NAME : K. Growtham reddy.

REG NO : 192311220

COURSE CODE : DSA0389.

COURSE NAME : Data
Structure.

ASSIGNMENT : 3.

1) perform the following operations using stack. Assume size of stack is 5 and having a value of 22,55, 33,66,88 in the stack form a position to size-1 Now perform the following operations:

i) insert the elements in the stack 2) pop() 3) pop () 4) 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) Implementation of the stack :-

```
#include <stdio.h>
# define max_size 5
typedef struct {
   int data [max_size];
     int top;
   }
   stack;
   void in stack (stack *s) {
        s->top = -1;
    }
  int  is_empty (stack*s) {
    return s->top = -1;
    }
  int  is_full (stack * s) {
  return s->top = max-size - 1;
```

```
 }
void push (stack *s ,int value )
  if (is full (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 insert (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);
 insert(&s);
printf ("After inserting : \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));

 print stack (&s);
 return 0;

 }
```

Output:

```
Initial stack :

stack : 22 55 33 66 88

After inserting: 88 66 33 55 22

 popped : 22

 popped : 55

 popped : 33

After pushing :

 stack : 88 60 90 36 11

 popped : 11

 popped : 36

final stack :

 stack : 88 66 90.
```

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.

A) To detect duplicate element in an unsorted array using linear search:

```
# include <stdio.h>
void detect duplicates (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 duplicates element found \n");
}
int main () {
int arr[] = {5,2,8,12, 3,2,1};
int n= sizeof(arr) / sizeof (arr[0]);
Detect Duplicates (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>
type def struct {
 int* data;
 int size;
 } Hash table;
Hash table * create Hash table (int size) {
Hash table* ht = (Hash table *)malloc( (size of(hastable));
 ht -> data =(int*) malloc(size * sizeof(int));
ht -> size = size;
return ht; }
void insert (hash table * 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; }
 int main () {
int arr[] = {5,2,8,12,3,2,1};
int n = sizeof(arr) / size of (arr[0]);
detect duplicates (arr, n);
return 0;
}
```