Lecture 11

By Urooj Ainuddin

CS-218 Data Structures and Algorithms

In the last lecture…

We were introduced to singly linked lists.

We implemented a class to create and manage a singly linked list.

We saw pictorial depictions of code fragments.

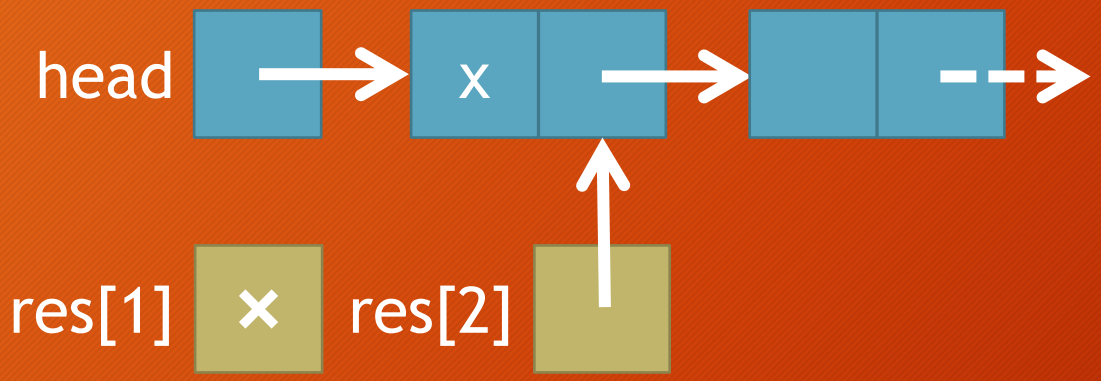
We found time complexities of all codes.

Linked Structures

Book 1 Chapter 6

Inserting a node containing val after the node containing x

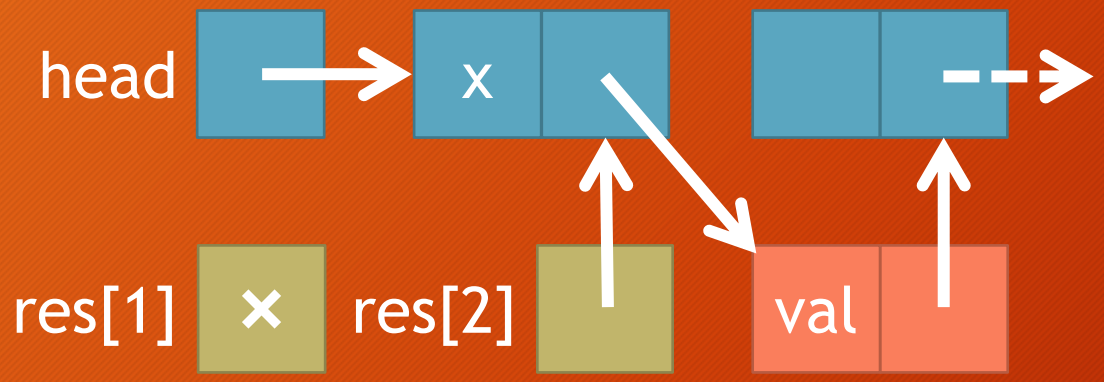
```
from singlylinkedlist import ListNode

def insafter(head,x,val):
    res=head.search(x)
    if res[0]==True:
        res[2].insert(val)
```



- The best case happens when the first node contains x.
- The search takes $O(1)$ time.
- res[0] = True
- res[1] = None
- res[2] contains the pointer to the node that contains x.

Inserting a node containing val after the node containing x

```
from singlylinkedlist import ListNode

def insafter(head,x,val):
    res=head.search(x)
    if res[0]==True:
        res[2].insert(val)
```

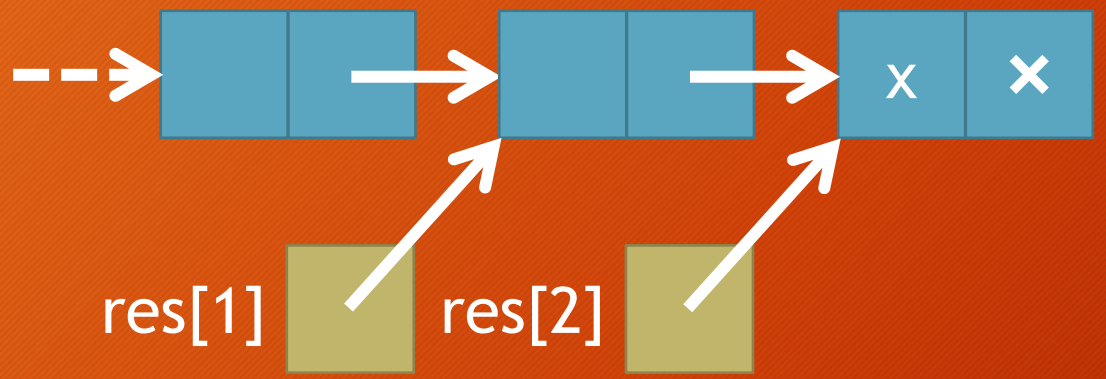


- The best case happens when the first node contains x.
- The search takes $O(1)$ time.
- res[0] = True
- res[1] = None
- res[2] contains the pointer to the node that contains x.
- We use insert to add a node containing val after the node containing x.
- The insertion takes $O(1)$ time.

$O(1)$

Inserting a node containing val after the node containing x

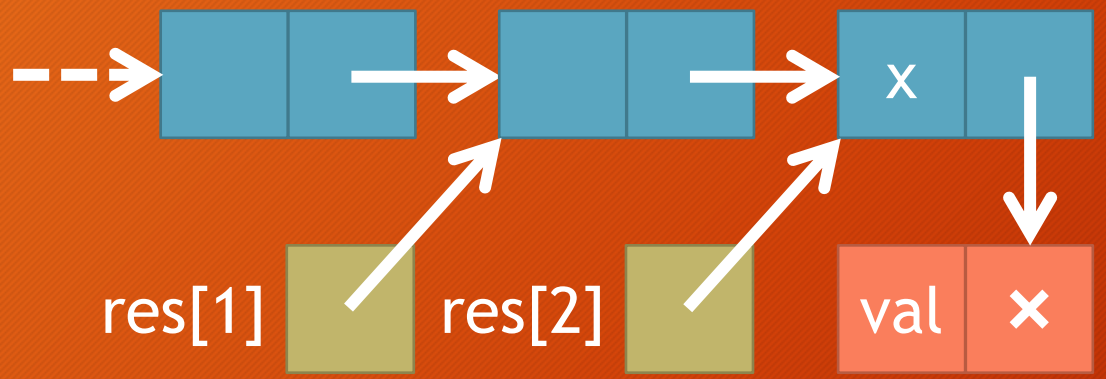
```
from singlylinkedlist import ListNode

def insafter(head,x,val):
    res=head.search(x)
    if res[0]==True:
        res[2].insert(val)
```



- The worst case happens when the last node contains x.
- The search takes $O(n)$ time.
- res[0] = True
- res[2] contains the pointer to the node that contains x.

Inserting a node containing val after the node containing x

```
from singlylinkedlist import ListNode

def insafter(head,x,val):
    res=head.search(x)
    if res[0]==True:
        res[2].insert(val)
```



- The worst case happens when the last node contains x.
- The search takes $O(n)$ time.
- res[0] = True
- res[2] contains the pointer to the node that contains x.
- We use insert to add a node containing val after the node containing x.
- The insertion takes $O(1)$ time.

$O(n)$

Inserting a node containing val after the node containing x

```
from singlylinkedlist import ListNode

def insafter(head,x,val):
    res=head.search(x)
    if res[0]==True:
        res[2].insert(val)
```

- Let us consider a linked list in which no node contains x.
- The search takes $O(n)$ time.
- res[0] = False

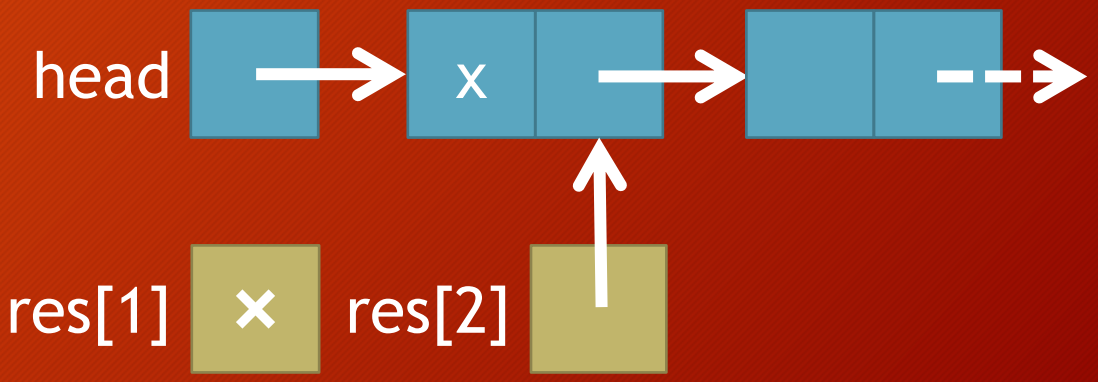
Do you think this is one of the worst cases?

$O(n)$

Inserting a node containing val before the node containing x

```
from singlylinkedlist import ListNode

def insbefore(head,x,val):
    res=head.search(x)
    if res[0]==True:
        if res[2] is head:
            new=ListNode(val)
            new.next=head
            head=new
        else:
            res[1].insert(val)
    return head
```

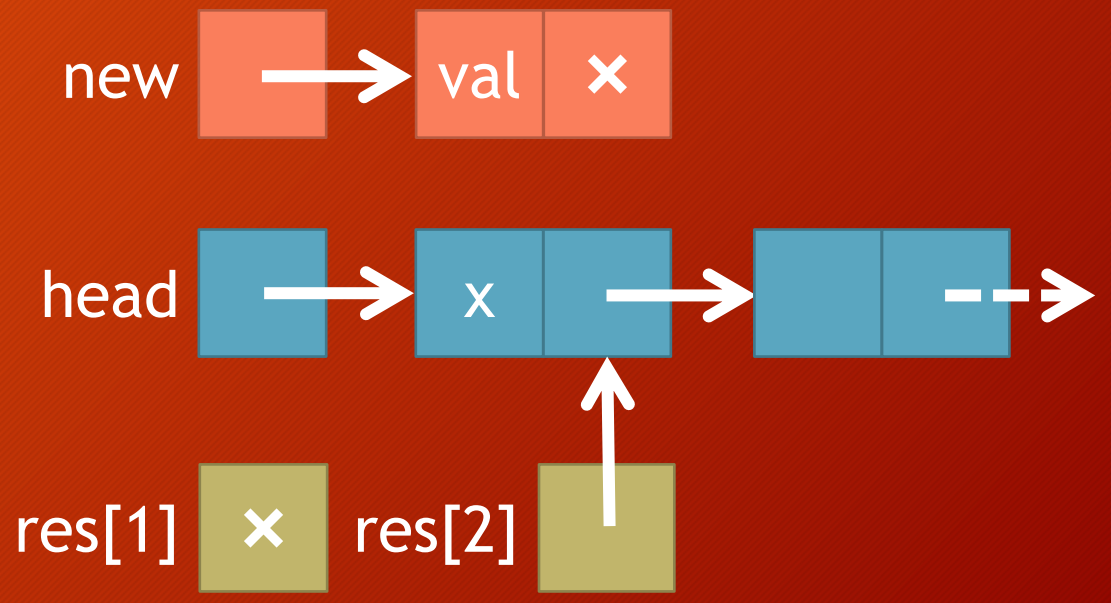
- The best case happens when the first node contains x.
- The search takes $O(1)$ time.
- res[0] = True
- res[1] = None
- res[2] contains the pointer to the node that contains x.



Inserting a node containing val before the node containing x

```
from singlylinkedlist import ListNode

def insbefore(head,x,val):
    res=head.search(x)
    if res[0]==True:
        if res[2] is head:
            new=ListNode(val)
            new.next=head
            head=new
        else:
            res[1].insert(val)
    return head
```

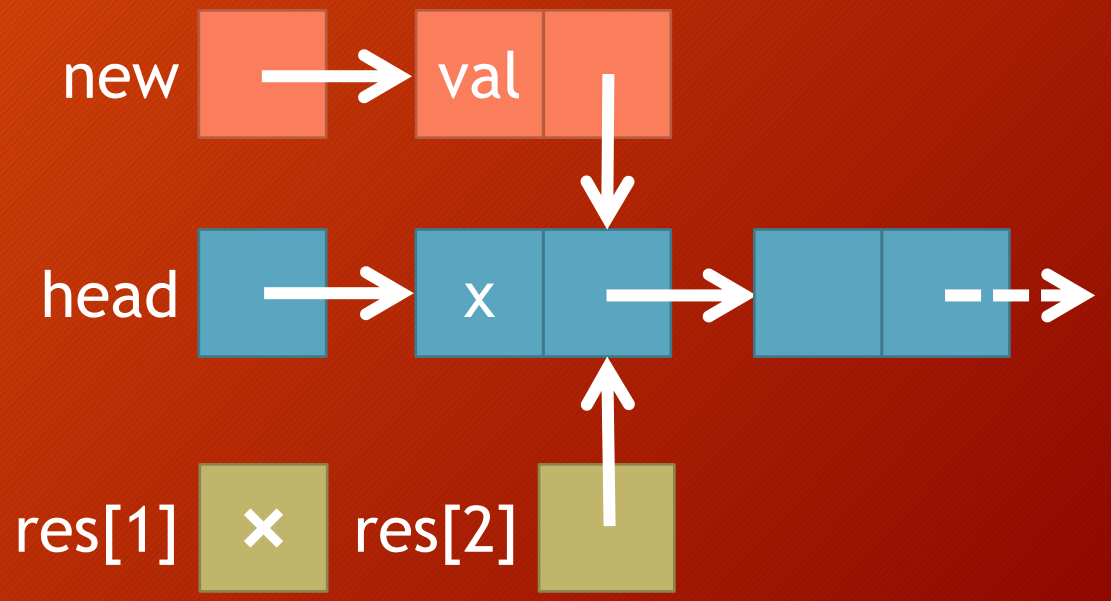
- We create a node containing val and a pointer called new points to it.



Inserting a node containing val before the node containing x

```
from singlylinkedlist import ListNode

def insbefore(head,x,val):
    res=head.search(x)
    if res[0]==True:
        if res[2] is head:
            new=ListNode(val)
            new.next=head
            head=new
        else:
            res[1].insert(val)
    return head
```

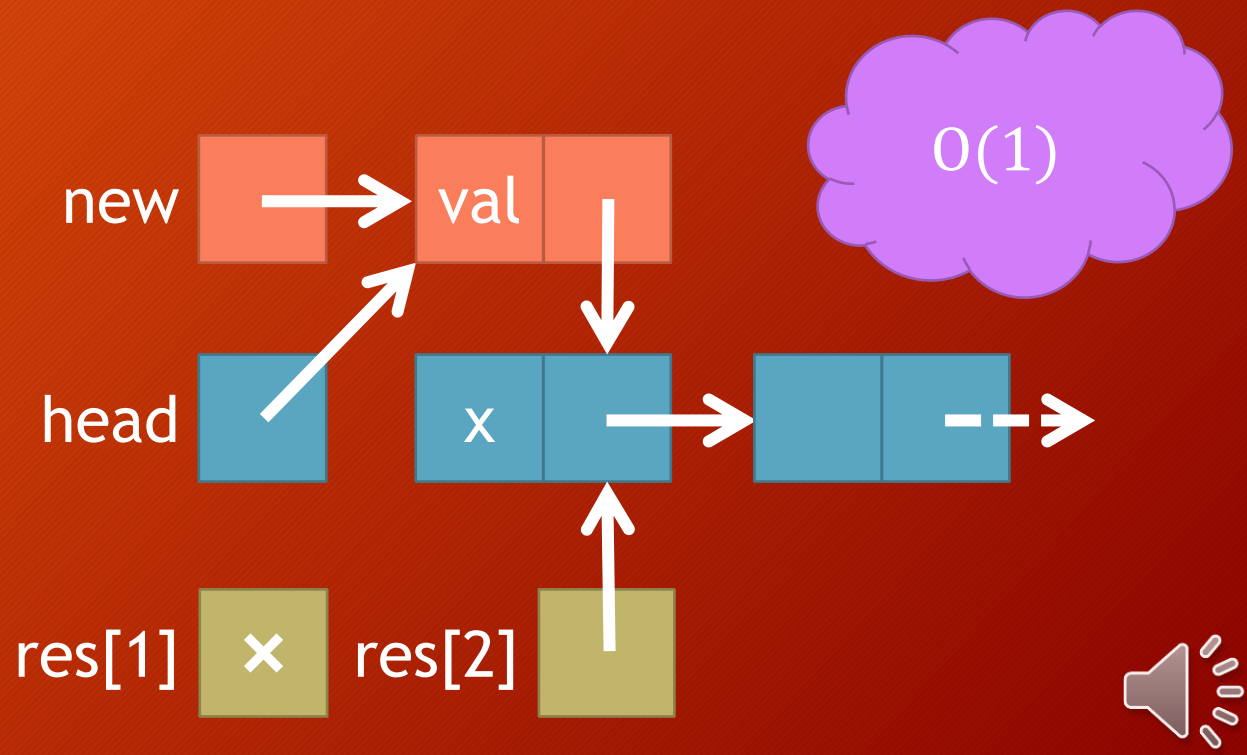
- We create a node containing val and a pointer called new points to it.
- The next field of this node points to the node containing x.



Inserting a node containing val before the node containing x

```
from singlylinkedlist import ListNode

def insbefore(head,x,val):
    res=head.search(x)
    if res[0]==True:
        if res[2] is head:
            new=ListNode(val)
            new.next=head
            head=new
        else:
            res[1].insert(val)
    return head
```

- head now points to the newly created node.
- The if constructs take $O(1)$ time.



Inserting a node containing val before the node containing x

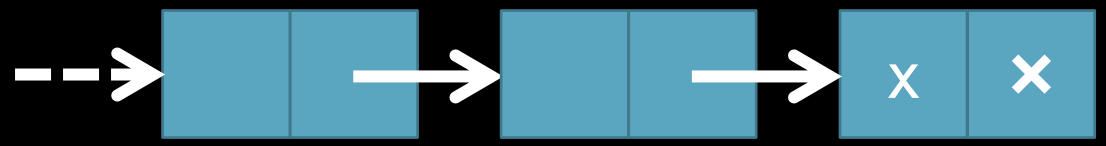
```
from singlylinkedlist import ListNode

def insbefore(head,x,val):
    res=head.search(x)
    if res[0]==True:
        if res[2] is head:
            new=ListNode(val)
            new.next=head
            head=new
        else:
            res[1].insert(val)
    return head
```

- The worst case happens when no node contains x.
- The search takes $O(n)$ time.
- res[0] = False

$O(n)$

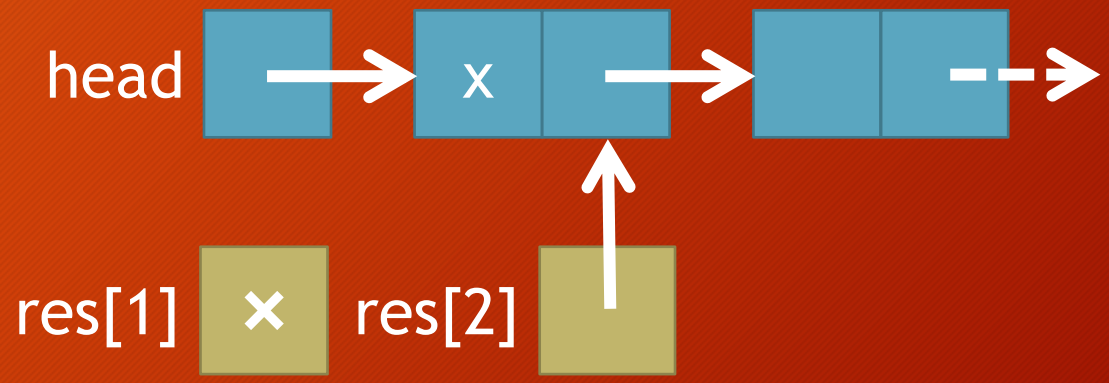
Homework

1. We want to insert a node before the node containing x in the singly linked list shown here. Run the insbefore function. Draw figures for all steps. Can this be considered the function's best or worst case?



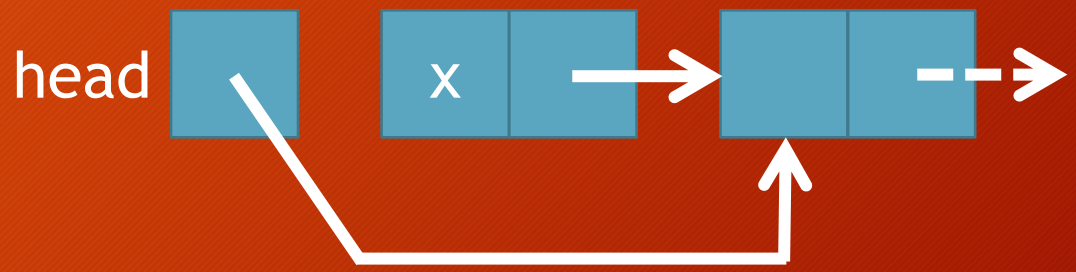
Deleting a node containing x

```
from singlylinkedlist import ListNode

def delnode(head,x):
    res=head.search(x)
    if res[0]==True:
        if res[2] is head:
```



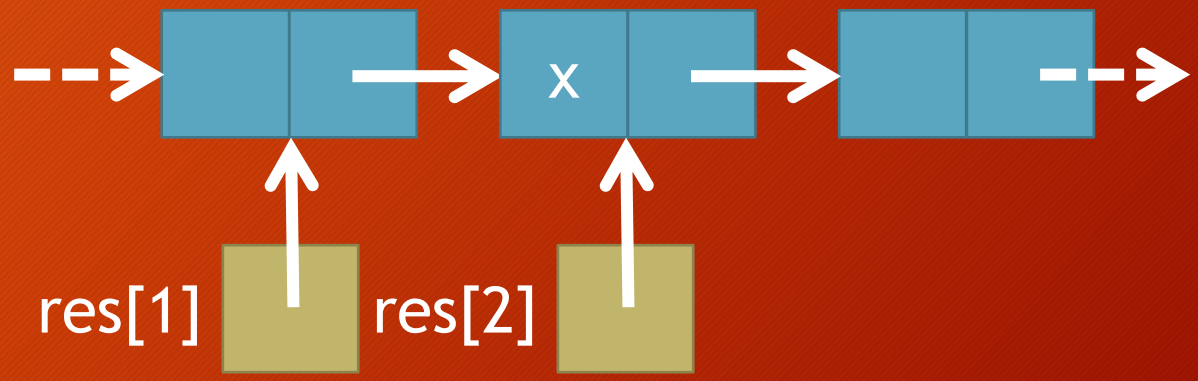
Deleting a node containing x

```
from singlylinkedlist import ListNode

def delnode(head,x):
    res=head.search(x)
    if res[0]==True:
        if res[2] is head:
            head=head.next
```



Deleting a node containing x

```
from singlylinkedlist import ListNode

def delnode(head,x):
    res=head.search(x)
    if res[0]==True:
        if res[2] is head:
            head=head.next
        else:
```



Deleting a node containing x

```
from singlylinkedlist import ListNode

def delnode(head,x):
    res=head.search(x)
    if res[0]==True:
        if res[2] is head:
            head=head.next
        else:
            res[1].delete()
    return head
```

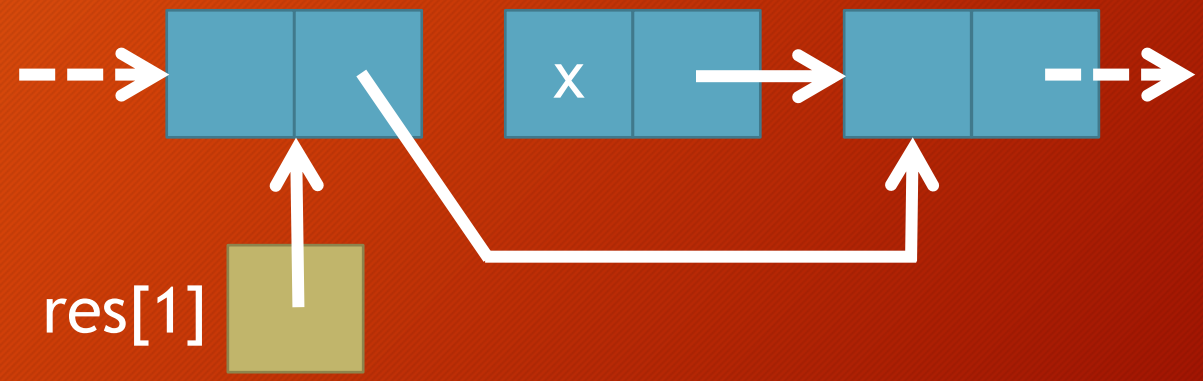
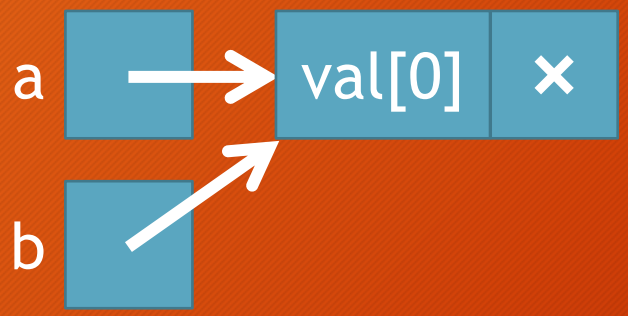


Figure out the best and worst cases for this function and their time complexities.

Building a singly linked list

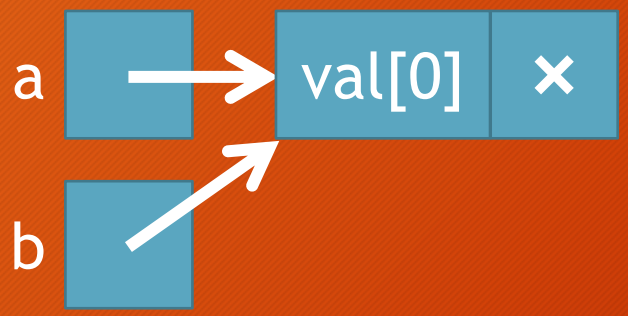
```
from singlylinkedlist import ListNode

def buildlist(val):
    assert len(val)>0,“no elements"
    a=ListNode(val[0])
    b=a
```



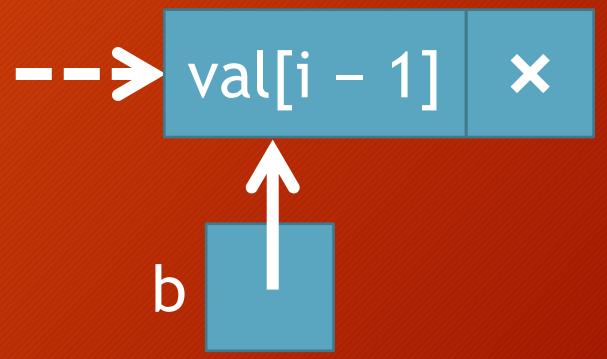
- val is a Python list containing the data we want to store in a singly linked list.
- val must have one or more elements.
- We create a node and put val[0] in it.
- We set a pointer b to point to this first node.
- We do not want to advance a from its position, because it is our head pointer.

Building a singly linked list

```
from singlylinkedlist import ListNode

def buildlist(val):
    assert len(val)>0,“no elements"
    a=ListNode(val[0])
    b=a
    for i in range(1,len(val),1):
```



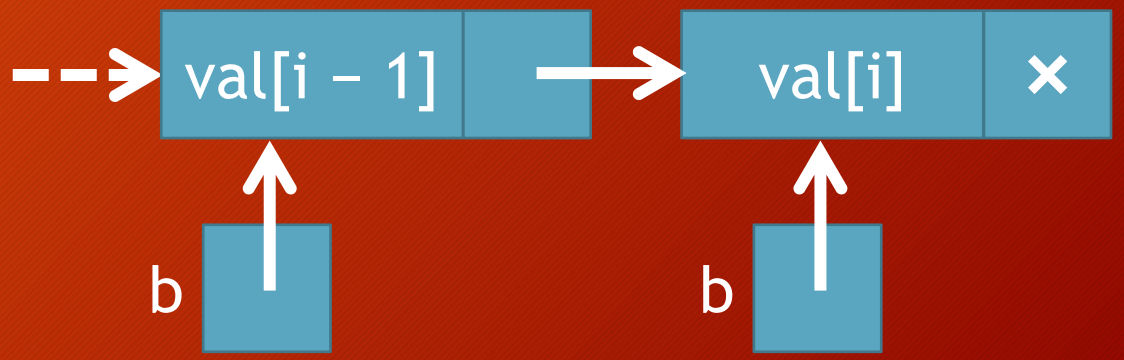
- We run a for loop from 1 to n – 1.
- In the body of the for loop, we insert a node containing val[i] after b.



Building a singly linked list

```
from singlylinkedlist import ListNode

def buildlist(val):
    assert len(val)>0,“no elements"
    a=ListNode(val[0])
    b=a
    for i in range(1,len(val),1):
        b.insert(val[i])
        b=b.next
```

- We run a for loop from 1 to n – 1.
- In the body of the for loop, we insert a node containing val[i] after b.
- We advance b to the node just created.
- When the for loop finishes, we return the head pointer, or a.



Building a singly linked list

```
from singlylinkedlist import ListNode

def buildlist(val):
    assert len(val)>0,“no elements"
    a=ListNode(val[0])
    b=a
    for i in range(1,len(val),1):
        b.insert(val[i])
        b=b.next
    return a
```

- We run a for loop from 1 to n – 1.
- In the body of the for loop, we insert a node containing val[i] after b.
- We advance b to the node just created.
- When the for loop finishes, we return the head pointer, or a.

O(n)

Homework

2. We want to build a linked list, from user input. The function is called buildlist(). Write the body of this function. Take elements as input and keep adding them to a linked list as long as the user wants. Return the head pointer of the list. Analyze the function for time complexity.

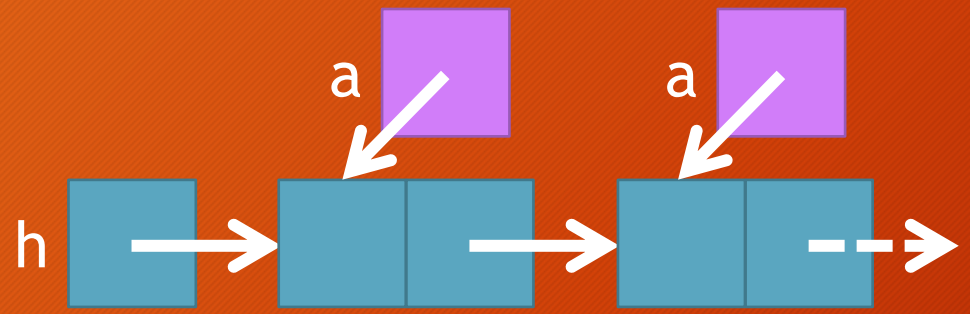
Inserting a tail node containing x

```
from singlylinkedlist import ListNode

def instail(h,x):
    if h is None:
        return ListNode(x)
```

- The pointer h points to the first node of the SLL.
- We want to insert a node containing x at the end of the SLL.
- Situation #1: There is no list.
  - In this case, we will create a node, put x in it, and return its pointer.

Inserting a tail node containing x

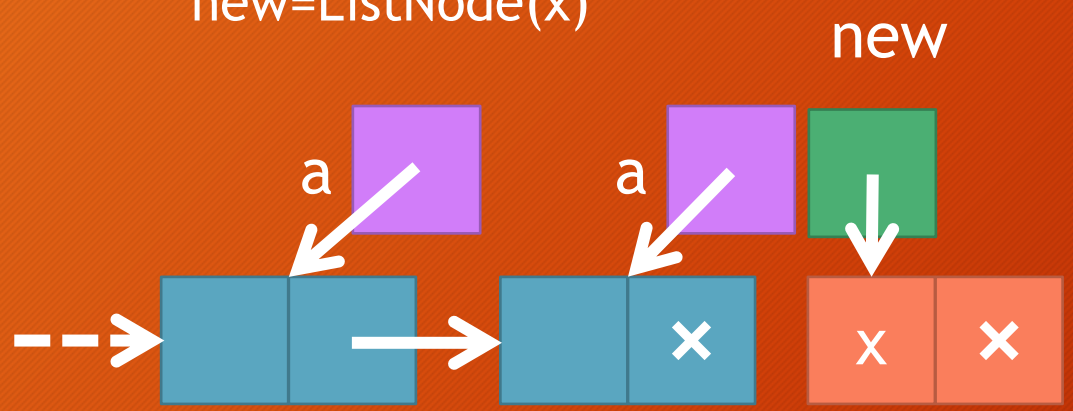
```
from singlylinkedlist import ListNode

def instail(h,x):
    if h is None:
        return ListNode(x)
    a=h
    while a.next is not None:
        a=a.next
```



- Situation #2: The list has some elements.
  - We do not want to advance h from its position, because it is our head pointer.
  - Set a pointer a to point to where h points.
  - We look for the last node, i.e. whose next field is None.
  - As long as we do not find it, we advance a.

Inserting a tail node containing x

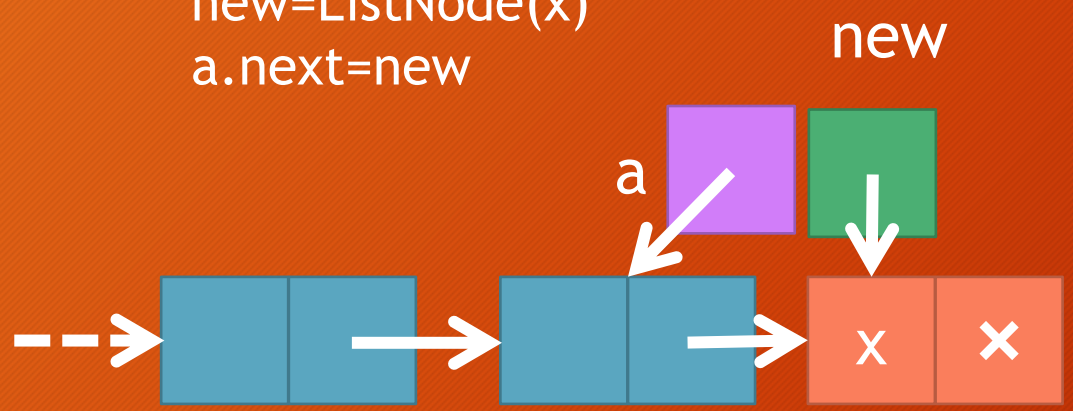
```
from singlylinkedlist import ListNode

def instail(h,x):
    if h is None:
        return ListNode(x)
    a=h
    while a.next is not None:
        a=a.next
    new=ListNode(x)
```



- Situation #2: The list has some elements.
  - We do not want to advance h from its position, because it is our head pointer.
  - Set a pointer a to point to where h points.
  - We look for the last node, i.e. whose next field is None.
  - As long as we do not find it, we advance a.
  - When we find the last node, we make a node containing x.
  - We set the next field of the last node to point to the newly created node.

Inserting a tail node containing x

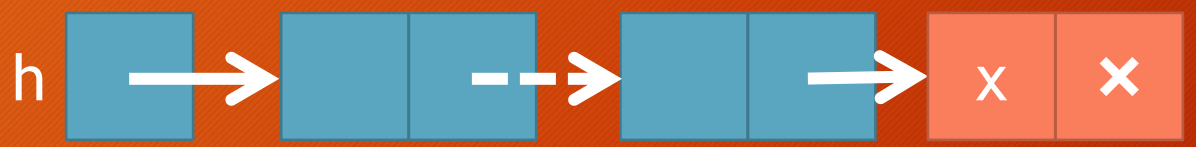
```
from singlylinkedlist import ListNode

def instail(h,x):
    if h is None:
        return ListNode(x)
    a=h
    while a.next is not None:
        a=a.next
    new=ListNode(x)
    a.next=new
```



- Situation #2: The list has some elements.
  - We do not want to advance h from its position, because it is our head pointer.
  - Set a pointer a to point to where h points.
  - We look for the last node, i.e. whose next field is None.
  - As long as we do not find it, we advance a.
  - When we find the last node, we make a node containing x.
  - We set the next field of the last node to point to the newly created node.
  - We return the head pointer.

Inserting a tail node containing x

```
from singlylinkedlist import ListNode

def instail(h,x):
    if h is None:
        return ListNode(x)
    a=h
    while a.next is not None:
        a=a.next
    new=ListNode(x)
    a.next=new
    return h
```



- Situation #2: The list has some elements.
  - We do not want to advance h from its position, because it is our head pointer.
  - Set a pointer a to point to where h points.
  - We look for the last node, i.e. whose next field is None.
  - As long as we do not find it, we advance a.
  - When we find the last node, we make a node containing x.
  - We set the next field of the last node to point to the newly created node.
  - We return the head pointer.

3. Analyze instail(h,x) for time complexity. Will it have different best and worst cases?

Homework

So what did we learn today?

We learned how to insert before or after a node containing a certain value in a SLL.

We learned how to delete a node containing a certain value in a SLL.

We built singly linked lists.

We inserted a tail node in a SLL.

We conducted complexity analysis for the running times of our functions.

Things to do

Read the book!

Note your questions and put them up in the relevant online session.

Email suggestions on content or quality of this lecture at uroojain@neduet.edu.pk