# ASSIGNMENT #3

SUBMITTED BY :

NAME : SAI KIRAN TADURI

NET ID : sxt161730

① a. Advantages of linked lists versus arrays

→ In a linked list the nodes need not be contiguous in memory. This allows a new node to be inserted or removed in O(1) time.

→ Elements can be inserted into linked lists indefinitely, while an array will eventually either fill up or need to be resized, an expensive operation that may not even possible if memory is fragmented. Similarly, an array from which many elements are removed may become wastefully empty or need to be made smaller.

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

→ An array's size needs to be known ahead of time while linked list grow organically.

(b) Advantages of arrays versus linked lists:

→ Accessing an item in array can be done by its index and can occur in O(1) time. On the other hand, going to the $i^{th}$ node is O(N) time for a linked list, because the list cannot be indexed the way array can. The only way to find an item is to traverse the list.

→ Arrays only need storage to store elements where as linked lists need extra storage for storing references.

(2) a)

```
public static void add(List<Integer> lst1,
                       List<Integer> lst2)
{
    for (Integer x: lst1)        - n times
        lst2.add(0,x); // add to front.
                       n times
}
```

If an ArrayList is passed for lst1 and lst2, adding an element to the front of the list takes n times as the list has to move the already stored elements to its right which takes n times. The for loop takes n times as lst1 has N items. Hence the time complexity for this algorithm that is the Big(O) = $\boxed{O(n^2)}$

2 (b) If a Linked list is passed for lst1 and lst2, adding an element to the front takes constant time O(1) as there is no need to move the elements. O(N) for traversing lst1.

Therefore, the running time is **O(N)**.

(1)

3 (a)

```
public static void erase(List<Integer> lst)
{
    Iterator<Integer> itr = lst.iterator();
    while (itr.hasNext())
    {
        Integer x = itr.next();
        itr.remove();
    }
}
```

(4)

If an arraylist is passed for lst, removing an element takes $O(N)$ as the items must be shifted. The loop makes $N$ iterations. Therefore total running time is

$\boxed{O(N^2)}$

(b) If a linked list is passed for lst, it will take $O(1)$ for removing the element since it already has the position. The loop makes $N$ iterations. Therefore total running time is $O(N)$.

(4) (a) If an arraylist is passed for lst1 and lst2 ; the outer loop i.e., lst1 iterator object loop ~~takes~~ makes $N$ iterations and the inner loop i.e., lst2 iterator

object loop makes N iterations. Therefore, the total running time is $\boxed{O(N^2)}$

(b) If a LinkedList is passed for lst1 and lst2, it will also take $O(N^2)$ as there are two loops. Therefore similar to ArrayList even if LinkedList is passed it takes $\boxed{O(N^2)}$ considering two loops.

(5) (a) If an ArrayList is passed for lst, for loop takes N time, lst.get(i) takes constant time $O(1)$, Therefore the total running time is $O(N) \times O(1) \times O(1)$

$= \boxed{O(N)}$

(b) If a linked list is passed for lst, the loop makes N iterations,

```
for (int i=0; i<N; i++)
{
    if (lst.get(i) > 0)          N times
        sum += lst.get(i);       N times.
    else
        sum += lst.get(i) * lst.get(i);   2N
                                (N times + N times)
}
return sum;
}
```

Therefore total time is

3N times and loop makes N iterations $= O(3N^2)$

$= \boxed{O(N^2)}$

(6)(a) If an ArrayList is passed, removing each item from the front of the list and moving the items takes N time. Pushing it onto a Stack takes constant time O(1). Popping also takes O(1) time. and inserting each item to the end of the list takes constant time. O(1) and the loop takes N iterations. Therefore, the total running time is

$$O(N^2)$$

(b) If a LinkedList is passed, removing each item from the front of the list takes constant time O(1). Pushing and popping takes constant time. Inserting each item

to the end of the list takes O(1). The loop takes N iterations. Therefore, the running time is

$\boxed{O(N)}$

(7) Given,

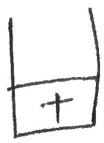
$a + b * c + (d - e)$

Output operand and push operators on to the stack.

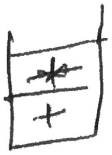
Initially,

a


a

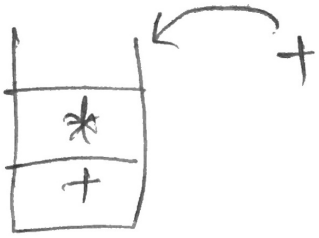



ab



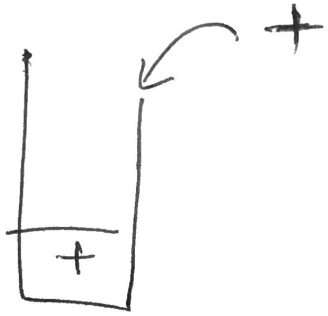

ab

abc



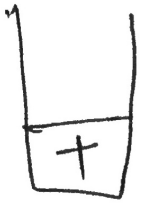

Now + is encountered and top of the stack element has higher precedence that the operator, therefore we pop.

abc *

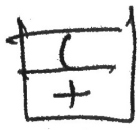



We pop the top of the stack as both have same precedence.

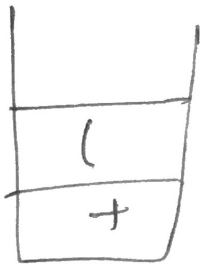
abc**+



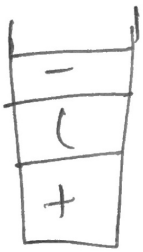

abc*+



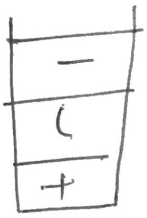
abc*+d




abc*+d

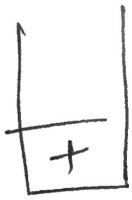



abc*+de




Now ')' is encountered so we pop all the items till we reach '('.

abc*+de-




As the expression is traversed we pop all the remaining elements from the stack. Therefore,

**abc*+de-+**