HW2

صادق حایری
۸۱۰۱۹۴۲۹۸

سوال ۱:

الف)

```
reverse( head ) {
    if( head->next ) {
        tmp = revese( head->next )
        head->next->next = head
        head->next = null
        return tmp
    } else {
        return head
    }
}
```

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!

ب)

```
reverse( head ) {
    stack s

    while( head ) {
        s.push( head )
        tmp = head
        head = head->next
        tmp->next = null
    }

    lastNode = s.top()

    p, n = null
    while( !s.empty() ) {
        p = s.pop()
        p->next = n
        n = p
    }

    return lastNode
}
```

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.

سوال ۲:

روش اول (الگوریتم Floyd's cycle–finding)

```
// Floyds cycle-finding algorithm

hasLoop( head ) {

    if( !head )
        return False;

    slow = fast = head;

    while( True ) {
        if( !fast || !fast->next )
            return False;

        if( fast == slow )
            return True;

        slow = slow->next           // 1 hop
        fast = fast->next->next     // 2 hops
    }

}
```

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 لینک‌لیست ما مشکل داشته است.

پیچیدگی زمانی: $O(n)$

پیچیدگی حافظه: $O(۱)$

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.

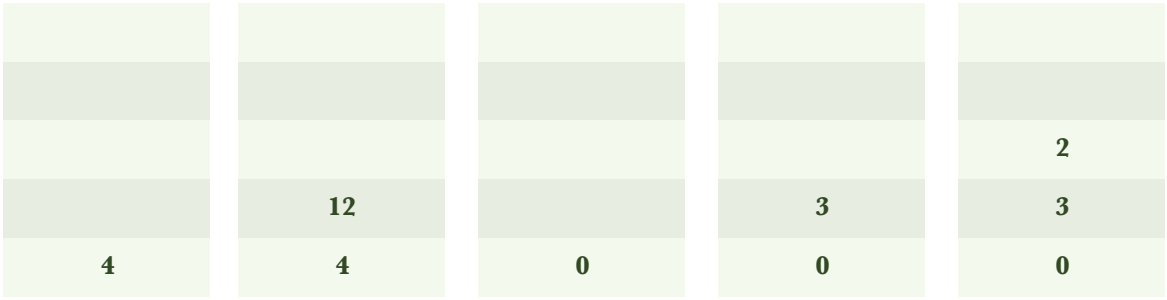
پیچیدگی زمانی: $O(n)$

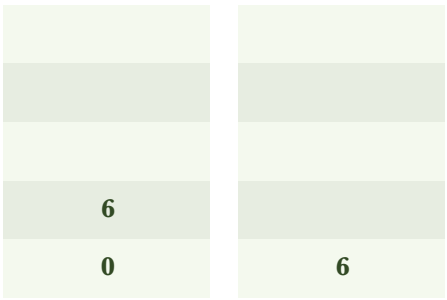
پیچیدگی حافظه: $O(n)$

سوال ۳:

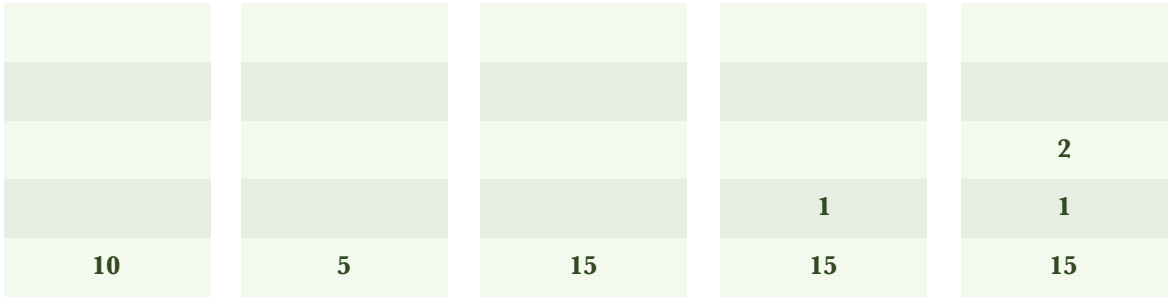
الف)

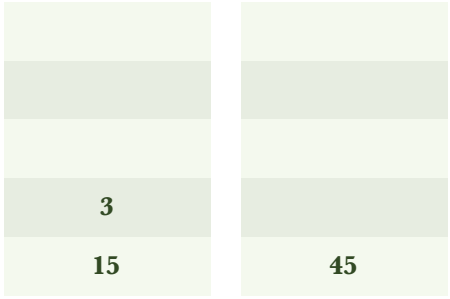
$pre - order : + \ (* \ 2 \ 3) \ (\% \ 12 \ 4)$





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





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

				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

ب)

هربار به پرانتز باز رسیدیم میریزیم توی استک و اگر به پرانتز بسته رسیدیم از استک بیرون می‌آوریم، اگر منظم باشد استک باید خالی شود و اضافه تر هم پاپ نشود.

```
validate( string )
    for char in string {

        if( char == '(' )
            s.push( '(' )

        else if( char == ')')
            if( !s.empty() )
                s.pop()
            else
                return False

    }
    return True if s.empty() else False
}
```

سوال۵ :

```
int getMaxArea(int hist[], int n)
{
    // Create an empty stack. The stack holds indexes of hist[] array
    stack<int> s;
    int max_area = 0; // Initalize max area
    int tp;  // To store top of stack
    int area_with_top; // To store area with top bar as the smallest bar
    // Run through all bars of given histogram
    int i = 0;
    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])
            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'
        else
        {
            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);

            // update max area, if needed
            if (max_area < area_with_top)
                max_area = area_with_top;
        }
    }
    // Now pop the remaining bars from stack and calculate area with every
    // popped bar as the smallest bar
    while (s.empty() == false)
    {
        tp = s.top();
        s.pop();
        area_with_top = hist[tp] * (s.empty() ? i : i - s.top() - 1);

        if (max_area < area_with_top)
            max_area = area_with_top;
    }
    return max_area;
}
```