Practical - 3 (A)

Aim :- Program to create a list-based stack and perform various stack operations.

Description :- Write algorithm.

1) Start
2) Create a list (stack)
3) Append the element in stack using append function.
4) Pop the element from stack using pop() function.
5) Print the stack elements of stack and empty stack
6) Exit

Code :-

```
# Python program to
# demonstrate stack implementation
# using list

stack = []

# append() function to push
# element in the stack
stack.append('a')
stack.append ('b')
stack.append ('c')
```

```
print('Initial stack')
print(stack)

# pop() function to pop
# element from stack in
# LIFO order
print('\nElements popped from stack:')
print(stack.pop())
print(stack.pop())
print(stack.pop())

print('\nStack after elements are popped:')
print(stack)
```

OUTPUT:-

```
Initial stack
['a', 'b', 'c']

Elements popped from stack:
c
b
a

Stack after elements are popped:
[]
```

(B)

Aim:- Program to create infix to postfix expression conversion using stack.

Description:- Write algorithm.

```
1) Start
2) Scan the infix expression from left to right.
3) If the scanned character is an oppend   append
   it with final infix to postfix string.
4) If the scanned character is an '(' or '[' or '{',
   put it to the start.
5) If the scanned character is an ')' or ']' or '}',
   pop the stack and output it until a ')' ~~on~~ or '('
   or '[' or '{' respectively it encountered and
   discard both the hypothesis.
6) Repeat step 2-6 until infix expression is scanned.
7) Print the output.
8) Pop and output from stack until the
```

Code:-

```
Operators = set(['+', '-', '*' '/', '(', ')', '^']) # collection of
                                                       operators
Priority = {'+':1, '-':1, '*':2 '/':2 '^':3} # dictionary
                           having priorities of Operators
```

```
def infixToPostfix(expression):
  stack=[] #initialization of empty stack
  output = ''

for character in expression:
  if character not in Operators: #if an operand
                    append in postfix expression
   output += character
  elif character=='(': #else Operators push
                             onto stack
   stack.append('(')
  elif character == ')':
   while stack and stack[-1]!='(':
     output += stack.pop()
   stack.pop()
  else:
   while stack and stack[-1]!='(' and Priority
       [character]<=Priority[stack[-1]]:
     output += stack.pop()
   stack.append(character)
 while stack:
   output += stack.pop()
 return output

expression=input('Enter infix expression')
print('infix notation:', expression)
print('postfix notation:', infixToPostfix(expression))
```

OUTPUT :-

```
Enter infix expression m*n+(p-q)+r
infix notation: m*n + (p - q) + r
postfix notation: mn*pq-+r+

Enter infix expression (a+(c+d*f+t*a)*b/c)
infix notation: (a+(c + d * f + t * a) * b /c
postfix notation: acdf *+ ta * + b * c /+
```

[illegible]

Practical - 4

(A) Aim :- Linear Search

Description :- Write linear search algorithm.

1) First, read the search element (Target element) in the array.
2) In the second step compare the search element with the first element in the array.
3) If both are matched, display "Target element is found" and terminate the Linear Search function.
4) If both are not matched, compare the search element with the next element in the array.
5) In this step, repeat steps 3 and 4 until the search (Target) element is compared with the last element of the array.
6) If the last element in the list does not match, the Linear Search Function will be terminated, and the message "Element is not found" will be displayed.

Code :-

```
def LinearSearch(array, n, k):
  for j in range(0, n):
    if (array [j] == k):
      return j
~~return -1~~
    return -1
```

```
array = [1, 3, 5, 7, 9]

k = 5
n = len(array)
result = LinearSearch(array, n, k)

if (result == -1):
   print("Element not found")
else:
   print("Element found at index:", result)
```

OUTPUT:-

```
Element not found
```

Aim:-

(B) Binary search (Iterative method)

Description:- Write binary search algorithm.

1) Sort the array in ascending order.
2) Set the low index to the first element of the array and the high index to the last element.
3) Set the middle index to the average of the low and high indices.
4) If the element at the middle index is the target element, return the middle index.
5) If the target element is less than the element

at the middle index, set the low index to the middle index -1.

6) If the target element is greater than the element at the middle index, set the high index to the middle index +1.

7) Repeat steps 3-6 until the element is found or it is clear that the element is not present in the array.

Code:-

```
def binarySearch (arr, k, low, high):
    while low <= high:
        mid = low + (high - low) //2
        if arr[mid] ==k:
            return mid
        elif arr[mid] < k:
            low = mid + 1
        else:
            high = mid -1
    return -1
arr = [1, 3, 5, 7, 9]
k = 5
result = binarySearch (arr, k, 0, len(arr) -1)
if result != 1:
    print("Element is present at index" + str(result))
else:
    print("Not found")
```

FOR EDUCATIONAL USE

OUTPUT:-

```
Element is present at index 2.
```

(C) Aim:- Binary Search (Recursive method)

Description:- Write binary search algorithm.

1) Sort the array in ascending order.
2) Set the low index to the first element of the array and the high index to the last element.
3) Set the middle index to the average of the low and high indices.
4) If the element at the middle index is the target element, return the middle index.
5) If the target element is less than the element at the middle index, set the low index to the middle index -1.
6) If the target element is greater than the element at the middle index, set the high index to the middle index +1.
7) Repeat steps 3-6 until the element is found or it is clear that the element is not present in the array.

Code:-

```
def BinarySearch(arr, k, low, high):
    if high >= low:
        mid = low + (high - low) // 2

        if arr[mid] == k:
            return mid
        elif arr[mid] > k:
            return BinarySearch(arr, k, mid - 1, ~~high~~ low)
        else:
            return BinarySearch(arr, k, mid + 1, high)
    else:
        return -1

arr = [1, 3, 5, 7, 9]
k = 5
result = BinarySearch(arr, k, 0, len(arr) - 1)

if result != -1:
    print("Element is present at index" + str(result))
else:
    print("Not found")
```

OUTPUT:-

```
Element is present at index 2
```

Practical - 5

(A) Aim:- Bubble Sort.

Description:- Write bubble sort algorithm

1) Run a nested for loop to traverse the input array using two variables i and j, such that $0 \leq i < n-1$ and $0 \leq j < n-i-1$
2) If arr[j] is greater than arr[j+1] then swap these adjacent elements, else move on
3) Print the sorted array.

Example:-

First Pass:

$(\underline{5}\ \underline{1}\ 4\ 2\ 8) \rightarrow (\underline{1}\ \underline{5}\ 4\ 2\ 8)$, Here, algorithm compares the first two elements, and swaps since $5 > 1$

$(1\ \underline{5}\ \underline{4}\ 2\ 8) \rightarrow (1\ \underline{4}\ \underline{5}\ 2\ 8)$, Swap since $5 > 4$

$(1\ 4\ \underline{5}\ \underline{2}\ 8) \rightarrow (1\ 4\ \underline{2}\ \underline{5}\ 8)$, Swap since $5 > 2$

$(1\ 4\ 2\ \underline{5}\ \underline{8}) \rightarrow (1\ 4\ 2\ \underline{5}\ \underline{8})$, Now, since these elements are already in order $(8 > 5)$, algorithm does not swap them.

Second Pass:

$(\underline{1}\ \underline{4}\ 2\ 5\ 8) \rightarrow (\underline{1}\ \underline{4}\ 2\ 5\ 8)$

$(1\ \underline{4}\ \underline{2}\ 5\ 8) \rightarrow (1\ \underline{2}\ \underline{4}\ 5\ 8)$, Swap since $4 > 2$

$(1\ 2\ \underline{4}\ \underline{5}\ 8) \rightarrow (1\ 2\ \underline{4}\ \underline{5}\ 8)$

$(1\ 2\ 4\ \underline{5}\ \underline{8}) \rightarrow (1\ 2\ 4\ \underline{5}\ \underline{8})$

Now, the array is already sorted, but our algorithm does not know if it is completed The algorithm needs one whole pass without any swap to know it is sorted

Third Pass:

$(\underline{1}\ \underline{2}\ 4\ 5\ 8) \rightarrow (\underline{1}\ \underline{2}\ 4\ 5\ 8)$

$(1\ \underline{2}\ \underline{4}\ 5\ 8) \rightarrow (1\ \underline{2}\ \underline{4}\ 5\ 8)$

$(1\ 2\ \underline{4}\ \underline{5}\ 8) \rightarrow (1\ 2\ \underline{4}\ \underline{5}\ 8)$

$(1\ 2\ 4\ \underline{5}\ \underline{8}) \rightarrow (1\ 2\ 4\ \underline{5}\ \underline{8})$

Code :-

```
#Python program for implementation of Bubble
 Sort
def bubbleSort(arr):
   n = len(arr)

   #Traverse through all array elements
   for i in range(n):

      #Last i elements are already in place
      for j in range (0, n-i-1):

         #traverse the array from 0 to n-i-1
         #Swap if the element found is greater
         #than the next element
         if arr[j] > arr[j+1]:
            arr[j], arr[j+1] = arr[j+1], arr[j]
```

```
#Driver code to test above
arr = [64, 34, 25, 12, 22, 11, 90]

bubbleSort(arr)

print("Sorted array is:")
for i in range(len(arr)):
    print("%d" %arr[i], end=" ")
```

(B) Aim :- Selection Sort.

Description :- Write selection sort algorithm.

1) Set Min to location 0 in step1.
2) Look for the smallest element on the list.
3) Replace the value at location Min with a different value.
4) Increase Min to point to the next element.
5) Continue until the list is sorted.

Code :-

~~arr[] = 64 25 12 22 11~~

```
#Python program for implementation of Selection
# Sort
import sys
A = [64, 25, 12, 22, 11]
```

```
#Traverse through all array elements
for i in range(len(A)):

  #Find the minimum element in remaining
  #unsorted array
  min_idx = i
  for j in range(i+1, len(A)):
    if A[min_idx] > A[j]:
      min_idx = j

 #Swap the found minimum element with
 # the first element
 A[i], A[min_idx] = A[min_idx], A[i]

#Driver code to test above
print("Sorted array")
for i in range(len(A)):
  print("%d"% A[i], end=" ")
```

OUTPUT :-

```
Sorted array
11 12 22 25 64
```

(C) Aim :- Insertion Sort

Description :- Write a insertion sort algorithm.

1) If it is the first element, it is already sorted. return1;
2) Pick next element
3) Compare with all elements in the sorted sub-list
4) Shift all the elements in the sorted sub-list that is greater than the value to be sorted.
5) Insert the value
6) Repeat until list is sorted.

Code :-

```
#Function to do insertion sort
def insertionSort(arr):

    #Traverse through 1 to len(arr)
    for i in range(1, len(arr)):
        key = arr[i]

        #Move elements of arr[0..i-1], that are
        #greater than key, to one position ahead
        #of their current position
        j = i-1
        while j>=0 and key < arr[j]:
            arr[j+1] = arr[j]
            j = 1
        arr[j+1] = key

arr = [12, 11, 13, 5, 6]
insertionSort(arr)
for i in range(len(arr)):
    print("%d" % arr[i])
```

[illegible]