Name : T. Sanhith

Reg no :- 192311228

course code :- CSA0389

course Name :- DATA structure

Assignment no: 02

Date of submission : 05-08-24

① Perform the following operations using stack. Assume the size of the stack is 5 and having a value at 22, 55, 33, 66, 68 in stack promoposition to size-1. Now perform the following operations.

i) Invert the elements in stack.

Initial stack :- Assume a stack of size 5 with following element (from bottom to top):

Index	0	1	2	3	4
values	22	55	33	66	88

The top of the stack is at index 4(88).

Operations :-

(i) POP()
- → Removes the top element (88)
- → stack [22, 55, 33, 66]
- → TOP is now at index 3(value 66)

(ii) POP()
- → Removes the top element (66)
- → stack : [22, 55, 33]
- → TOP is now at index 2 [value 33).

(iii) POP()
- → Removes the top element (33)
- → stack : [22, 55]
- → TOP is now at index 1 [value 55)

(iv) Push(90):

→ add 90 to the stack

→ stack: [22, 55, 90]

→ TOP is now at index 2 [value 90]

(v) Push(36):

→ add 36 to the stack

→ stack: [22, 55, 90, 36]

→ TOP is now at index 3 [value 36]

vi) Push(11)

→ add 11 to the stack

→ Stack: [22, 55, 90, 36, 11]

→ TOP is now at index 4 (value 11)

(vii) Push(88)

→ The stack is full, cannot add 88

→ stack remains: [22, 55, 90, 36, 11]

→ TOP remains at index 4 [value 11)

(viii) POP()

→ Remove the top element (11)

→ stack: [22, 55, 90, 36]

→ TOP is now at index 3 (value 36)

(ix) POP()

→ Removes the top element (36)

→ stack: [22, 55, 90]

→ TOP is now at index 2 (value 90)

Final stack Diagram:-

Index	0	1	2	3	4
stack	22	35	90		

→ The TOP of stack is now at index 2(90)

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

What is linear search Algorithm?

linear search is a method for searching for an element in collection of element in linear search, each element of collection is visited one by one in a sequential fashion to find the desired element linear search is also known as sequential search

Find '20' :-

0	1	2	3	4	5	6	7	8	9
10	15	30	70	80	60	20	90	40	55

optimization using a Hash set :-

To optimise this process, we can use a hash set to keep track of the elements we have seen so far. This reduces the time complexity to (O(n)), where n is the number of element in array.

optimized Algorithm :-

1. Initialise an empty Hash set
2. Iterate through each element in array
3. for each element, check if it is already in the hash set
4. if is true, return true
5. if it is not, add it to hash set.
6. If no duplicates are found after all iterations, return false.

Time complexity and space complexity :-

⇒ The time complexity is (O(n)) because.
- each element is checked for presence in hash set in constant time (O(1))

⇒ each element is added to the hash set in constant time (O(1))

→ space complexity of the optimized algorithm is (O(n)) because algorithm is O(n) because, in the worst case, all elements might be unique, requiring storage for all elements in the Hash set.

Ex :- Initial array : [4, 2, 7, 3, 7, 1]

i) Initialize array Hash set : seen = { }

(ii) Iterate through array :

→ check if 4 is in seen : No, add 4 to seen = {4}

→ check if 2 is in seen : No, add 2 to seen: seen = {2, 4}

→ check if 4 is in seen : No, add 7 to seen seen = {2, 4, 7}

→ check if 3 is in seen : No, add 3 to seen seen = {2, 3, 4, 7}

→ check if 7 is in seen : yes, duplicate found return True

* The optimized algorithm finds the duplicate more efficiently, using a hash set to keep track of seen elements and checking for duplicate in constant time.

Time complexity :-

- Basic Algorithm : $(O(n^2))$
- optimized Algorithm : $(O(n))$