Data Structure & Algorithm
Chapter 3

NO. SHIH-HAN WANG
DATE 2021389848

1)

a. Linked list can removed and inserted elements in O(1) time without memory overhead. When dealing with frequently changed elements in a list, it is much efficient than array list.

b. Array list can get and set elements in O(1) time, because array list have index, and its size is variable to grow and shrink. It is better to use array list when handling sequential add and get to elements to the list.

2)

```
public static void add(List<Integer> lst1, List<Integer> lst2)
{
    for (Integer. x : lst1)        // N
        lst2. add (0, x);          // insert x to the front of lst2
}
```

a) ArrayList ($N + N*N = N^2+N \Rightarrow O(N^2)$)
When lst2 add element to the front, lst2 have to move the rest of element down, so it will cost in O(N) time, and there are N iterations, so the ArrayList is $O(N^2)$.

b) LinkedList ($N + N*1 = 2N \Rightarrow O(N)$)
When lst2 add element to the front, it only need O(1) time, because it can add new node to the front of the list, and there are N iterations, so the LinkedList is O(N).

NO.

DATE / /

3) public static void erase (List<Integer> lst)

```
{
    Iterator<Integer> itr = lst.iterator();
    while(itr.hasNext())        // N
    {
        Integer x = itr.next();
        itr.remove();           // N
    }
}
```

a) ArrayList ($N*N=N^2 \Rightarrow O(N^2)$)

remove must shift the items over is $O(N)$ time, the ArrayList is $O(N^2)$ time #

b) LinkedList ($N*1=N \Rightarrow O(N)$)

Because it already have position, so it will take $O(1)$, the LinkedList is $O(N)$ time. #

4) public static int Count (List<Integer> lst1, List<Integer> lst2)

```
{
    Iterator<Integer> itr1 = lst1.iterator();
    int count=0;
    while(itr1.hasNext())       // N
    {
        Integer x = itr1.next();        // N
        Iterator<Integer> itr2 = lst2.iterator();
        while(itr2.hasNext())   // N
        {
            if(x.equals(itr2.next()))
                count++;                // 1
```

NO.

DATE / /

```
        }
        return count;
    }
```

a) ArrayList

two N-loop → $O(N^2)$ #

b) LinkedList

two N-loop → $O(N^2)$ #

5)

```
public static int calc (List<Integer> lst)
{
    int count = 0;
    int N = lst.size();
    for( int i=0 ; i<N; i++ )        // N
    {
        if (lst.get(i) > 0)
            sum += lst.get(i);
        else
            sum += lst.get(i) * lst.get(i);
    }
    return Sum;
}
```

a) ArrayList

lst.get(i) is O(1) time, only one for-loop

→ ArrayList is O(N) #

b) LinkedList

lst.get(i) is O(N) time, if is O(N), and there have one for-loop → Linked List is $O(N^2)$ #

SEASON

NO.

DATE / /

6) public int method (List<integer> lst)

```
{
    Stack<integer> stack = new Stack<integer>();

    for(int i=0; i<lst.size()-1; i++)          //N
    {
        int x=lst.remove(i);        //N      //N
        stack.push(x);              //1      //1
    }

    for(int j=0;j<stack.size()-1;j++)          //N
    {
        int x=stack.pop();          //1      //1
        lst.add();                  //1      //1
    }

}
```

a) ArrayList.

remove() have to move element each time is O(N), and there are N iterations, so ArrayList is $O(N^2)$.#

b) LinkedList.

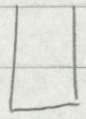
remove() have to traverse the list, so is O(N), and there are N iterations, so ArrayList is $O(N^2)$.#

NO.

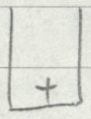
DATE / /

7) a+b*c+(d-e) ⇒ abc*+de-+ #

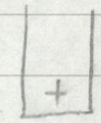
① a



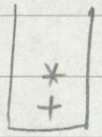
② a



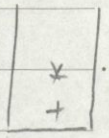
③ ab



④ ab

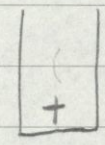


⑤ abc

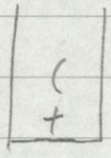


two operator is seen, push all.

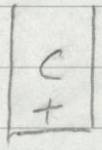
⑥ abc*+



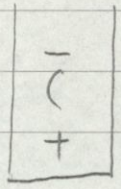
⑦ abc*+



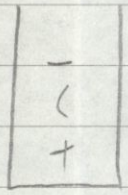
⑧ abc*+d



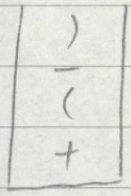
⑨ abc*+d



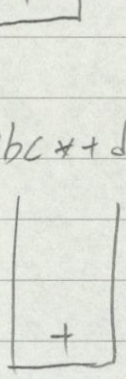
⑩ abc*+de



⑪ abc*+de



⑫ abc*+de-



⑬ abc*+de-+