**1. Linked lists and arrays:**

**a. What are some advantages of linked lists versus arrays?**

Linked lists are dynamic in nature and elements can be stored at any available place as address of node is stored in previous node.

Inserting of new items or removal of existing items is inexpensive provided the position of the changes is known.

Adding a new element at the beginning of the list and removal at the end of the list are constant time operations.

Methods addFirst(),addLast(),removeFirst(),removeLast(),getFirst(),getLast() are defined to efficiently add, remove and access items at both ends of the list.

It's easier to store data of different sizes in a linked list. An array assumes every element is exactly the same size.

Besides inserting into the middle of the list being easier, it is also much easier to delete from the middle of a linked list than an array.

**b. What are some advantages of arrays versus linked lists?**

In array, elements are stored in **consecutive** manner in memory so an array is very good when it comes to accessing index. If we need to access elements at specific positions often, it is a good idea to use an array.

Arrays are also good at searching/finding (cheaper if the index position is known).

Arras can be randomly accessed like a[20], a[13], etc. Hence, array provides fast and **random access.**

Array can be **single dimensional**, two dimension or **multidimensional**.

Each element is independent and each element has no connection with previous element or with its location.

No pointers are used like linked list so no need of extra space in memory for pointer.

**2. a. If an ArrayList is passed for lst1 and lst2. Explain your answer.**

O(N^2) because adding to the front is an O(N) operation and N iterations take place (N=size of lst1).

**b. If a LinkedList is passed for lst1 and lst2. Explain your answer.**

O(N) because adding to the front of a linked list is an O(1) operation and the loop iterates N times.

**3. a. If an ArrayList is passed for lst. Explain your answer.**

O(N^2) because removing in an ArrayList if an O(N) operation as after an element is removed all the remaining elements shift by one position also the while loop runs N times(i.e the size of lst)

Hence, N times O(N).

**b. If a LinkedList is passed for lst. Explain your answer.**

O(N) as removing in a linked list if O(1) operation and while loop runs N times.

Hence,N times O(1).

**4.**

**a. If an ArrayList is passed for lst1 and lst2. Explain your answer.**

O(N^2) because it costs O(N) for each while loop and hasNext() and next() are O(1) operations.

**b. If a LinkedList is passed for lst1 and lst2. Explain your answer.**

O(N^2) because it costs O(N) for each while loop and hasNext() and next() are O(1) operations.

**5.**

**a.If an ArrayList is passed for lst. Explain your answer.**

O(N) because the loop runs N times and get() method costs O(1) for an ArrayList.(Considering the get() method in the else branch as it is the longer branch)

**b.If a LinkedList is passed for lst. Explain your answer.**

O(N^2) because the loop runs N times and get() method costs O(N) since the entire list must be traversed through. (Considering the get () method in the else branch as it is the longer branch)

**6.** **What is the expected Big-O running time if:**

**a. If an ArrayList is passed. Explain your answer.**

To remove each element from the front of an ArrayList = O(N) because each elements shifts left by one position.

Adding each element on to the stack -> pushing a stack = O(1)

Popping a stack = O(1)

Adding to the back of an ArrayList = O(1)

This happens N times(The size of the array).

Hence, the running time is O(N^2).

1. **If a LinkedList is passed. Explain your answer.**

To remove each element from the front of a linked list = O(1)

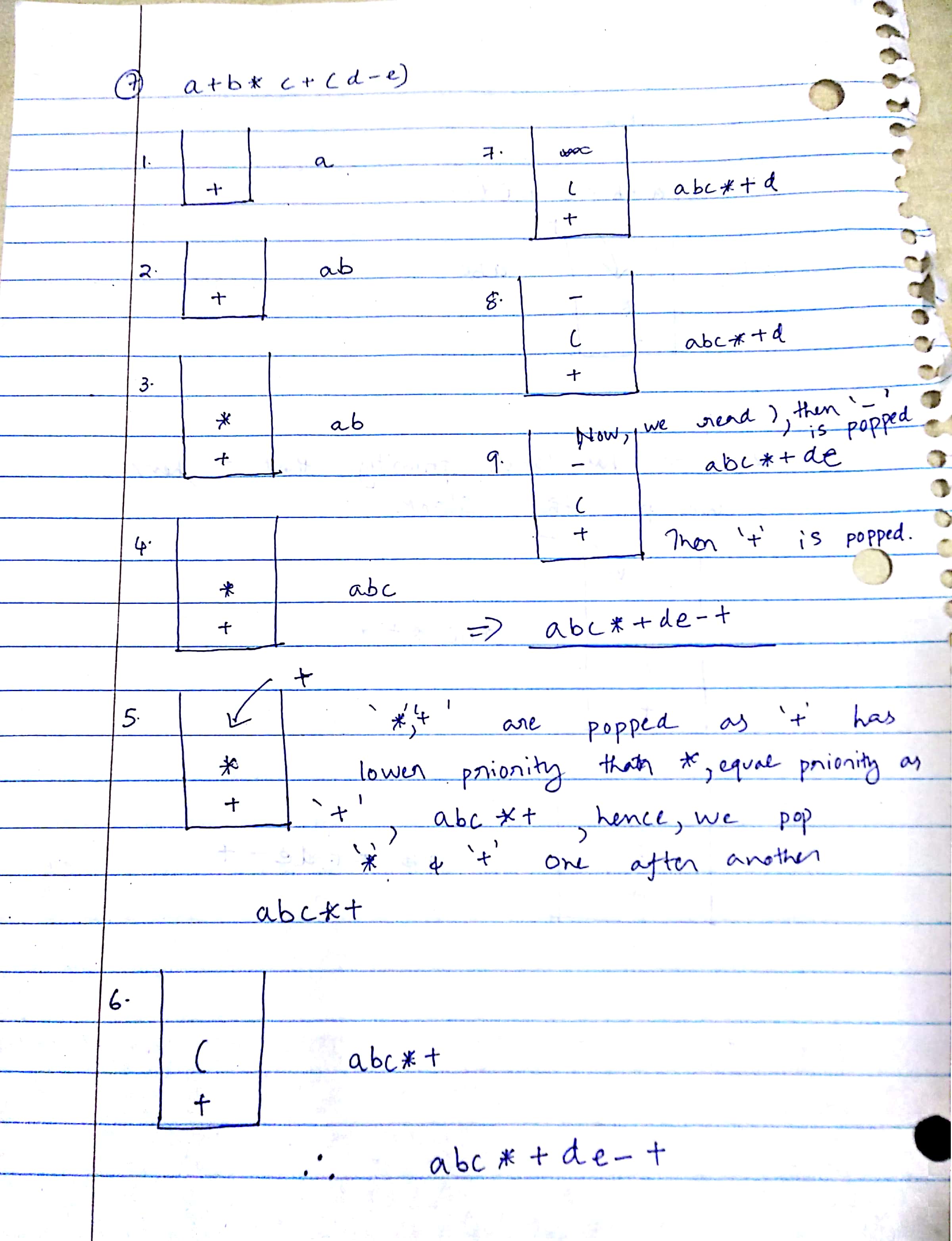
Adding each element on to the stack -> pushing a stack = O(1)

Popping a stack = O(1)

Adding to the back of a linked list = O(N) as each time we need to find the tail node, we need to traverse through the entire list to check if that node is pointing to null and add the new node and get the last but one node(which was previously the tail node) to point to this new node and in turn get the new node to point to null.

Hence, the running time is O(N).

**7. Show each step of converting a+b\*c+(d-e) from infix to postfix notation, using the algorithm described in the textbook that uses a stack.**

****