Assignment - 3.

Name : P. Asluritha

Reg no : 192372200

Dept : CSE-AI

Course code : CSA0389

Course name : Datastructure for stack Overflow.

Perform the following operations using stack. Assume size of the 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) Invert the elements in the stack (2) 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
type def struct {
  int data [MAX_SIZE];
  int top;
}
 stact;
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 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");
Print stack (&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 ("Popped : %d \n", Pop(&s));
Printf ("Popped : %d \n", Pop (&s));
Print stack(&s);
  return 0;
 }
```

Output :-

```
Intial stack :-
  Stack  : 22 55 33 66 88
After inverting : 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 elements in an unsorted array linear search:

```
#include <stdio.h>
void detect duplicates (int arr[], int n){
 for (int i=0 ; i<n; i++){
 for (int j=0=i+1 ; j<n; j++) {
 if (arr[i] = arr[j])) {
Printf ("Duplicate element found : %d \n", arr[i]);
return;
}
}
}
Printf (" Duplicate element found : %d \n");
```

```
}
int main() {
int arr[] = {5,2,8,12,3,2,1};
int n = size of (arr)/ size of (arr[0]);
Detect Duplicates (arr/n);
return 0;
}
```

Time Complexity :- The time complexity of this algorithm $O(n^2)$ where n is the no. of elements in array. This is because using two nearest loop to compare each element.

Optimized Version :-

```
#include <stdio.h>
#include <stdlib.h>
type def of struct {
int *data;
 int size;
} Hash table;
Hash table * creat Hashlable (int size) {
Hash table * ht = (Hash table *) malloc (size of (hash
                                          table));
nt → data = (int *) malloc (size of (int));
 nt → 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:}
index = (index+1) % ht→ size; }
ht→ data [index] = value; }
int main() {
int arr() = {5,2,8,12,3,2,1};
int n = size of (arr) / size of (arr[0]);
detect duplicate (arr,n);
return 0;
}
```