Lakesh.B
192365037

CS A0389: Data Structures for stack Overflow

ASSIGNMENT-3

1) Perform the following operations using stack. Assume size of the stack is 5 & having a value of 22,55,33,36,88,66 in the stack from a position to size-1 Now perform the following operations

(1) 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 & illustrate the above operations & identify where the top is?

A) Implementation of the Stack:

```
# include <stdio.h>
# define MAX_SIZE 5
type def 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 [t+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++); --) {
        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,88);
printf("After Pushing:\n");
 print stack(&S);
printf("Popped: %\n", pop(&S));
print("popped:%\n", Pop(&S));
print stack(&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 60 90 36 11

Popped : 11

Popped : 36

Final Stack :-

Stack : 88 66 90

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

A) To detect duplicate elements 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 Duplicate 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.