Assignment - 3

Name :- R. Thejisha

Reg NO :- 192373045

Department :- CSE(DS)

Course code :- CSA0389

Course name :- Data structures

faculty Name :- Dr. Ashok Kumar

Assignment NO :- 03

Submission Date :- 5/8/24.

g operations using stack. ...
s s and having a value of 22, 55,
k from 0. position to size-1
llowing operations
1) ... in the stack, 2) pops 2, 3) pop (3,
5) push (36), 6) push (11), 7) push (88),
... the diagram of stack and
perations and identify where

: 5

from bottom to top: 88, 55, 33,

88.

← TOP

nt in the stack:-
ion reverse the order of elements in

the stack will look like:

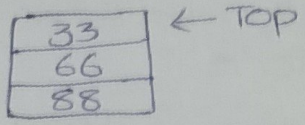
← TOP

ement (22)

← TOP

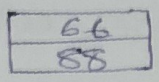
3) POP()

* Remove the top element list



4) POP():

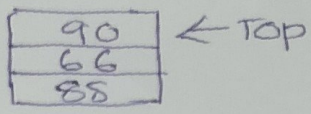
* Remove the top element (33) stack after pop



5) Push(90):-

* Push the element 90 onto the stack

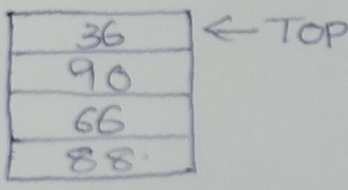
Stack after push.



6) push(36):

* Push the element 36 onto the stack.

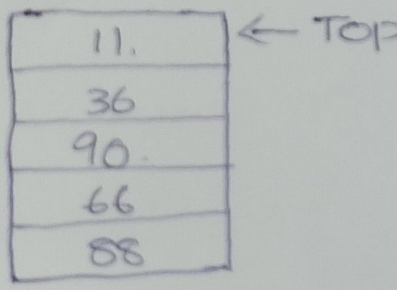
Stack after path.



7) Push(11):-

* push the element 11 onto the stack

Stack after push



8) push(88):-

* Push the element 88 onto the stack.

Stack after push.

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

Algorithm :-

1) Initialization :-

Create an empty set or list to keep track of elements that have already been seen.

2) linear search :-

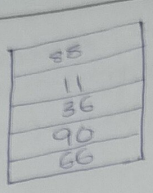
* Iterate through each element of the array
* for each element check if it is already in the list of seen elements
* If it is, a duplicate has been found.
* If it is found, add it to the set of seen elements

3) output :-

Return the list of duplicates, or simply indicate that duplicates exist

C code :-

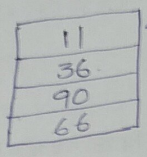
```
# include <stdio.h>
# include <stdbool.h>
   int main ()
   {
int arr [] = {4,5,6,7,8,5,4,9,0};
int size = sizeof(arr)/sizeof(arr[0]);
bool seen (1000) = {false};
for (int i=0; i<size; i++)
  if (seen[arr[i]]),
```



9) pop():-

* Remove the top element (88)

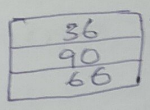
Stack after pop:



10) pop():-

* Remove the top element (11):

Stack after pop



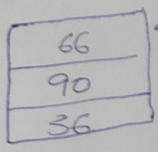
final stack state:-

size of stack:-5

Elements in stack (from bottom to top):

36, 90, 66.

TOP of stack: 66



```
printf("Duplicate found: %d\n", arr[i]);
else
    seen[arr[i]] = true;
return 0;
```

3.

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. May store up to 'n' elements in the worst ca

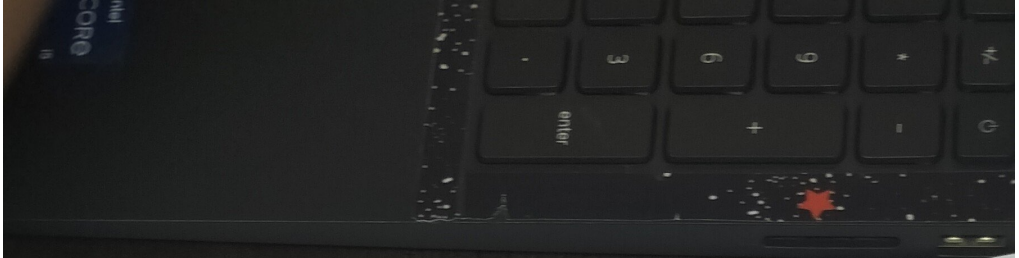
Optimization:-

Hashing:-

The use of a set for checking duplicates is efficient because sets provide average O(1) complexity for membership tests and insertio

Sorting:-

If we are allowed to modify the arr another approach is to sort the array



finest 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(n). additional space if sorting in-place.