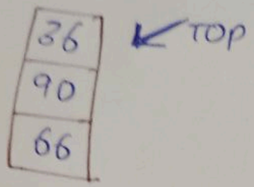
Stack after top:



pop()

Remove the top element (11).

stack after pop:



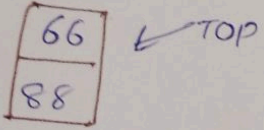
Final stack state:

size of stack :5

elements in stack (fron bottom to top)

36, 90, 66

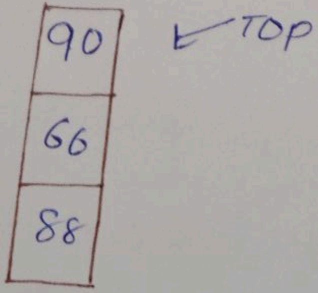
stack after pop.



Push(90):

- push element 90 onto stack

Stack after push



push (36):

- push element 36 onto stack

Stack after push:

88	↙ TOP
11	
36	
90	
66	

push(36):

- push element 36 onto stack

  after push;

88	↙ TOP
11	
36	
90	
66	

push 88:

- push element 88 onto stack.

  stack after push:

88	↙ TOP
11	
36	
90	
66	

Develop alogorithm to detect duplicate elements in an unsorted array using liner search. Determine the time complexity & discuss how you would optimize this process.

gorithm:

Intialization:

creat am empty set or list to keep track of elements that have already been seen.

linear search:

- for each element, check if it is already in the set of seen elements.
- If found. add it to set of seen element.

Output:

Return the list of duplicates, or simple indicate that duplicates exit.

Code:

```
# include< stdio.h>
# include< stdbool.h>
  int main()
  {
   int arr[] = {4,5,6,7,8,5,4,9,0}
   int size = size of (arr)) size of (arr[0]).
    bool seen [1000] = {false}
    for (int i = 0; i< size; i++)
      if(seen [arr[i]])
     printf ("Duplicate found : %d\n", arr[i]);
    else
      seen[arr[i]] = true;
```

```
return0;
}
```

Time complexity

The time complexity for this algorithm is O(n), where 'n', is the number of elements in the array. This is because each element is checked only once, and operations (checking for membership) are O(1) on average.

space complexity:

The space complexity is O(n) due to additional space used by 'seen' & 'duplicates" sets, which may store up to 'n' elements in worst case.

Optimization:

Hashing:

The use of a set for checking duplicates is already efficient because sets provide average O(1) time complexity for membership tests & insertions.