Assignment -03

Name : D.Nandu priya

Reg.NO : 192311173

Sub.Code : CSA0389

Sub.Name : Data structore

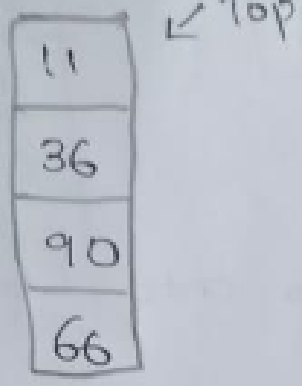
Faculty name : Dr.Ashok kumar

Date : 05-08-2024.

7) pop ( ):

- Remove the top element (88).

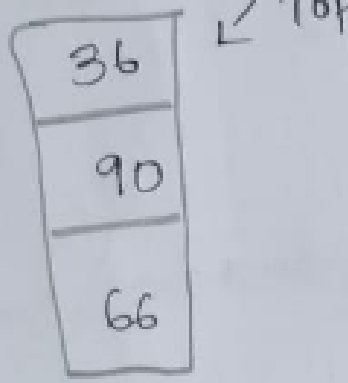
Stack after top:



pop ( )

- Remove the top element (11).

Stack after pop:



Final stack state:-

Size of stack: 5

Elements in stack (from bottom to top):

36, 90, 66.

2) Perform 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 operations.

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

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

Size of stack: 5

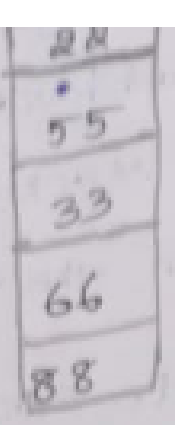
Elements in stack (from bottom to top): 22, 55, 33, 66, 88.

Top of stack: 88

operations.

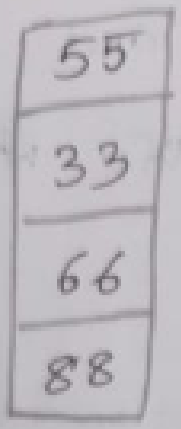
1) Invert the elements in stack:

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



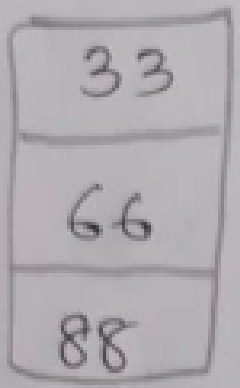
pop( )

- Remove the top element (22).



Pop ( )

- Remove the top element (55)



Pop ( )

Remove the top element (33).

Space Complexity:-

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

optimization :-

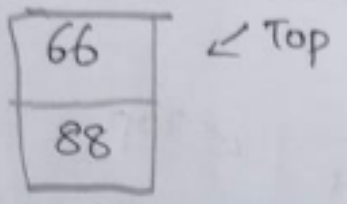
Hashing:-

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

Sorting:-

If we are allowed to modify the array, another approach is to sort the array first and then perform a linear scan to find duplicates. Sorting would take O(n log n) time, and subsequent scan would take O(n) time. This approach uses less Space (O(1) additional space if sorting in

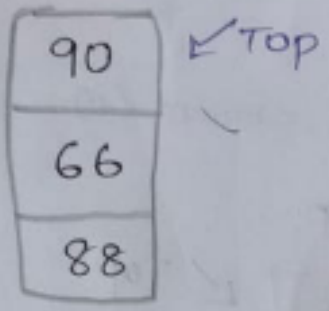
Stack after pop.



Push (90):-

- Push element 90 onto stack.

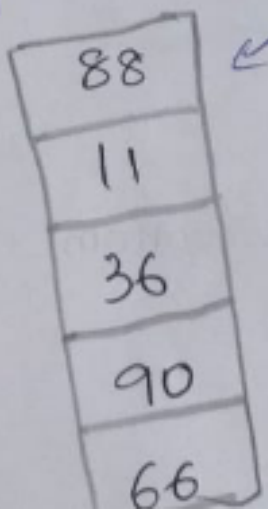
Stack after push.



Push (36):-

- Push element 36 onto stack.

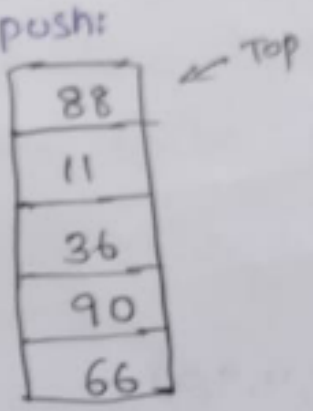
Stack after push:



Push 88:-

Push element 88 onto stack.

Stack after push:



(2) Develop an algorithm to detect duplicate elements in an unsorted array using linear search. Determine the time complexity and discuss how you would optimize this process.

Algorithm:-

1) Initialization:

Create an 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 elements.

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};
int size = sizeof(arr)/sizeof(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;
return 0;
}
```

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