#### HW<sub>2</sub>

صادق حایری ۸۱۰۱۹۴۲۹۸

### سوال ۱:

```
الف)
```

تا وقتی به آخری نرسیده میگیم بعدیتو برعکس کن و در آخر جوابشو برگردون که اونی که برگردونده میشه در اصل نود آخر ماست که آدرسش ریترن شده!

```
1
     reverse( head ) {
 2
         if( head->next ) {
 3
             tmp = revese( head->next )
 4
             head->next->next = head
 5
             head->next = null
 6
              return tmp
 7
         } else {
              return head
 9
         }
10
     }
```

```
ب)
```

توی وایل اول همه رو توی استک پوش میکنیم و آدرس بعدیشونو نال میذاریم (واسه نود آخر)، بعد توی وایل دوم دونه دونه میاریمشون بیرون و آدرس رو برعکس حالت قبل قرار میدیم و آخرش هم نودی که آخر از همه بوده رو به عنوان اولی برمیگردونیم.

```
reverse( head ) {
1
 2
         stack s
 3
         while( head ) {
 4
 5
             s.push( head )
             tmp = head
 6
 7
             head = head->next
             tmp->next = null
 8
 9
         }
10
11
         lastNode = s.top()
12
13
         p, n = null
14
         while( !s.empty() ) {
15
             p = s.pop()
16
             p->next = n
17
             n = p
18
         }
19
20
         return lastNode
21
```

### سوال ۲:

# روش اول (الگوريتم –Floyd's cycle) finding)

۲ پوینتر ابتدایی را در نظر میگیریم و یکی را با سرعت ۱ و دیگری را با سرعت ۲ حرکت میدهیم، اگر به آخر رسیدیم درست است ولی اگر به هم رسیدن یعنی لینکالیست ما مشکل داشته است.

O(n) پیچیدگی زمانی: O(1)

```
// Floyds cycle-finding algorithm
 3
    hasLoop( head ) {
 4
 5
        if( !head )
 6
             return False;
 8
        slow = fast = head;
 9
        while( True ) {
10
            if( !fast || !fast->next )
                 return False;
14
            if( fast == slow )
15
                 return True;
            slow = slow->next
17
                                         // 1 hop
            fast = fast->next->next
                                         // 2 hops
19
        }
20
```

## روش دوم (استفاده از مارک کردن نودها)

فرض میکنیم نودهای ما یک بولین seen دارند (یا یک لینکلیست جدا با این قابلیت میسازیم)، حال هر نود که از روی آن رد میشویم سین میکنیم و به بعدی میرویم، اگر به خانه نال رسیدیم و کسی که سین کرده بودیم دوباره ندیدیم یعنی درست است ولی اگر در بین راه به کسی که سین کرده باشیم برسیم نتیجه میگیریم که لینکلیست خراب است.

پیچیدگی زمانی: (O(n

پیچیدگی حافظه: (O(n

## سوال ۳:

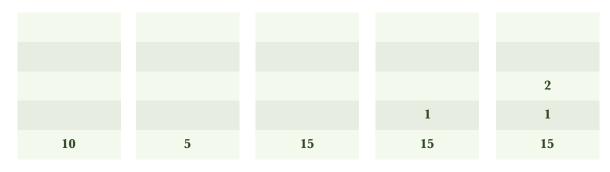
الف)

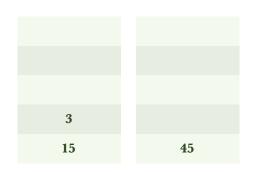
 $pre-order: +\ (*\ 2\ 3)\ (\%\ 12\ 4)$ 

				2
	12		3	3
4	4	0	0	0

6	
0	6

 $post-order: (10\ 5\ +)(1\ 2\ +)\ *$ 





# $pre-order:-/**3\ 4\ 2\ 6\ 4$

				3
			4	4
		2	2	2
	6	6	6	6
4	4	4	4	4
12				
2	24			
6	6	4		
4	4	4	0	

### $post-order:\ 4\ 10\ 2\ 11\ 11\ *+*+$

				11
			11	11
		2	2	2
	10	10	10	10
4	4	4	4	4
121				
2	123			
10	10	1230		
4	4	4	1234	

```
validate( string )
 2
         for char in string {
 3
 4
             if( char == '(' )
 5
                 s.push( '(')
 6
 7
             else if( char == ')')
 8
                      s.pop()
9
                 else
10
                      return False
11
12
         }
13
         return True if s.empty() else False
14
```

```
سوال۵ :
```

ب)

یاب نشود.

هربار به پرانتز باز رسیدیم میریزیم توی استک و اگر

به یرانتز بسته رسیدیم از استک بیرون میآوریم،

اگر منظم باشد استک باید خالی شود و اضافه تر هم

```
1 int getMaxArea(int hist[], int n)
    {
        // Create an empty stack. The stack holds indexes of hist[] array
4
        stack<int> s;
        int max_area = 0; // Initalize max area
        int tp; // To store top of stack
 6
        int area_with_top; // To store area with top bar as the smallest bar
        // Run through all bars of given histogram
9
        int i = 0;
10
        while (i < n)
            // If this bar is higher than the bar on top stack, push it to stack
            if (s.empty() || hist[s.top()] <= hist[i])</pre>
14
                s.push(i++);
            // If this bar is lower than top of stack, then calculate area of rectangle
            // with stack top as the smallest (or minimum height) bar. 'i' is
            // 'right index' for the top and element before top in stack is 'left index'
18
            else
            {
20
                tp = s.top(); // store the top index
                s.pop(); // pop the top
                // Calculate the area with hist[tp] stack as smallest bar
                area_with_top = hist[tp] * (s.empty() ? i : i - s.top() - 1);
24
                // update max area, if needed
                if (max_area < area_with_top)</pre>
                    max_area = area_with_top;
            }
        }
30
        // Now pop the remaining bars from stack and calculate area with every
        // popped bar as the smallest bar
        while (s.empty() == false)
34
            tp = s.top();
            s.pop();
            area_with_top = hist[tp] * (s.empty() ? i : i - s.top() - 1);
            if (max_area < area_with_top)</pre>
                max_area = area_with_top;
41
42
        return max_area;
43 }
```