CISC 610-90- O-2018/Late Fall
Extra Credit
Time space complexity analysis

1 algorithm A

Initializing an empty `DLinkedList` only creates two nodes and one int number, thus only costs constant time and space, so the time complexity in this step is

$$O(1), \tag{1}$$

and the space complexity is also

$$O(1). \tag{2}$$

Next step is to traverse and add each number in array to the first of list. Since `addFirst` adds a node to the beginning of the list, and only costs constant time, the whole loop complexity is proportional to n. That is, in this step, time complexity is

$$O(n), \tag{3}$$

and space complexity is

$$O(n) \tag{4}$$

Next, print the list elements from the beginning, `printNextList`. This step traverses every elements in list, but do not need extra space. So the time complexity is

$$O(n) \tag{5}$$

Similarly, `printPrevList` also traverses every elements in list, from tail to head this time, but no extra space needed, and the time complexity is

$$O(n) \tag{6}$$

Summing up (1) (3) (5) and (6) leads to the time complexity of algorithm A:

$$O(1 + n + n + n) = O(1 + 3n) = O(n). \tag{7}$$

The space complexity is the sum of (2) and (4):

$$O(1 + n) = O(n). \tag{8}$$

2 algorithm B

Getting the size of list, `count`, only returns the integer size, which is constant time and space operation. The time complexity is

$$O(1), \tag{9}$$

and the space complexity is

$$O(1). \tag{10}$$

Next, initializing all values in the $n \times n$ array would take both time and space to go through all elements in the array. That is, the time complexity is

$$O(n^2), \tag{11}$$

and the space complexity is

$$O(n^2). \tag{12}$$

In the dual loop, `deleteFirst` simply delete the first element, which is constant operation, and no extra space needed. Since i only loops once, the dual loop actually acts as a single loop, the time complexity is

$$O(n), \tag{13}$$

and no extra space needed. Now call the `print` function. The function uses a dual loop to traverse the whole array and print all elements, which doesn't need extra space, but consumes time as

$$O(n^2). \tag{14}$$

In sum, summing up (9) (11) (13) and (14), one can get the time complexity of algorithm B:

$$O(1 + n^2 + n + n^2) = O(2n^2 + n) = O(n^2), \tag{15}$$

while the space complexity can be obtained by the sum of (10) and (12):

$$O(1 + n^2) = O(n^2). \tag{16}$$

3 algorithm C

Getting the size of list, `count`, only returns the integer size, which is constant time and space operation. The time complexity is

$$O(1), \tag{17}$$

and the space complexity is

$$O(1). \tag{18}$$

Initializing is the same of initializing a `DLinkedList`, which only costs constant time and space. The time complexity in this step is

$$O(1), \tag{19}$$

and the space complexity is also

$$O(1). \tag{20}$$

In the loop of i, the list first calls `deleteLast`, and only costs $O(1)$ time to delete the last without extra space. Then push, the complexity is the same as `DLinkedList.addFirst`, which just add one element to the head and both time and space consumption are constant. Therefore, the time complexity in the loop is

$$O(n), \tag{21}$$

and the complexity of time in the loop is also

$$O(n), \tag{22}$$

Then, we run the `action` function. First step is to get the stack size. It returns the size integer, so just a constant time and space operation. The time complexity is

$$O(1), \tag{23}$$

and the space complexity is

$$O(1). \tag{24}$$

Then an array is initialized, both linear time and space are needed. Then the time complexity is

$$O(n), \tag{25}$$

and space complexity is

$$O(n) \tag{26}$$

For the i loop to pop the stack. Since each `pop` is constant time operation, the whole loop time complexity is

$$O(n), \tag{27}$$

with no extra space. Next, we call print in each iteration of j loop. In the print function, `print` is called in each h loop which traverses from 0 to j. Therefore, the time cost in each print function is $O(j)$, and the time complexity in the j loop is

$$O(0 + 1 + ... + n - 1 + n) = O(n^2/2) = O(n^2), \tag{28}$$

with no extra space needed. In sum, one can get Algorithm C's time complexity by adding up (17) (19) (21) (23) (25) (27) and (28):

$$O(1 + 1 + n + 1 + n + n + n^2) = O(n^2), \tag{29}$$

while the space complexity can be get by adding (18) (20) (22) (24) and (26)

$$O(1 + 1 + n + 1 + n) = O(n). \tag{30}$$