ASSIGNMENT-3

Name :- A. SriHarini

Reg no :- 192372067

Course code :- CSA0389

Course name :- Datastructure

Date of submission :- 05/08/2024

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 operations.
1) Invert the Elements in the Stack, 2, pop [3,3) pop[ ] 4) Push[90], 5) Push [36], 6) Push [11], 7) push [88], 8) Pop[ ], 9) pop[ ]. Draw the diagram of stack and illustrate the above operations and identify where the top is?

A) Size of the Stack : 5

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

Top of stack : 88

88	← Top
66	
33	
55	
22	

Operations :-

1) Invert the Elements in the stack :

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

22	← Top
55	
33	
66	
88	

2) Pop( ):

- Remove the top element (22).

55	← Top
33	
66	
88	

3) Pop( ):

- Remove the top element (55).

33	← Top
66	
88	

4, Pop( ):

- Remove the top element (33).

Stack after pop :

66	← Top
88	

5, Push (90):

- Push the element 90 onto the stack.

Stack after Push.

90	← Top.
66	
88	

6, Push (36) :

- Push the Element 36 onto the stack.

Stack after push :

36	← Top
90	
66	
88	

7) Push (11) :

- Push the Element 11 Onto the stack.

Stack after Push :

11	← Top
36	
90	
66	
88	

8) Push(88) :

- Push the Element 88 Onto the stack

Stack after Push :

88	← Top
11	
36	
90	
66	

9, Pop():

- Remove the top Element (88).

Stack after pop:

11	← Top
36	
90	
66	

10, Pop():

- Remove the top Element (11).

Stack after pop:

36	← Top
90	
66	

Final stack state:-

Size of stack : 5

Elements in stack (from bottom to top):

36, 90, 66

Top of stack : 66

66	← Top
90	
36	

Develop an algorithm to detect duplicate elements in an unsorted array using linear Search. Determine the time complexity and discuss how you would Optimize the process.

Algorithm:-

1) Initialization:

Create an Empty set (or) list to keep track of Elements that have already been seen.

2) Linear Search:

Iterate through Each Element of the array:

- For Each element, check if it is already in the Stack of seen elements.
- If it is, a duplicate has been found.
- If it is found, add it to the Set of seen Elements.

3, Output:

Return the list of duplicates, or Simply indicate that duplicates Exit.

C 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;
 return 0;
}
```

Time Complexity :-

The Linear Search 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 and adding to a set) are O(1) on the average.

Space Complexity :-

The Space complexity is O(n) due to the additional space used on the 'seen' and `duplicates' sets, which may store up to 'n' Elements in the worst Core.

ptimization :-

- 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 the Subsequent Scan would take O(n) time, This approach uses less Space (O(1) additional Space if Sorting in. place).