**Assignment #3**

Yaokun Wu

**1.**

**a).** Inserting or deleting a node only takes O(1) time in linked list whereas O(n) time is required in array.

**b).** Accessing an item in arrays can occur in O(1) time whereas O(n) time is required in LinkedList.

**2.**

**a).** O(N2) because each time of adding operation, all the item should be move back one place and there are N times of adding.

**b).** O(N) because there is no need to move items and there are N times of adding, each adding takes O(1).

**3.**

**a).** O(N2) because each time of removing all the item after the removing one should move forward one place and there are N times of removing.

**b).** O(N) because there is no need to move items and there are N times of removing.

**4.**

**a).** O(N2) because for each item in lst1, we need to go through all the items in lst2.

**b).** O(N2) because for each item in lst1, we need to go through all the items in lst2.

**5.**

// I don’t think the code can compile. There is no variable “sum” defined in the function unless sum is a global variable.

**a).** O(N) because we traverse the lst once, and each time it take O(1) to access the current item.

**b).** O(N2) because we traverse the lst once, and each time it take O(N) to access the current item.

**6.**

**a).** O(N2) because removing each item from front requires O(N) time to move each item forward and there are N items so that the total would be O(N2).

**b).** O(N) because removing each item only requires O(1) and each item only be processed once so that the total would be O(N).

**7.**

Steps:

(1). initialize an empty stack and an empty string named result.

(2). traverse the characters in the formula (assume the formula is given in string format). Let i be the position index in the traversal.

(3). i = 0, result append ‘a’ becoming “a”.

(4). i = 1, push ‘+’ to the stack becoming [+].

(5). i = 2, result append ‘b’ becoming “ab”.

(6). i = 3, push ‘\*’ to the stack becoming [+, \*].

(7). i = 4, result append ‘c’ becoming “abc”.

(8). i = 5, pop the ‘\*’ and ‘+’ from the stack, then result append the two becoming “abc\*+”, then push ‘+’ to the stack becoming [+].

(9). i = 6, push ‘(‘ to the stack becoming [+, (].

(10). i = 7, result append ‘d’ becoming “abc\*+d”.

(11). i = 8, push ‘-‘ to the stack becoming [+, (, -].

(12). i = 9, result append ‘e’ becoming “abc\*+de”.

(13). i = 10, pop ‘-‘ from the stack, then result append the ‘-‘ becoming “abc\*+de-“, then pop ‘(‘ from the stack becoming [+].

(14). end of traversal, pop all item from the stack and appending them into result consecutively and result becoming “abc\*+de-+”.

(15). return the result.