Problem 1: Consider the following operations on a stack .

stack = empty stack

[]

stack.push(10)

[10]

stack.push(9)

[9,10]

item1 = stack.pop()

[10] ; item1=9

stack.push(8)

[8,10]

stack.push(7)

[7,8,10]

item2 = stack.pop()

[8,10] ; item2=7

stack.push(item1)

[9,8,10]

stack.push(item2)

Final Stack=[7,9,8,10]

Problem 2 :

You are using a queue to build an app that helps users to manage administrative tasks.

This queue supports three operations:

- add : enqueues a task in the queue
- do : dequeues from the queue
- skip : dequeues a task and sends it back to the start of the queue

Consider the following operations on that queue.

tasks = empty queue

tasks.add('email CEO')

tasks.add(‘grab coffee’)

tasks.do()

tasks.add(‘give pay raise’)

tasks.skip()

tasks.add(‘hire employees’)

tasks.skip()

tasks.do()

a) Draw what the queue looks like after each operation .

tasks = empty queue

[]

tasks.add(‘email CEO’)

[‘email CEO’]

tasks.add(‘grab coffee’)

[‘grab coffee’,‘email CEO’]

tasks.do()

[‘grab coffee’]

tasks.add(‘give pay raise’)

[‘give pay raise’,’grab coffee’]

tasks.skip()

[’grab coffee’,‘give pay raise’]

tasks.add(‘hire employees’)

[’hire employees’,’grab coffee’,‘give pay raise’]

tasks.skip()

[‘give pay raise’,’hire employees’,’grab coffee’]

tasks.do()

[‘give pay raise’,’hire employees’]

b) Write down a pseudo-code implementation of the skip function. The function should take a queue as argument and does not return anything.

```
function skip(queue):
    If queue is not empty:
        Item=queue.last
        queue.last=item.previous
        queue.last.next=nil
        new_node=Node(value=item,next=queue.top)
        queue.top=new_node
```

Problem 3 :

In class, we mentioned that a doubly-linked-list is a good data structure for implementing a queue . In this problem, we will verify if an array-list could also be a good choice of data structure.

a) (2 points) In one or two sentences explain which two features of a doubly-linked-list allows to implement a queue efficiently

Sol : we can access the first node as well as last node of doubly linked list instantly without traversing the whole list which makes the implementation of queue efficient in doubly-linked-list.

b) The function prepend inserts an item at the beginning of an array-list . Write the pseudocode for the function prepend. The function should take a list and an item as arguments and return nothing

Sol :

```
function prepend(arrayList,item):
    Size=length(arraylist)
    LastElementIndex=value
    If arrayList is full:
        NewArrayList=new [length(size*2)]
        For element in arrayList:
            NewArrayList[element+1]=arrayList[element]
        NewArrayList[0]=item
        LastElementIndex=LastElementIndex+1
```

```
  else:
       for element in arrayList(start=LastElementIndex,end=0):
            arrayList[element+1]=arrayList[element]
       ArrayList[0]=item
       LastElementIndex=LastElementIndex+1
```

c) Using the pseudocode you wrote in the previous question, count how many operations are needed to prepend an item and determine the time-complexity of the function.

Sol : time complexity: O(n)

d) In one or two sentences explain why an array-list is not a good choice of data structure to implement a queue.

Sol : array-list is not a good choice of data because whenever we want to queue ,we need to shift all the elements in array list to next indexes .if array-list is full then we need to create new array-list with increased size ,then copy all the elements and then insert the element at zero index.

Problem 4

You are working for a company developing an IDE similar to PyCharm. You have been given the task of implementing parentheses matching. Parentheses matching warns the user if their opening and closing parentheses do not match.

Using pseudo-code , implement a function called match which takes a string of parentheses as argument and returns True if the parentheses match and False if the parentheses mismatch.

Sol :

```
function match(list):
   New_list=emptyList
   Boolean verify=True
   for each element in list up to length(list):
       If list[element] is equal to  " ( " :
           append to the New_list( " ( " )
       else:
            If length(New_list) is equal to zero:
                Change verify to False
        else if list[element] is equal to “ ) ” :
            pop last element from the New_list


If length(New_list) is equal to zero  and verify is equal to True:
    return True
else:
     return False
```