بهداد اعتضادی سری ۲ ۹۹۱۳۰۵۳

۱-

الف:

برای رجوع عنصر در موقعیت فعلی، ما نیاز به دسترسی به عنصر قبلی داریم که آن را به عنصر جدید ارجاع دهیم اما در SlList امکان رفتن به عقب وجود ندارد پس برای اینکار نیاز داریم از اول لیست شروع کنیم پس اگر خانه nام بودیم باید از اول لیست n-1 بار حرکت کنیم.

ب: حذف آخرین عنصر در SlList به تعداد عناصر وابسته است زیرا پیمایشگر برای ارجاع tail به عنصر کی مانده به آخر نیازمند هستیم که از اول لیست شروع به حرکت کنیم تا عنصر کی مانده به آخر جلو بریم.

۲- الف:

```
Node * f = dummy.next; int check = 0;
Node * b = dummy.prev;
for (int i = 1 ; i <= n/2 ; i++)
{
  if ( (f -> value) == (b -> value))
      {
        f = f -> next;
        b = b -> prev;
      }
  else { print (not palindrome); check = +1) ; break;}
}
 if (check != (-1)) {print(is palindrome);}
```

worst case: $O(n/2) \subset O(n)$

عامل اعمال انجام شده مقدار ثابتی وقت گرفته یک حلقه داریم که در بدترین حالت (عدد اتفاق نیافتد) $n/2$ بار می چرخد.

ب)-٢

```
if (k==(n-k+1))
    pass;
else if( k<=n and k>0 )
    Node * kth_first = head ;
    Node * before_first , *before_last ;
    Node * kth_last = head ;
    for(int i=1 ; i< k ; i++)
        ~ before_first = kth_first ;
        ~ kth_first = kth_first -> next ;
    for(int i=1 ; i<=(n-k) ;i++)
        ~ before_last = kth_last;
        ~ kth_last = kth_last -> next;
    Node * reserve ;
    if ( before_first != null)
        ~ before_first -> next = kth_last ;
    if ( before_last != null )
        ~ before_last -> next = kth_first;
    reserve = kth_first -> next;
    kth_first -> next = kth_last -> next;
    kth_last -> next = reserve ;
    if ( k == 1) { head = kth_last ; tail = kth_first; }
    if(k==n) { head = kth_last ; tail = kth_first; }
```

تحلیل ۲ ب:

تمام اعمال در زمان ثابت هستند فقط ۲ حلقه داریم که در صورت وقوع else ، k-1 و n-k بار می چرخند. پس در مجموع n-1 بار می چرخند پس درست پس اکسپکتنس ما O(n) می باشد

```
Node { int data; Node* next;}
class list_stack
{  public:
     Node head, tail;
     list_stack max_reserve, min_reserve;
     int size = 0;
     int push (int x)
     {
       Node to_push = new Node;
       if ( size == 0 )
         {  tail = to_push; max_reserve.push(x);
            min_reserve.push (x);
         }
       to_push.next = head;
       head = to_push;
       if ( getmax() <= x )
            max_reserve.push (x);
      if ( getmin() >= x)
           min_reserve.push(x);
      size++;

     }
```

```
int pop()
{
    if (size == 0)
        return NULL;
    int to_return;
    to_return = head.data;
    Node to_remove;
    head = head.next;
    delete to_remove;
    size--;
    if(size == 0)
        tail = NULL;
    if (get_max() == to_return)
        max_reserve.pop();
    if (get_min() == to_return)
        min_reserve.pop();
    return to_return;
}
```

```
int get_max()
{
    return max_reserve.head.data;
}
int get_min()
{ return min_reserve.head.data; }
}
```

توضیح: کلاس List_stack، مانند SLList، از node ها به هم وصل شده از fail تا head تشکیل شده و دارای متدهای است است یعنی push، pop، ۲ لیست_استک را نگه می دارد که یکی برای انبار مینیمم ها، دیگری برای انبار ماکسیمم ها است.

۴-

```
Reverse (L):
    if L = null then return L
    r ← Reverse (next[L])
    next[next[L]] ← L
    next[L] ← null
    return r
```

- در داده شده غلط است نه تنها head را آپدیت می‌کند وقتی به انتهای لیست می‌رسد هم هربار از next[null] استفاده می‌کند اروری خواهد داد

همچنین جای L و r را برمی‌گرداند حالت درست و کامل آن به شکل زیر است:

```
if L = null then return null
if next[L] = null then head ← L , return L
 r ← reverse (next[L])
  next[r] ← L
  next[L] ← null
  return L
```