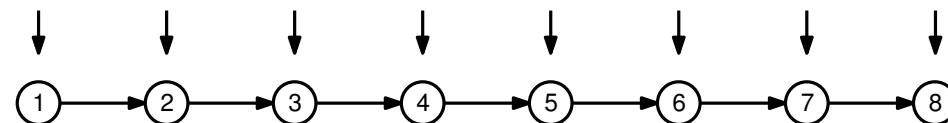
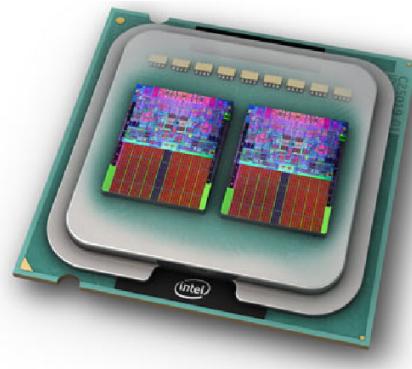




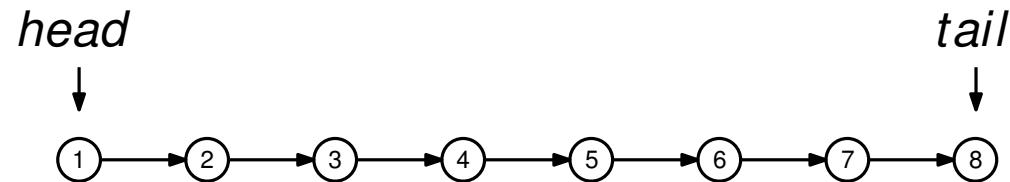
ICS 643: Advanced Parallel Algorithms

Prof. Nodari Sitchinava

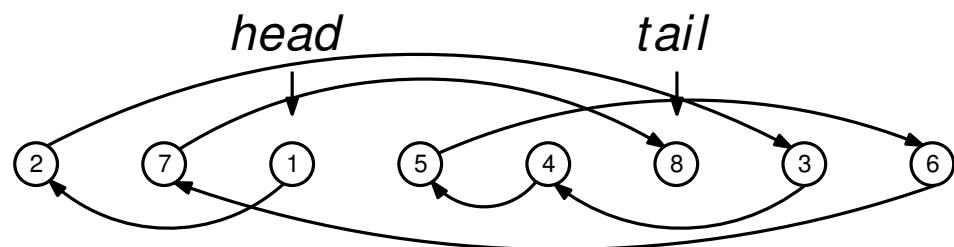
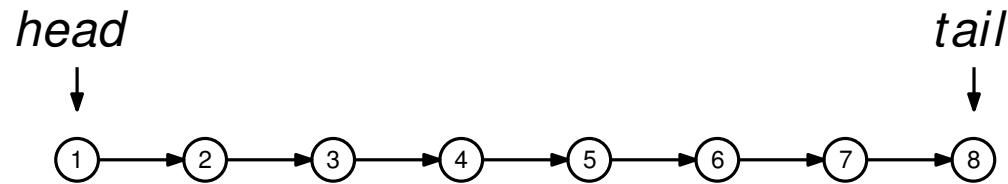


Lecture 14: List Ranking

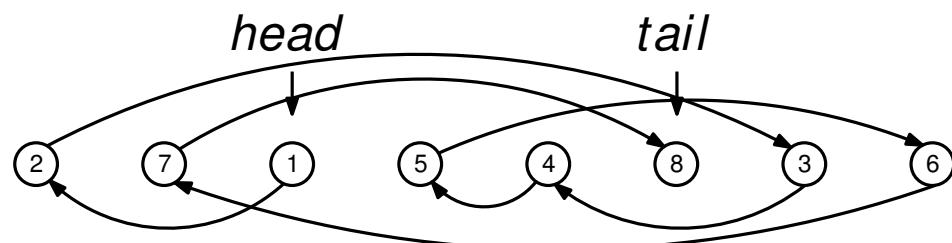
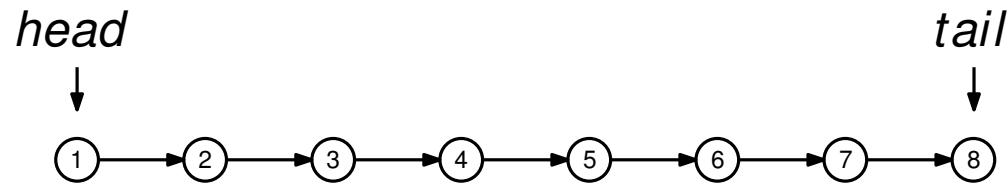
Link List Representation



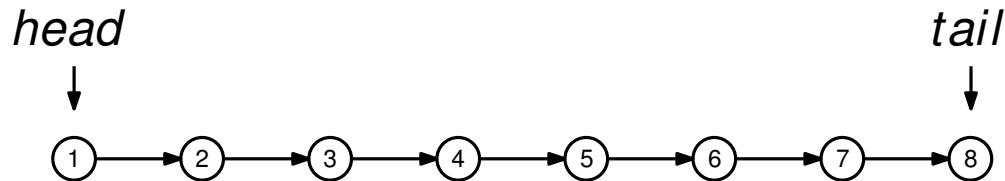
Link List Representation



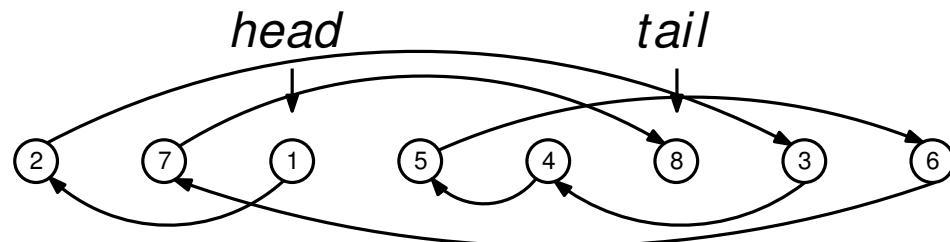
Link List Representation



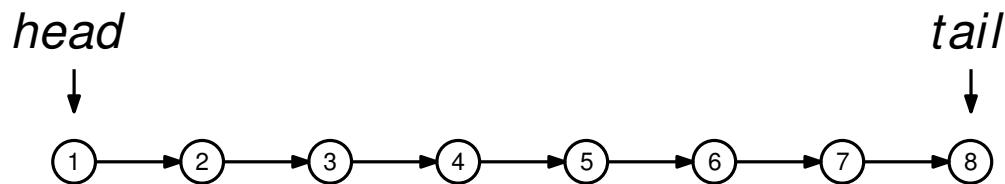
Link List Representation



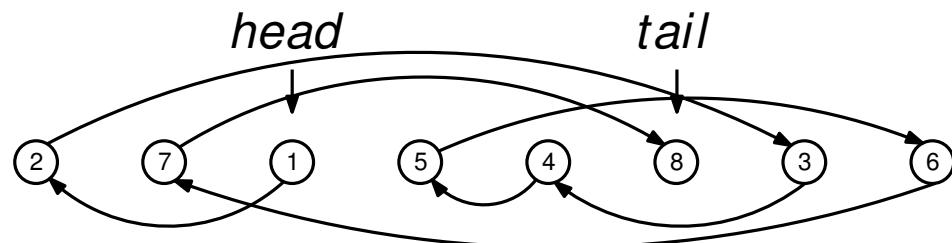
```
class NODE
    <TYPE> data
    NODE *next
    NODE *prev
```



Link List Representation



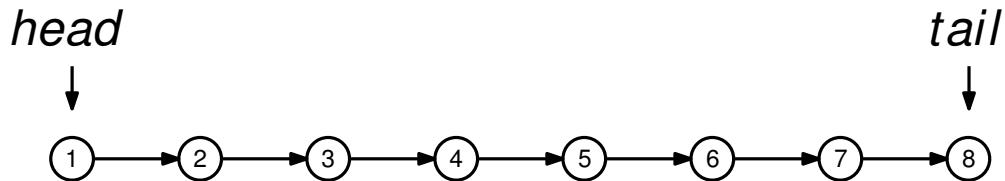
```
class NODE
    <TYPE> data
    NODE *next
    NODE *prev
```



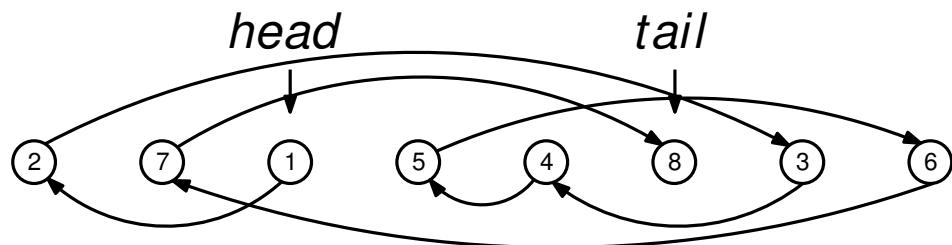
```
class LIST
    Node *head
    INT size
    Node *tail
```



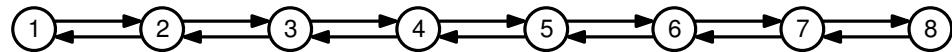
Link List Representation



```
class NODE
    <TYPE> data
    NODE *next
    NODE *prev
```



```
class LIST
    Node *head
    INT size
    Node *tail
```

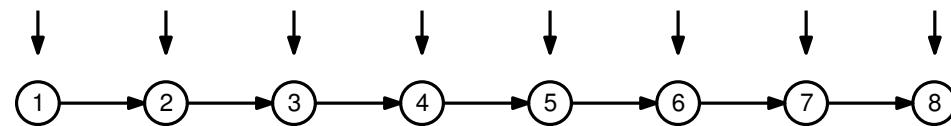


```
procedure PROCESSLIST(L)
    v = L.head
    while v ≠ NIL do
        PROCESS(v)
        v = v.next
```

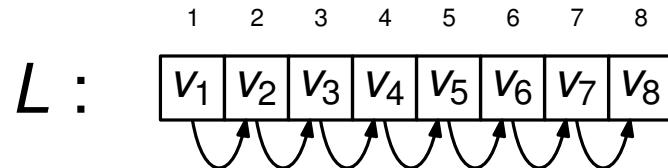
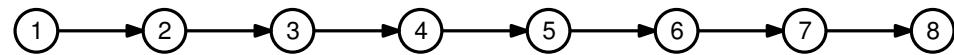
Parallel Processing of Lists

Would like to do:

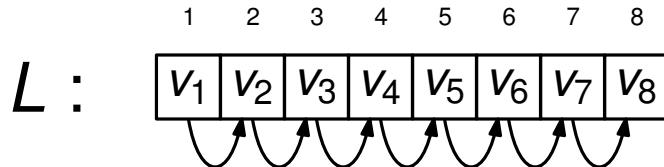
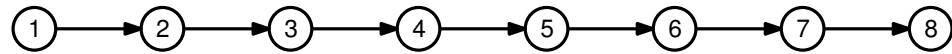
```
procedure PROCESSLIST(LIST L)
  for each v  $\in L$  in parallel do
    PROCESS(v)
```



List via Array Representation

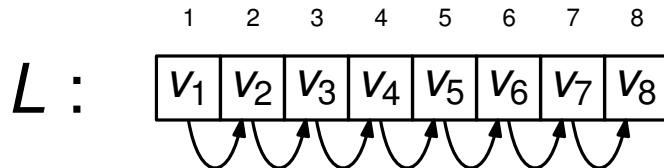
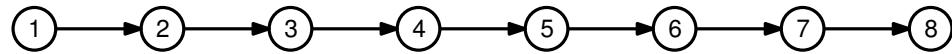


List via Array Representation

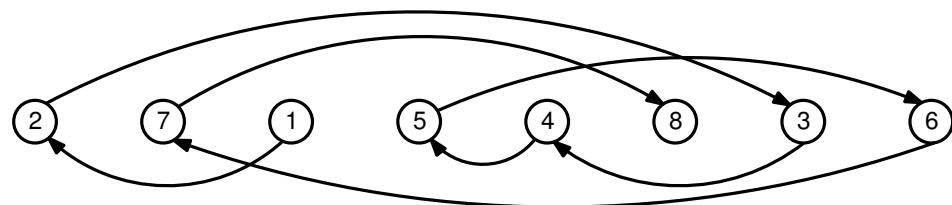


```
class LIST
    INT size
    NODE[] v[1..size]      ▷ array of NODES
    INT head               ▷ index of the head
```

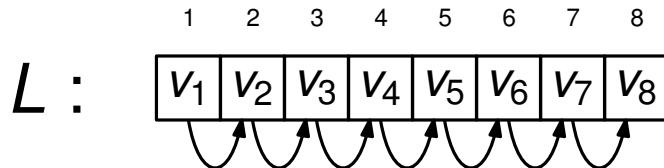
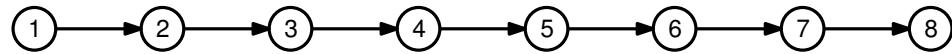
List via Array Representation



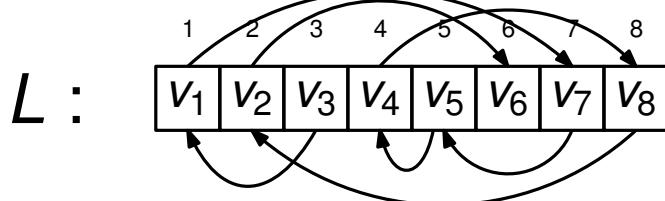
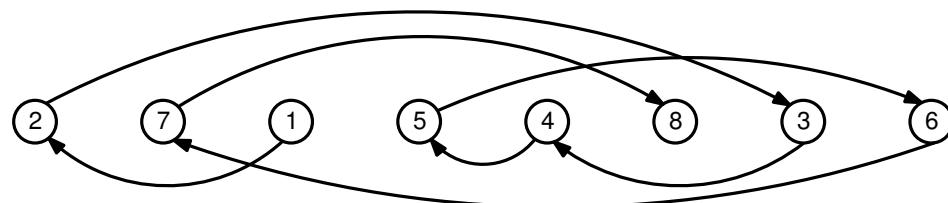
```
class LIST
    INT size
    NODE[] v[1..size]      ▷ array of NODES
    INT head               ▷ index of the head
```



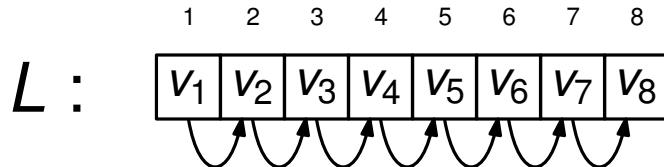
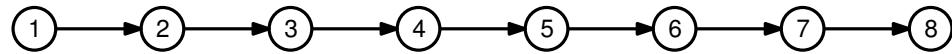
List via Array Representation



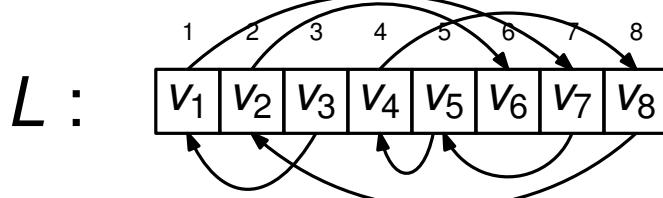
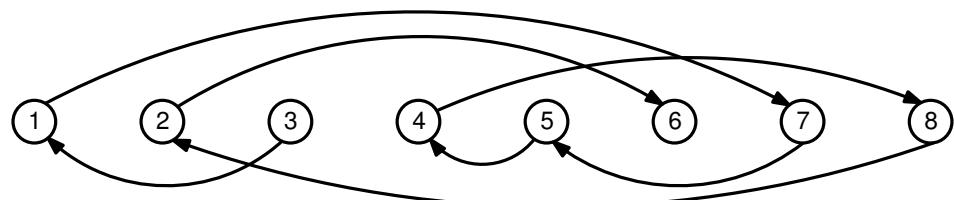
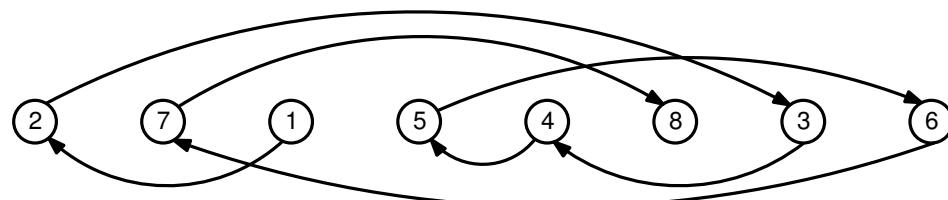
```
class LIST
    INT size
    NODE[] v[1..size]      ▷ array of NODES
    INT head                ▷ index of the head
```



List via Array Representation

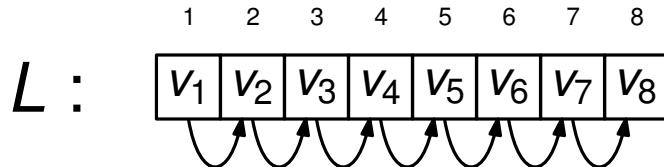
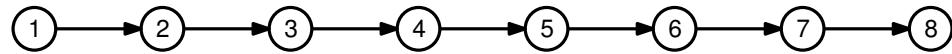


```
class LIST
    INT size
    NODE[] v[1..size]      ▷ array of NODES
    INT head                ▷ index of the head
```

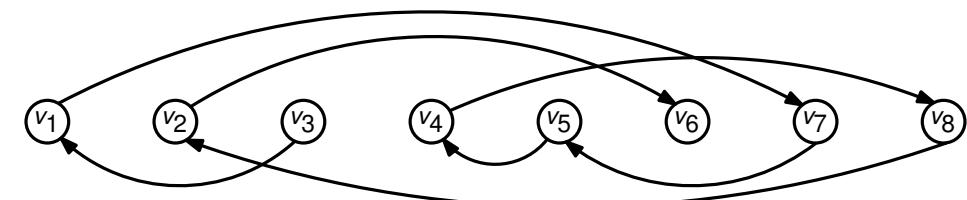
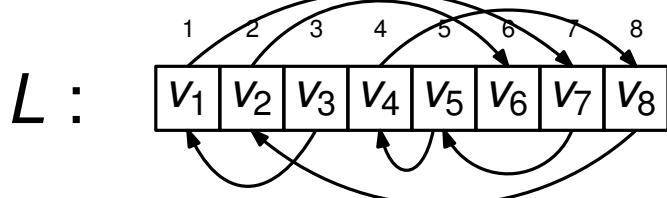
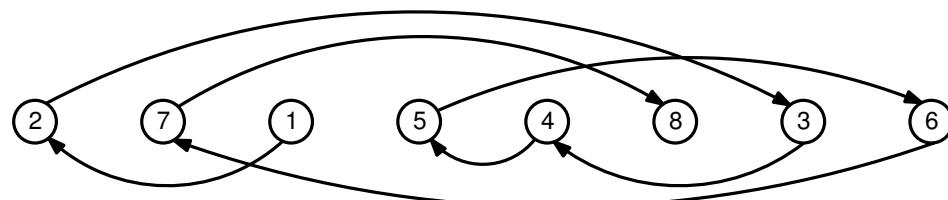


Node labels are indices in the array representation: $L[i] = v_i$

List via Array Representation

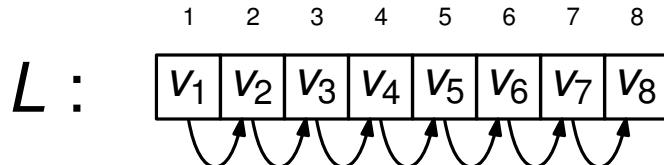
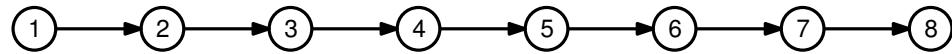


```
class LIST
    INT size
    NODE[] v[1..size]      ▷ array of NODES
    INT head                ▷ index of the head
```



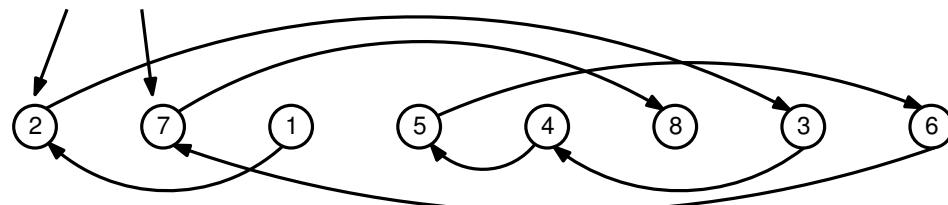
Node labels are indices in the array
representation: $L[i] = v_i$

List via Array Representation

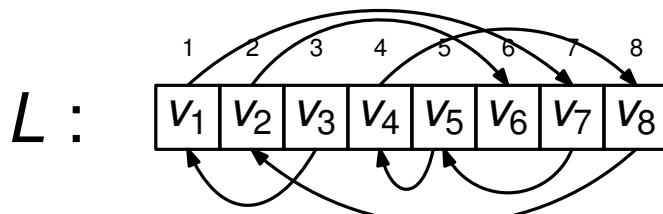
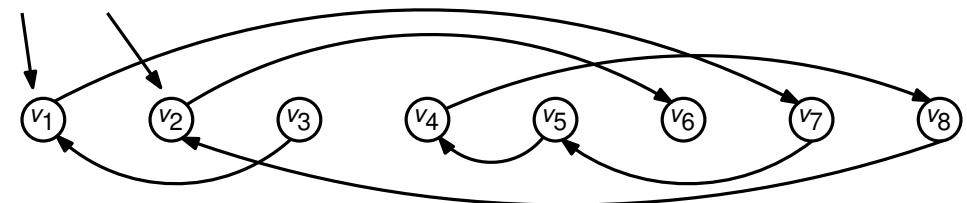


```
class LIST
    INT size
    NODE[] v[1..size]      ▷ array of NODES
    INT head                ▷ index of the head
```

node rank

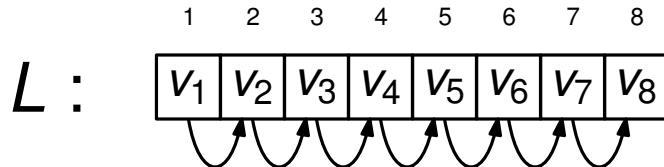
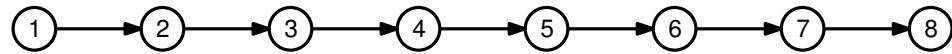


node label / node ID



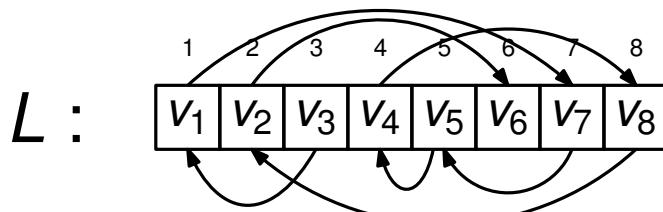
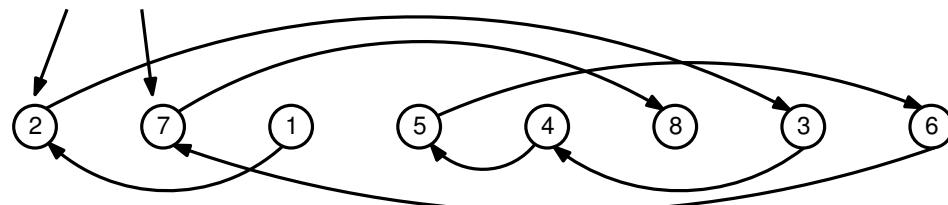
Node labels are indices in the array
representation: $L[i] = v_i$

List via Array Representation

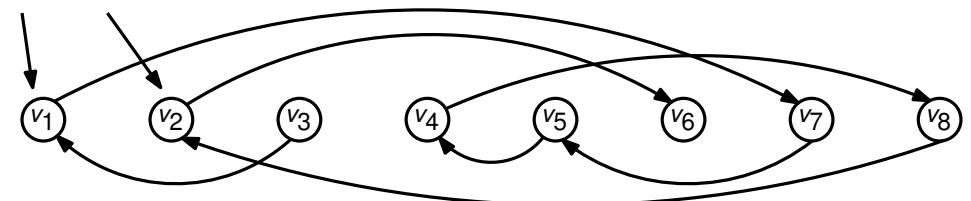


```
class LIST
    INT size
    NODE[] v[1..size]      ▷ array of NODES
    INT head                ▷ index of the head
```

node rank



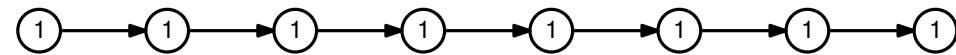
node label / node ID



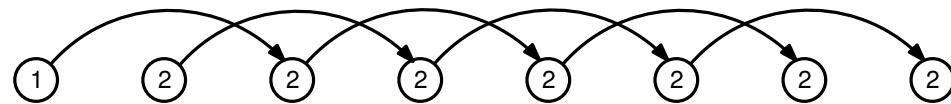
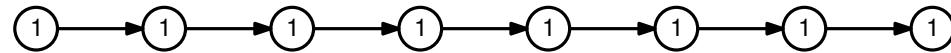
Node labels are indices in the array representation: $L[i] = v_i$

Array must be contiguous

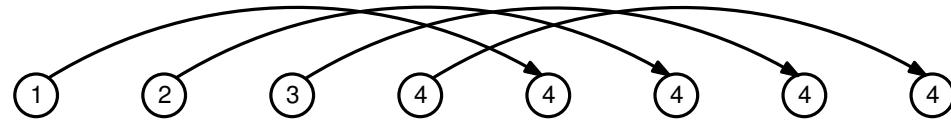
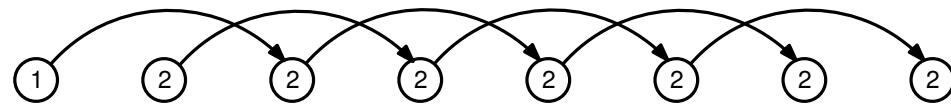
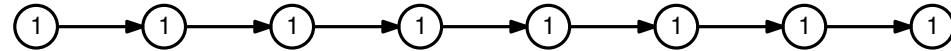
List Ranking via Pointer Hopping



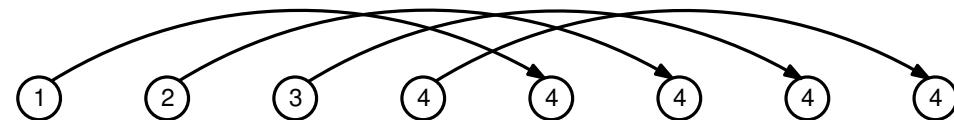
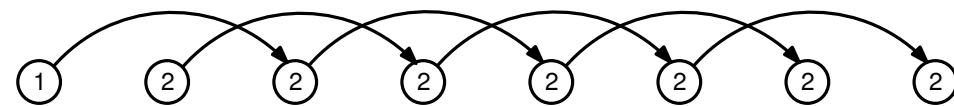
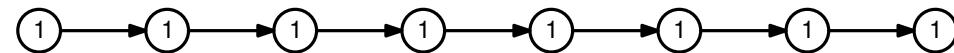
List Ranking via Pointer Hopping



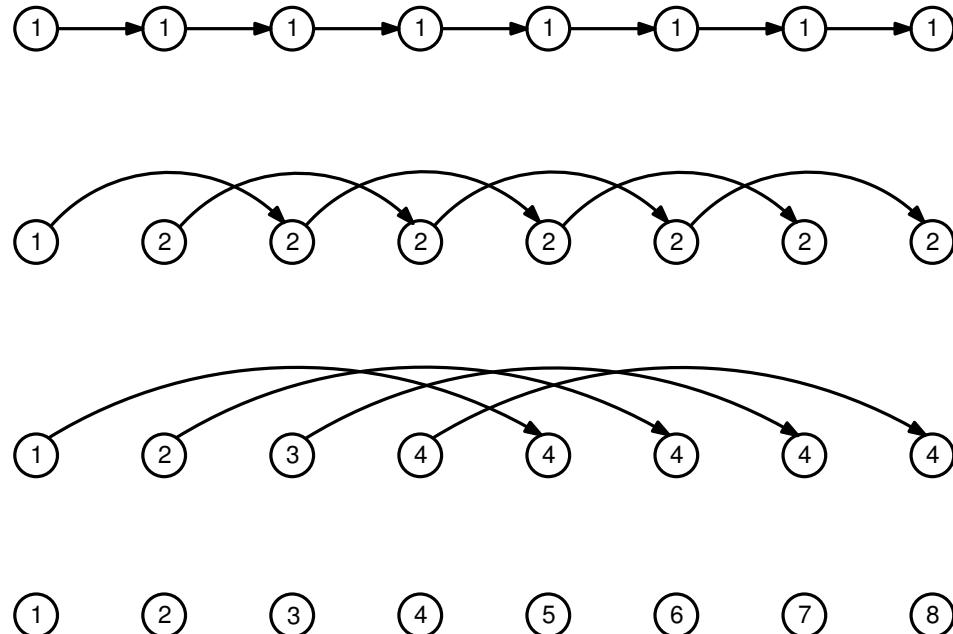
List Ranking via Pointer Hopping



List Ranking via Pointer Hopping



List Ranking via Pointer Hopping



```
procedure RANK( $L$ )
     $v = \text{new array of size } L.\text{size}$ 
    for  $i = 1$  to  $|v|$  in parallel do
         $v[i] = L.v[i]$             $\triangleright$  make a copy of  $L.v$ 
         $v[i].rank = 1$            $\triangleright$  Initialize  $v[i].rank$ 
    for  $i = 1$  to  $|v|$  in parallel do
        while  $v[i].next \neq \text{NIL}$  do
             $v[i].next.rank += v[i].rank$ 
             $v[i].next = v[i].next.next$ 
    return  $v$ 
```

Proof of Correctness

Claim. After the k -th iteration of the **while** loop, for each node v_i :

- $v_i.rank$ is set to the sum of the (initial) values of the preceding 2^k nodes (in the list order).
- $v_i.next$ is set to the node 2^k hops away from v_i (or `NIL` if no such node exists)

```
procedure RANK( $L$ )
     $v$  = new array of size  $L.size$ 
    for  $i = 1$  to  $|v|$  in parallel do
         $v[i] = L.v[i]$             $\triangleright$  make a copy of  $L.v$ 
         $v[i].rank = 1$            $\triangleright$  Initialize  $v[i].rank$ 
    for  $i = 1$  to  $|v|$  in parallel do
        while  $v[i].next \neq \text{NIL}$  do
             $v[i].next.rank += v[i].rank$ 
             $v[i].next = v[i].next.next$ 
    return  $v$ 
```

Proof of Correctness

Claim. After the k -th iteration of the **while** loop, for each node v_i :

- $v_i.rank$ is set to the sum of the (initial) values of the preceding 2^k nodes (in the list order).
- $v_i.next$ is set to the node 2^k hops away from v_i (or `NIL` if no such node exists)

Proof:

Termination: After the $\log n$ -th iteration, $2^{\log n} = n$, i.e., for all n nodes of the list:

- the rank is set correctly,
- the *next* pointer is set to `NIL`.

So the **while** loop terminates successfully.

```
procedure RANK( $L$ )
     $v$  = new array of size  $L.size$ 
    for  $i = 1$  to  $|v|$  in parallel do
         $v[i] = L.v[i]$            ▷ make a copy of  $L.v$ 
         $v[i].rank = 1$           ▷ Initialize  $v[i].rank$ 
    for  $i = 1$  to  $|v|$  in parallel do
        while  $v[i].next \neq \text{NIL}$  do
             $v[i].next.rank += v[i].rank$ 
             $v[i].next = v[i].next.next$ 
    return  $v$ 
```

Proof of Correctness

Claim. After the k -th iteration of the **while** loop, for each node v_i :

- $v_i.rank$ is set to the sum of the (initial) values of the preceding 2^k nodes (in the list order).
- $v_i.next$ is set to the node 2^k hops away from v_i (or `NIL` if no such node exists)

Proof:

Termination: After the $\log n$ -th iteration, $2^{\log n} = n$, i.e., for all n nodes of the list:

- the rank is set correctly,
- the *next* pointer is set to `NIL`.

So the **while** loop terminates successfully.

Initialization: Prior to the first iteration

(i.e., after $k = 0$ -th iteration) $2^k = 1$:

- *rank* of every node is set to $1 = 2^k$,
- *next* pointer of the tail node is `NIL` and the rest are $1 = 2^k$ hops away.

procedure RANK(L)

```
v = new array of size L.size
for i = 1 to |v| in parallel do
    v[i] = L.v[i]           ▷ make a copy of L.v
    v[i].rank = 1            ▷ Initialize v[i].rank
for i = 1 to |v| in parallel do
    while v[i].next ≠ NIL do
        v[i].next.rank += v[i].rank
        v[i].next = v[i].next.next
return v
```

Proof of Correctness

Claim. After the k -th iteration of the **while** loop, for each node v_i :

- $v_i.rank$ is set to the sum of the (initial) values of the preceding 2^k nodes (in the list order).
- $v_i.next$ is set to the node 2^k hops away from v_i (or `NIL` if no such node exists)

Proof:

Maintenance: If true **prior** to the k -th iteration, then true **after** the k -th iteration.

```
procedure RANK( $L$ )
     $v$  = new array of size  $L.size$ 
    for  $i = 1$  to  $|v|$  in parallel do
         $v[i] = L.v[i]$            ▷ make a copy of  $L.v$ 
         $v[i].rank = 1$           ▷ Initialize  $v[i].rank$ 
    for  $i = 1$  to  $|v|$  in parallel do
        while  $v[i].next \neq NIL$  do
             $v[i].next.rank += v[i].rank$ 
             $v[i].next = v[i].next.next$ 
    return  $v$ 
```

Proof of Correctness

Claim. After the k -th iteration of the **while** loop, for each node v_i :

- $v_i.rank$ is set to the sum of the (initial) values of the preceding 2^k nodes (in the list order).
- $v_i.next$ is set to the node 2^k hops away from v_i (or `NIL` if no such node exists)

Proof:

Maintenance: If true **prior** to the k -th iteration, then true **after** the k -th iteration.

After the $(k - 1)$ -th iteration:

```
procedure RANK( $L$ )
     $v$  = new array of size  $L.size$ 
    for  $i = 1$  to  $|v|$  in parallel do
         $v[i] = L.v[i]$             $\triangleright$  make a copy of  $L.v$ 
         $v[i].rank = 1$            $\triangleright$  Initialize  $v[i].rank$ 
    for  $i = 1$  to  $|v|$  in parallel do
        while  $v[i].next \neq \text{NIL}$  do
             $v[i].next.rank += v[i].rank$ 
             $v[i].next = v[i].next.next$ 
    return  $v$ 
```



Proof of Correctness

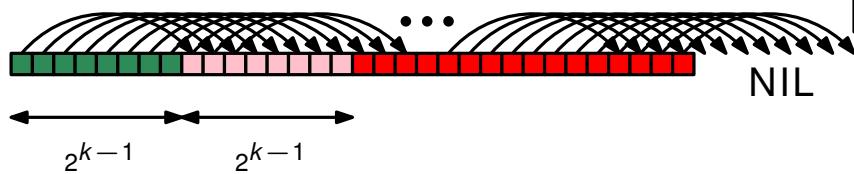
Claim. After the k -th iteration of the **while** loop, for each node v_i :

- $v_i.rank$ is set to the sum of the (initial) values of the preceding 2^k nodes (in the list order).
- $v_i.next$ is set to the node 2^k hops away from v_i (or `NIL` if no such node exists)

Proof:

Maintenance: If true **prior** to the k -th iteration, then true **after** the k -th iteration.

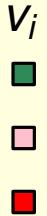
After the $(k - 1)$ -th iteration:



procedure RANK(L)

```
 $v = \text{new array of size } L.size$ 
for  $i = 1$  to  $|v|$  in parallel do
     $v[i] = L.v[i]$             $\triangleright$  make a copy of  $L.v$ 
     $v[i].rank = 1$            $\triangleright$  Initialize  $v[i].rank$ 
for  $i = 1$  to  $|v|$  in parallel do
    while  $v[i].next \neq \text{NIL}$  do
         $v[i].next.rank += v[i].rank$ 
         $v[i].next = v[i].next.next$ 
return  $v$ 
```

Prior to k -th iteration:



Proof of Correctness

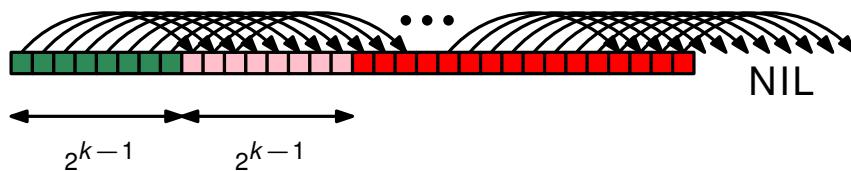
Claim. After the k -th iteration of the **while** loop, for each node v_i :

- $v_i.rank$ is set to the sum of the (initial) values of the preceding 2^k nodes (in the list order).
- $v_i.next$ is set to the node 2^k hops away from v_i (or NIL if no such node exists)

Proof:

Maintenance: If true **prior** to the k -th iteration, then true **after** the k -th iteration.

After the $(k - 1)$ -th iteration:



procedure RANK(L)

```

 $v = \text{new array of size } L.size$ 
for  $i = 1$  to  $|v|$  in parallel do
     $v[i] = L.v[i]$             $\triangleright$  make a copy of  $L.v$ 
     $v[i].rank = 1$            $\triangleright$  Initialize  $v[i].rank$ 
for  $i = 1$  to  $|v|$  in parallel do
    while  $v[i].next \neq \text{NIL}$  do
         $v[i].next.rank += v[i].rank$ 
         $v[i].next = v[i].next.next$ 
return  $v$ 

```

Prior to k -th iteration:

| | | |
|-------|------------------------|-------------------------|
| v_i | | |
| ■ | $v_i.rank = i$ (final) | $v_i.pred = \text{NIL}$ |
| □ | | |
| ■ | | |

Proof of Correctness

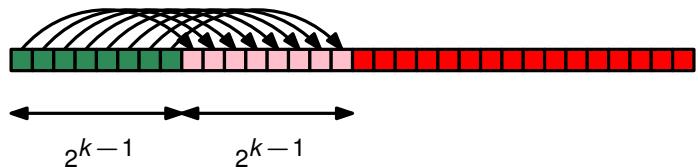
Claim. After the k -th iteration of the **while** loop, for each node v_i :

- $v_i.rank$ is set to the sum of the (initial) values of the preceding 2^k nodes (in the list order).
- $v_i.next$ is set to the node 2^k hops away from v_i (or `NIL` if no such node exists)

Proof:

Maintenance: If true **prior** to the k -th iteration, then true **after** the k -th iteration.

After the $(k - 1)$ -th iteration:



procedure RANK(L)

```
 $v = \text{new array of size } L.size$ 
for  $i = 1$  to  $|v|$  in parallel do
     $v[i] = L.v[i]$             $\triangleright$  make a copy of  $L.v$ 
     $v[i].rank = 1$             $\triangleright$  Initialize  $v[i].rank$ 
for  $i = 1$  to  $|v|$  in parallel do
    while  $v[i].next \neq \text{NIL}$  do
         $v[i].next.rank += v[i].rank$ 
         $v[i].next = v[i].next.next$ 
return  $v$ 
```

Prior to k -th iteration:

| v_i | $v_i.rank = i$ (final) | $v_i.rank = 2^{k-1}$ | $v_i.rank = \square$ | $v_i.rank = \blacksquare$ |
|-------|------------------------|----------------------|----------------------|---------------------------|
| | | | | |

Proof of Correctness

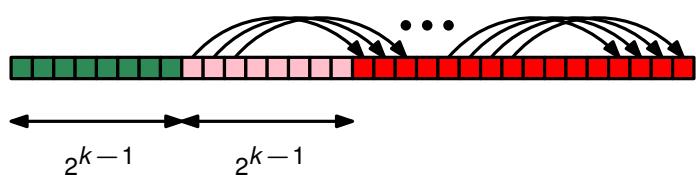
Claim. After the k -th iteration of the while loop, for each node v_i :

- $v_i.rank$ is set to the sum of the (initial) values of the preceding 2^k nodes (in the list order).
- $v_i.next$ is set to the node 2^k hops away from v_i (or NIL if no such node exists)

Proof:

Maintenance: If true **prior** to the k -th iteration, then true **after** the k -th iteration.

After the $(k - 1)$ -th iteration:



procedure RANK(L)

```

 $v = \text{new array of size } L.size$ 
for  $i = 1$  to  $|v|$  in parallel do
     $v[i] = L.v[i]$             $\triangleright$  make a copy of  $L.v$ 
     $v[i].rank = 1$             $\triangleright$  Initialize  $v[i].rank$ 
for  $i = 1$  to  $|v|$  in parallel do
    while  $v[i].next \neq \text{NIL}$  do
         $v[i].next.rank += v[i].rank$ 
         $v[i].next = v[i].next.next$ 
return  $v$ 

```

Prior to k -th iteration:
 v_i

- | | |
|--------------------------|--|
| ■ $v_i.rank = i$ (final) | $v_i.pred = \text{NIL}$ |
| □ $v_i.rank = 2^{k-1}$ | $v_i.pred = \blacksquare$ |
| ■ $v_i.rank = 2^{k-1}$ | $v_i.pred = \blacksquare \text{ or } \blacksquare$ |

Proof of Correctness

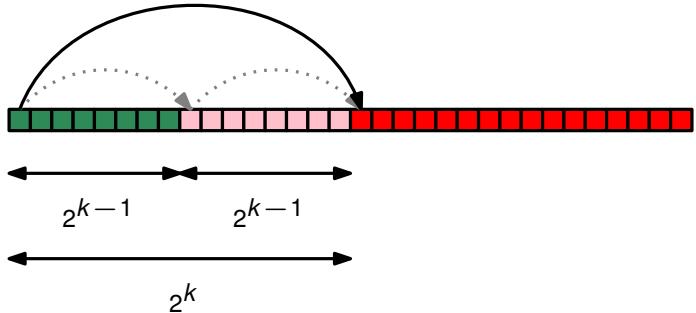
Claim. After the k -th iteration of the **while** loop, for each node v_i :

- $v_i.rank$ is set to the sum of the (initial) values of the preceding 2^k nodes (in the list order).
- $v_i.next$ is set to the node 2^k hops away from v_i (or `NIL` if no such node exists)

Proof:

Maintenance: If true **prior** to the k -th iteration, then true **after** the k -th iteration.

After the k -th iteration:



procedure RANK(L)

```
 $v = \text{new array of size } L.size$ 
for  $i = 1$  to  $|v|$  in parallel do
     $v[i] = L.v[i]$   $\triangleright$  make a copy of  $L.v$ 
     $v[i].rank = 1$   $\triangleright$  Initialize  $v[i].rank$ 
for  $i = 1$  to  $|v|$  in parallel do
    while  $v[i].next \neq \text{NIL}$  do
         $v[i].next.rank += v[i].rank$ 
         $v[i].next = v[i].next.next$ 
return  $v$ 
```

Prior to k -th iteration:

| v_i | |
|--------------------------|---|
| ■ $v_i.rank = i$ (final) | $v_i.pred = \text{NIL}$ |
| □ $v_i.rank = 2^{k-1}$ | $v_i.pred = \blacksquare$ |
| ■ $v_i.rank = 2^{k-1}$ | $v_i.pred = \blacksquare$ or \blacksquare |

After k -th iteration:



Proof of Correctness

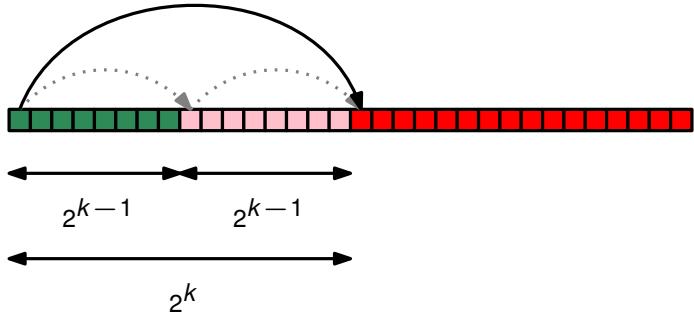
Claim. After the k -th iteration of the while loop, for each node v_i :

- $v_i.rank$ is set to the sum of the (initial) values of the preceding 2^k nodes (in the list order).
- $v_i.next$ is set to the node 2^k hops away from v_i (or NIL if no such node exists)

Proof:

Maintenance: If true ***prior*** to the k -th iteration, then true ***after*** the k -th iteration.

After the k -th iteration:



procedure RANK(L)

```

 $v = \text{new array of size } L.size$ 
for  $i = 1$  to  $|v|$  in parallel do
     $v[i] = L.v[i]$             $\triangleright$  make a copy of  $L.v$ 
     $v[i].rank = 1$             $\triangleright$  Initialize  $v[i].rank$ 
for  $i = 1$  to  $|v|$  in parallel do
    while  $v[i].next \neq \text{NIL}$  do
         $v[i].next.rank += v[i].rank$ 
         $v[i].next = v[i].next.next$ 
return  $v$ 
```

Prior to k -th iteration:

| v_i | |
|--------------------------|---|
| ■ $v_i.rank = i$ (final) | $v_i.pred = \text{NIL}$ |
| □ $v_i.rank = 2^{k-1}$ | $v_i.pred = \blacksquare$ |
| ■ $v_i.rank = 2^{k-1}$ | $v_i.pred = \blacksquare$ or \blacksquare |

After k -th iteration:

| | |
|--------------------------------------|---|
| ■ $v_i.rank = i$ (final) | $v_i.pred = \text{NIL}$ |
| □ $v_i.rank = i + 2^{k-1}$ (final) | $v_i.pred = \text{NIL}$ |
| ■ $v_i.rank = 2 \cdot 2^{k-1} = 2^k$ | $v_i.pred: 2 \cdot 2^{k-1}$ hops prior |

Time & Work Analysis

```
procedure RANK( $L$ )
     $v$  = new array of size  $L.size$ 
    for  $i = 1$  to  $|v|$  in parallel do
         $v[i] = L.v[i]$             $\triangleright$  make a copy of  $L.v$ 
         $v[i].rank = 1$            $\triangleright$  Initialize  $v[i].rank$ 
    for  $i = 1$  to  $|v|$  in parallel do
        while  $v[i].next \neq \text{NIL}$  do
             $v[i].next.rank += v[i].rank$ 
             $v[i].next = v[i].next.next$ 
    return  $v$ 
```

Time & Work Analysis

```
procedure RANK( $L$ )
     $v$  = new array of size  $L.size$ 
    for  $i = 1$  to  $|v|$  in parallel do
         $v[i] = L.v[i]$            ▷ make a copy of  $L.v$ 
         $v[i].rank = 1$           ▷ Initialize  $v[i].rank$ 
    for  $i = 1$  to  $|v|$  in parallel do
        while  $v[i].next \neq \text{NIL}$  do
             $v[i].next.rank += v[i].rank$ 
             $v[i].next = v[i].next.next$ 
    return  $v$ 
```

Termination: After the $\log n$ -th iteration,
 $2^{\log n} = n$, i.e., for all n nodes of the list:

- the rank is set correctly,
- the *next* pointer is set to NIL.

So the **while** loop terminates successfully.

Time & Work Analysis

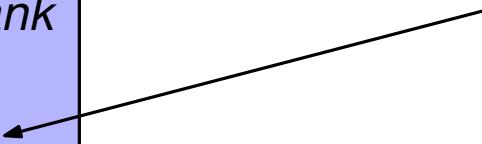
```
procedure RANK( $L$ )
     $v$  = new array of size  $L.size$ 
    for  $i = 1$  to  $|v|$  in parallel do
         $v[i] = L.v[i]$            ▷ make a copy of  $L.v$ 
         $v[i].rank = 1$           ▷ Initialize  $v[i].rank$ 
    for  $i = 1$  to  $|v|$  in parallel do
        while  $v[i].next \neq \text{NIL}$  do
             $v[i].next.rank += v[i].rank$ 
             $v[i].next = v[i].next.next$ 
    return  $v$ 
```

Termination: After the $\log n$ -th iteration,
 $2^{\log n} = n$, i.e., for all n nodes of the list:

- the rank is set correctly,
- the *next* pointer is set to NIL.

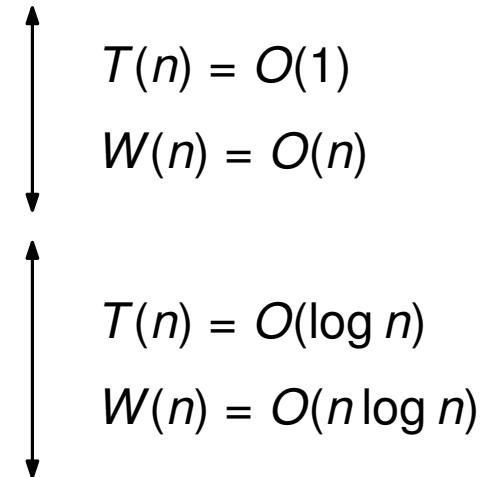
So the **while** loop terminates successfully.

Terminates after
 $O(\log n)$ iterations



Time & Work Analysis

```
procedure RANK( $L$ )
     $v$  = new array of size  $L.size$ 
    for  $i = 1$  to  $|v|$  in parallel do
         $v[i] = L.v[i]$             $\triangleright$  make a copy of  $L.v$ 
         $v[i].rank = 1$            $\triangleright$  Initialize  $v[i].rank$ 
    for  $i = 1$  to  $|v|$  in parallel do
        while  $v[i].next \neq \text{NIL}$  do
             $v[i].next.rank += v[i].rank$ 
             $v[i].next = v[i].next.next$ 
    return  $v$ 
```



Time & Work Analysis

```
procedure RANK( $L$ )
     $v$  = new array of size  $L.size$ 
    for  $i = 1$  to  $|v|$  in parallel do
         $v[i] = L.v[i]$             $\triangleright$  make a copy of  $L.v$ 
         $v[i].rank = 1$            $\triangleright$  Initialize  $v[i].rank$ 
    for  $i = 1$  to  $|v|$  in parallel do
        while  $v[i].next \neq \text{NIL}$  do
             $v[i].next.rank += v[i].rank$ 
             $v[i].next = v[i].next.next$ 
    return  $v$ 
```

$$T(n) = O(1)$$

$$W(n) = O(n)$$

$$T(n) = O(\log n)$$

$$W(n) = O(n \log n)$$

$$T(n) = O(\log n)$$

$$W(n) = O(n \log n)$$

Time & Work Analysis

```
procedure RANK( $L$ )
     $v$  = new array of size  $L.size$ 
    for  $i = 1$  to  $|v|$  in parallel do
         $v[i] = L.v[i]$            a copy of  $L.v$ 
         $v[i].rank = 1$           initialize  $v[i].rank$ 
    for  $i = 1$  to  $|v|$  in parallel do
        while  $v[i].next \neq \text{NIL}$  do
             $v[v[i].next.rank].rank += v[i].rank$ 
             $v[i].next = v[i].next.next$ 
    return  $v$ 
```

Not work-efficient

$$T(n) = O(1)$$

$$W(n) = O(n)$$

$$T(n) = O(\log n)$$

$$W(n) = O(n \log n)$$

$$T(n) = O(\log n)$$

$$W(n) = O(n \log n)$$