Assignment - 3

Name	:	R. Nikhila
Register Number	:	192372186
Department	:	CSE (AI)
year	:	2nd year
course name	:	Data structure for stack overflow
course code	:	CSA0389
Submission Date	:	05-08-2024
Number of pages	:	3
Faculty name	:	Dr. S. Ashok Kumar

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:
i) Insert the elements in the stack 2) pop() 3) pop()
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 ?

Implementation of the stack :

```
# include <stdio.h>
# define MAX_SIZE 5
typedef struct {
  int data [MAX_SIZE];
     int top ;
     }
    stack;
  void initstack (stack *s) {
       s->top = -1 ;
     }
   int is empty (stack *s) {
      return s->top ==-1 ;
      }
    int  isfull (stack * s) {
    return s->top == MAX_SIZE-1;
     }
  void push (stack * s, int value)
    if (isfull (s)) {
     printf ("stack is full. cannot push %d.\n", value);
  return;
 }
s-> data [++s->top] = value;
 }
int pop (stack *s) {
  if (is empty (s)) {
 printf ("stack is empty. cannot pop.\n");
   return -1
    }
return s-> data [s -> top--];
}
void invert (stack *s) {
   int temp [MAX_SIZE];
    int i, j;
for (i=0, j = s->top ; i≤j ; i++, j--) {
       temp [i] = s-> data [j];
        temp [j] = s-> data [i];
  }
  for (i=0; i<s->top ; i++)
```

```
      s->data [i] = temp [i];
   }
   }
   int main () {
    stack s;
     push (&s, 22);
     push (&s, 55);
     push (&s, 33);
     push (&s, 66);
     push (&s, 88);
printf ("Initial stack :\n");
   printstack (&s);
 invert (&s);
printf (" After inverting :\n");
    print stack (&s);
printf ("popped : %d\n", pop (&s));
printf ("popped : %d\n", pop(&s));
printf ("popped : %d\n", pop (&s));
     push (&s, 90);
     push (&s, 36);
      push (&s, 11);
      push (&s, 88);
      printf(" After pushing :\n");
       print stack(&s);
```

```
printf ("popped : %d \n", pop(&s));
printf ("popped : % d\n", pop(&s));
printf ("final stack : \n");
 printstack (&s);
  return 0;
  }
```

output:

```
Initial stack    :
 stack           : 22 55 33 66 88
After Inverting  : 88 66 33 55 22
 popped          : 22
 popped          : 55
 popped          : 33
After pushing    :
 stack           : 88 66 90 36 11
 popped          : 11
 popped          : 36
final stack      :
  stack          : 88 66 90
```

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.

To detect duplicate elements in an unsorted array using linear search:

```
#include <stdio.h>
void detectduplicates (int arr[], int n){
    for (int i=0; i<n; i++) {
     for ( int j=i+1; j<n; j++) {
         if (arr[i] = arr[j]) {
  printf ("Duplicate element found: %d\n", arr[i]);
        return;
         }
          }
          }
   printf (" No Duplicate element found.\n");
 }
int main() {
int arr[] = {5, 2, 8, 12, 3, 2, 1};
 int n= sizeof (arr) / sizeof (arr[0]);
 DetectDuplicates (arr, n);
  return 0;
   }
```

Time complexity: The time complexity of this algorithm is o(n^2) where n is the no of elements in array. this is because using two nested loop to compare each element

optimized version:

```
#include <stdio.h>
#include <stdlib.h>
typedef of struct {
 int * data ;
 int size;
 } Hashtable ;
Hashtable * createHashtable (int size){
Hashtable * ht = (Hashtable *) malloc(sizeof(Hashtable));
ht -> data = (int *) malloc (size * sizeof(int));
ht -> size = size;
return ht ; }
void insert(Hashtable * ht, int value){
int index = value % ht -> size;
While (ht -> data [index] ≠ 0) {
 if (ht -> data [index] = value)
printf("Duplicate element found : %d\n");
return ; }
index = (index + 1) % ht -> size ; }
 ht -> data [index] = value ; }
 int main (){
 int arr() = {5, 2, 8, 12, 3, 2, 1};
 int n = sizeof (arr)/ sizeof (arr[0]);
detectduplicates (arr, n);
return 0;
 }
```