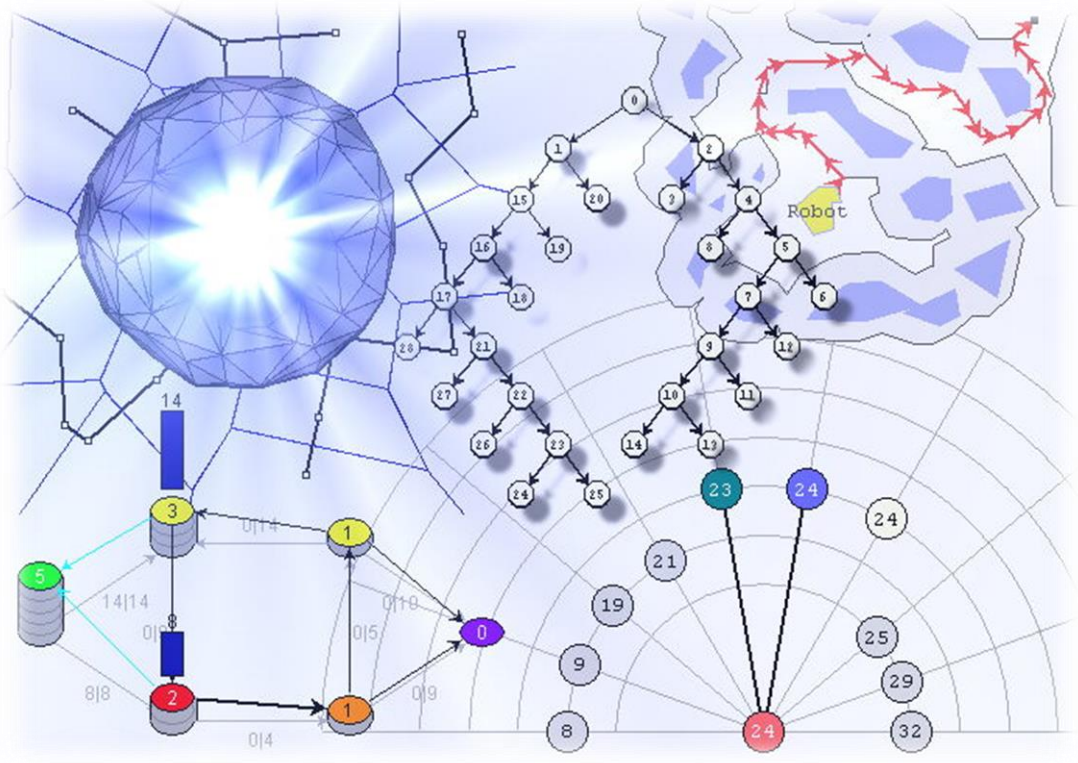
Islamic University – Gaza
Engineering Faculty
Department of Computer Engineering
ECOM 3012: Data Structures and Algorithms Discussion

Chapter 3

Arrays, Linked Lists, and Recursion

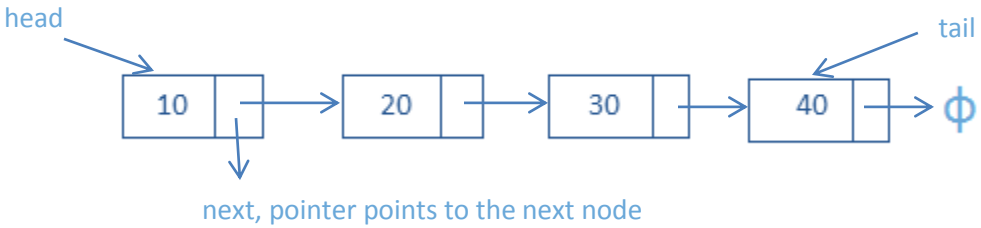


Eng. Eman R. Habib

October, 2013

➢ Singly linked list

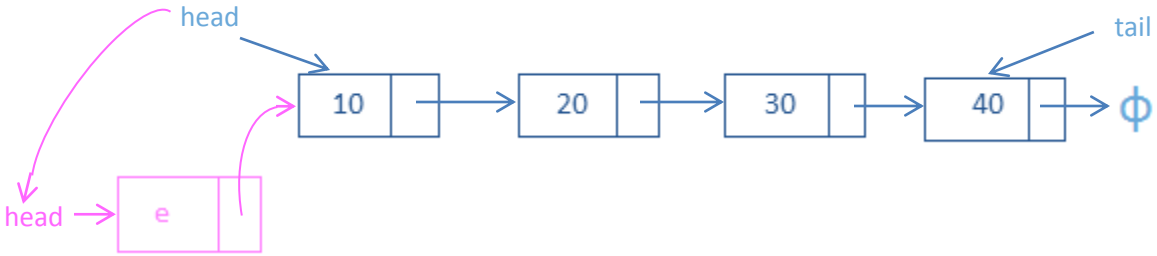
- **Singly linked list :** a collection of nodes that form a liner ordering
- **Link hopping:** moving from one node to another.
- **Singly**: you can move in one direction, from the node to the next one only
- There is no fixed size.



R-3.9:

Describe a method for inserting an element at the beginning of a singly linked list. Assume that the list does not have a sentinel header node, and instead uses a variable head to reference the first node in the list.

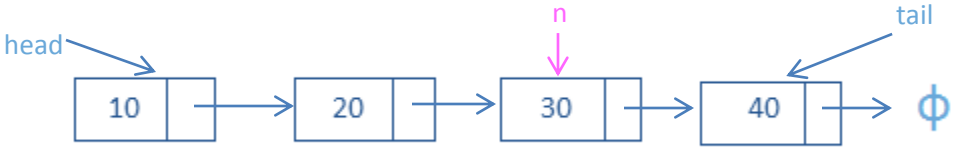
```
Public void insert (String e){
     Node n = new Node();
     n.setElement(e);
     if(size > 0)
          n.setNext(head);
     head = n;
     size++;
}
```



R-3.10:

Give an algorithm for finding the penultimate node in a singly linked list where the last element is indicated by a null next reference.

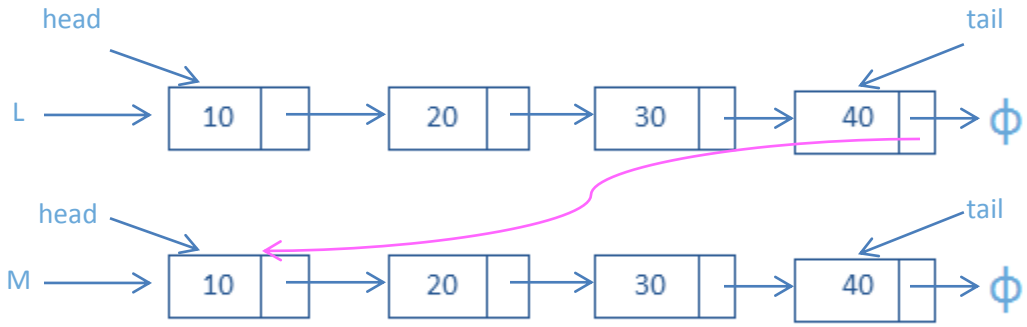
```
Algorithm findPenultimate(S):
Node n ← head
while (n.getNext()!=tail)do
     N ← n.getNext()
return n
```



C-3.8:

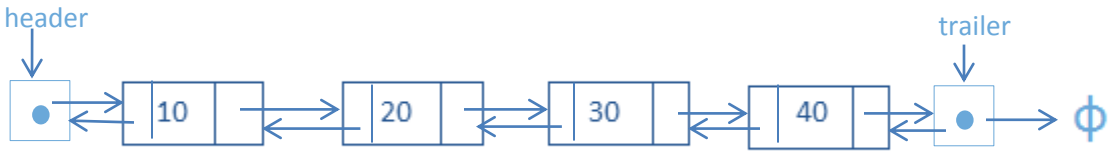
Describe a good algorithm for concatenating two singly linked lists Land M, with header sentinels, into a single list L' that contains all the nodes of L followed by all the nodes of M.

```
Algorithm concatenate(L,M):
Node n ← L.getHead()
While (n.getNext()!=null)do
     n ← n.getNext()
n.setNext(M.getHead())
L′ ← L
```



➢ Doubly linked list

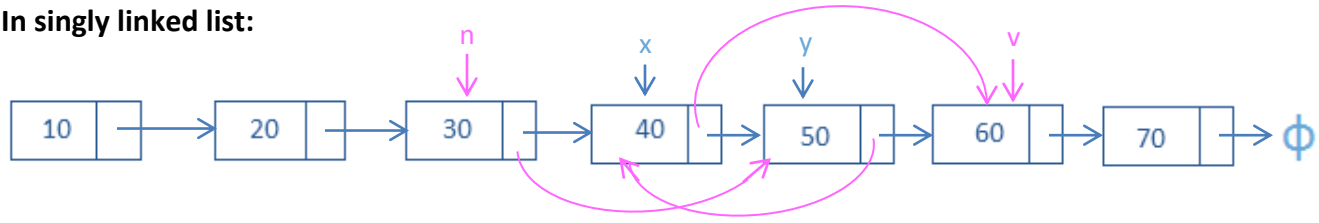
- Each node has two references, one for next and the other for previous.
- DLL has “header” and “trailer” nodes called dummy or sentinel nodes.
- An empty DLL has header and trailer only and its size is zero (not counting sentinel nodes).



C-3.10

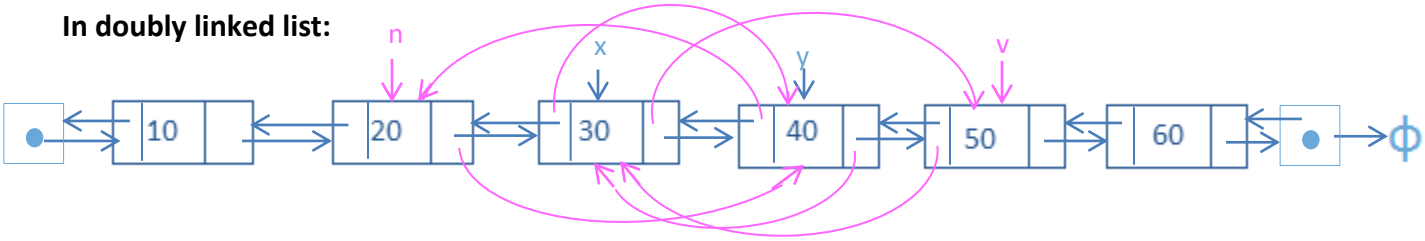
Describe in detail how to swap two nodes *x* and *y* (and not just their contents) in a singly linked list *L* given references only to *x* and *y*. Repeat this exercise for the case when *L* is a doubly linked list. Which algorithm takes more time?

In singly linked list:



```
Algorithm swap(x, y):
Node n ← head
while( n.getNext() != x ) do
     n ← n.getNext()
Node v ← y.getNext()
n.setNext(y)
y.setNext(x)
x.setNext(v)
```

In doubly linked list:

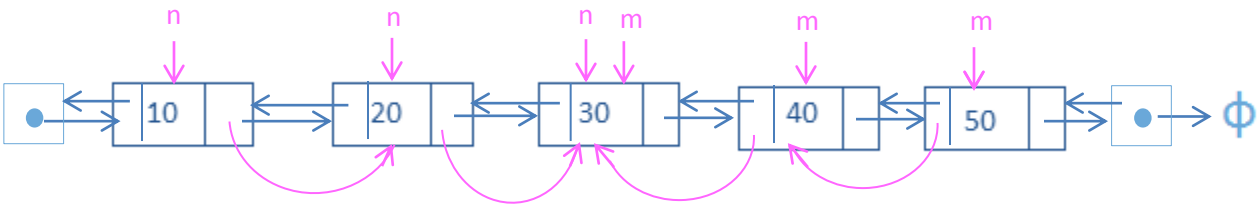



```
Algorithm swapDoubly(x, y):
DNode n ← x.getPrev()
DNode v ← y.getPrev()
n.setNext(y)
y.setPrev(n)
y.setNext(x)
x.setPrev(y)
x.setNext(v)
v.setPrev(x)
```

Swap in singly linked list take more time because we have to move from head to the node before x.

R-3.11

Describe a nonrecursive method for finding, by link hopping, the middle node of a doubly linked list with header and trailer sentinels. (Note: This method must only use link hopping; it cannot use a counter.) What is the running time of this method?




```
DNode findMiddle(){
DNode n = header.getNext();
DNode m = trailer.getPrev();
if(n == trailer)
     return null;
```

```
While (n != m){
      n=n.getNext();
      m=m.getPrev();
}
return m;
}
```

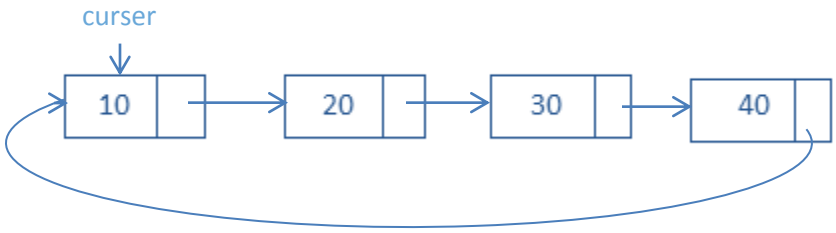
C-3.9

Give a fast algorithm for concatenating two doubly linked lists Land M, with header and trailer sentinel nodes, into a single list L'.

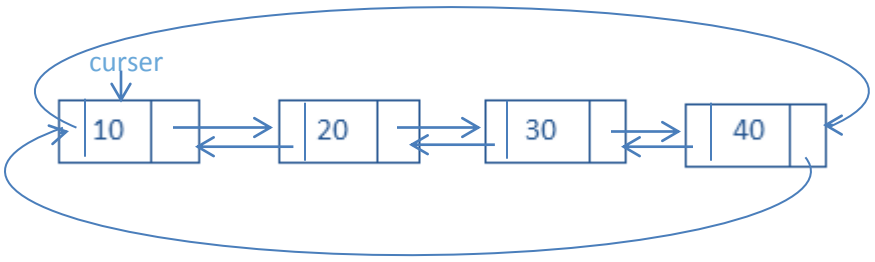
```
Algorithm Concatenate(L, M):
DNode V = (L.getTrailer()).getPrev()
DNode x = (M.getHeader()).getNext()
(M.getHeader()).setNext(null)
(L.getTrailer()).setPrev(null)
v.setNext(x)
x.setPrev(v)
L' = L
L'.setTrailer(M.getTrailer())
return L'
```

➢ Circularly linked list

- There is no head or tail but special node called curser.
- **Circularly singly linked list:** Pointer in the last node points back to the first node



- **Circularly doubly linked list:** Forward pointer of the last node points to the first node and backward pointer of the first node points to the last node



R-3.16

Write a short Java method to count the number of nodes in a circularly linked list.

```
int Count(){
Node n = curser.getNext();
int counter = 1;
while(n != curser){
     n = n.getNext();
     counter ++;
}
return counter;
```

➢ recursion

- Method called itself.
- Used to achieve repetition.
- **Base case:** case to get out of recursion

Linear recursion:

Perform only one recursive call.

Tail recursion:

Tail recursion occurs when a linearly recursive method makes its recursive call as its last step. Such methods can be easily converted to non-recursive methods (loop).

Binary recursion:

Binary recursion occurs whenever there are two recursive calls for each non-base case.

R-3.13

Draw the recursion trace for the execution of method ReverseArray(A, 0,4) (Code Fragment 3.32) on array A = {4, 3,6,2, 5}.

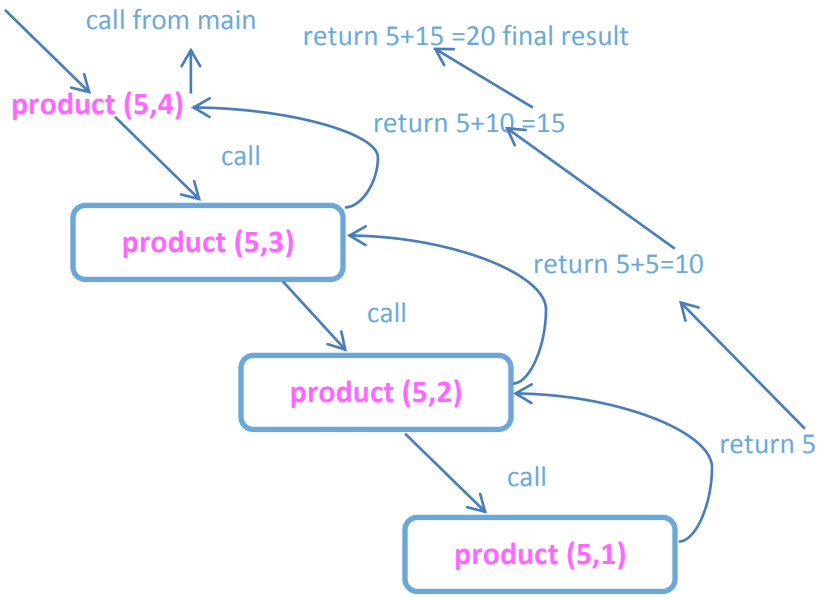
```
Algorithm ReverseArray(A, i, j):
 Input: An array A and nonnegative integer indices i and j
 Output: The reversal of the elements in A starting at index i
and ending at j
if i < j then
     Swap A[i] and A[j]
     ReverseArray(A, i+1, j-1)
return
```

1) i=0, j=4, A= {4, 3, 6, 2, 5}
   i<j → 0<4 (yes)
   swap(A[0], A[4])
   A= {5, 3, 6, 2, 4}
2) i=1, j=3, A= {5, 3, 6, 2, 4}
   i<j → 1<3 (yes)
   swap(A[1], A[3])
   A= {5, 2, 6, 3, 4}
3) i=2, j=2, A= {5, 2, 6, 3, 4}
   i<j → 2<2 (no)
   return

C-3.6

Give a recursive algorithm to compute the product of two positive integers, m and n, using only addition and subtraction.

```
Algorithm product(m, n):
if n=1
     return m
else
     return m + product(m, n+1)
```



C-3.14

Describe a recursive algorithm that counts the number of nodes in a singly linked list.

```
Algorithm count(n):
if(n=null)
     return 0
else
     return 1+count(n.getNext())
```

R-3.12

Describe a recursive algorithm for finding the maximum element in an array A of n elements. What is your running time and space usage?

```
Algorithm Max (A, m, n)
if A[n-1]>m
     m ← A[n-1]
if n=1
     return m
else
     return Max(A, m, n-1)
```

C-3.7

Describe a fast recursive algorithm for reversing a singly linked list L, so that the ordering of the nodes becomes opposite of what it was before.

```
Algorithm reverse(current, previous):
Node temp
if(current=tail)
     current.setNext(previous)
     temp ← head
     L.setHead(tail)
     L.setTail(temp)
else
     reverse(current.getNext(), current)
     current.setNext(previous)
```

C-3.13

Describe a recursive method for converting a string of digits into the integer it represents. For example,"13531 " represents the integer 13,531.

```
int convert(String s){
if(s.length()==1)
     return s.charAt(0)-48;
else{
     int c = s.charAt(s.length()-1)-48;
     return c + 10*convert(s.substring(0,s.length()-1));
}
```

Trace:

1+10(1353)
1+10(3+10(135))
1+10(3+10(5+10(13)))
1+10(3+10(5+10(3+10(1))))
1+10(3+10(5+10(3+10*1)))

C-3.18

Write a short recursive Java method that will rearrange an array of int values so that all the even values appear before all the odd values.

```
void rearrange(int []a,int n){
    if (n==0)
        return;
    else if(a[n-1]%2==0){
        for(int i=0;i<n-1;i++){
            if(a[i]%2!=0){
                swap(a[i], a[n-1])
                rearrange(a,n-1);}}}
    else
        rearrange(a,n-1);
}
```

C-3.19

Write a short recursive Java method that takes a character string sand outputs its reverse. So for example, the reverse of "pots&pans II would be "snap&stop".

```
String reverseString(String s){
        if (s.length() < 1)
                    return s;
        else {
            char c = s.charAt(0);
            return reverseString(s.substring(1))+c;
            }}
```

C-3.20

Write a short recursive Java method that determines if a string s is a palindrome, that is, it is equal to its reverse. For example, “racecar” and “ gohangasalamiimalasagnahog” are palindromes.

```
boolean isPalindrome(String s){
        if (s.length() <= 1)
                    return true;
        if(s.charAt(0)== s.charAt(s.length()-1))
            return isPalindrome(s.substring(1, s.length()-1));
        return false;
}
```

C-3.21

Use recursion to write a Java method for determining if a string s has more vowels than consonants.

```
boolean moreVowels(String s, int c){
        if (s.length() == 0)
                    return (c>0);
        if(s.charAt(s.length()-1)=='a'
                ||s.charAt(s.length()-1)=='e'
                ||s.charAt(s.length()-1)=='i'
                ||s.charAt(s.length()-1)=='o'
                ||s.charAt(s.length()-1)=='u')
            c++;
        else
            c--;
        return moreVowels(s.substring(0, s.length()-1),c);
}
```