COMP108
Data Structures and Algorithms

Selection Sort Algorithm

Professor Prudence Wong

pwong@liverpool.ac.uk

2022-23

Outline

Selection sort algorithm

- using array
- using linked list

Learning outcome:

- Be able to describe and carry out asymptotic analysis of selection sort algorithm

Selection sort - Idea

- find minimum key from the input sequence
- delete it from input sequence
- append it to resulting sequence
- repeat until nothing left in input sequence

Selection sort - Example

<u>underlined</u>: current position **bold-red**: current smallest *italic*: sorted

34	10	64	51	32	21	

Selection sort - Example

underlined: current position **bold-red**: current smallest *italic*: sorted

34	10	64	51	32	21	To swap
34	**10**	64	51	32	21	

Selection sort - Example

underlined: current position **bold-red**: current smallest *italic*: sorted

34	10	64	51	32	21	To swap
34	**10**	64	51	32	21	34, 10
10	34	64	51	32	**21**	

Selection sort - Example

underlined: current position **bold-red**: current smallest *italic*: sorted

34	10	64	51	32	21	To swap
34	**10**	64	51	32	21	34, 10
10	34	64	51	32	**21**	34, 21
10	*21*	64	51	**32**	34	

Selection sort - Example

underlined: current position **bold-red**: current smallest *italic*: sorted

34	10	64	51	32	21	To swap
34	**10**	64	51	32	21	34, 10
10	34	64	51	32	**21**	34, 21
10	*21*	64	51	**32**	34	64, 32
10	*21*	*32*	51	64	**34**	

Selection sort - Example

underlined: current position **bold-red**: current smallest *italic*: sorted

34	10	64	51	32	21	To swap
34	**10**	64	51	32	21	34, 10
10	34	64	51	32	**21**	34, 21
10	*21*	64	51	**32**	34	64, 32
10	*21*	*32*	51	64	**34**	51, 34
10	*21*	*32*	*34*	64	**51**	

Selection sort - Example

underlined: current position **bold-red: current smallest** *italic*: sorted

34	10	64	51	32	21	To swap
34	**10**	64	51	32	21	34, 10
10	34	64	51	32	**21**	34, 21
10	*21*	64	51	**32**	34	64, 32
10	*21*	*32*	51	64	**34**	51, 34
10	*21*	*32*	*34*	64	**51**	64, 51
10	*21*	*32*	*34*	*51*	64	

Selection sort - Example

underlined: current position **bold-red**: current smallest *italic*: sorted

34	10	64	51	32	21	To swap
34	**10**	64	51	32	21	34, 10
10	34	64	51	32	**21**	34, 21
10	*21*	64	51	**32**	34	64, 32
10	*21*	*32*	51	64	**34**	51, 34
10	*21*	*32*	*34*	64	**51**	64, 51
10	*21*	*32*	*34*	*51*	64	
10	*21*	*32*	*34*	*51*	*64*	

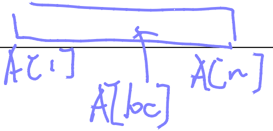
Selection sort algorithm

```
for i ← 1 to (n-1) do
begin



end
```

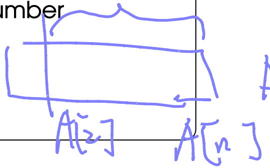
```
for i ← 1 to (n-1) do
begin    // find index 'loc' in the range $A[i]$ to $A[n]$




end
```

Selection sort algorithm

```
for i ← 1 to (n-1) do
begin
        // find the index 'loc' of the minimum number
        // in the range $A[i]$ to $A[n]$

end
```

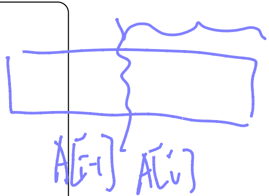


A[i] & A[loc]



A[2] & A[loc]

```
for i ← 1 to (n-1) do
begin    // find index 'loc' in the range $A[i]$ to $A[n]$



end
```



Swap A[i] & A[loc]

Selection sort algorithm

```
for i ← 1 to (n-1) do
begin
    // find the index 'loc' of the minimum number
    // in the range $A[i]$ to $A[n]$
    swap $A[i]$ and $A[loc]$
end
```

```
for i ← 1 to (n-1) do
begin    // find index 'loc' in the range $A[i]$ to $A[n]$




end
```

Selection sort algorithm

```
for i ← 1 to (n-1) do
begin
    // find the index 'loc' of the minimum number
    // in the range $A[i]$ to $A[n]$
    swap $A[i]$ and $A[loc]$
end
```

```
for i ← 1 to (n-1) do
begin    // find index 'loc' in the range $A[i]$ to $A[n]$
    loc ← i




    swap $A[i]$ and $A[loc]$
end
```

Selection sort algorithm

```
for i ← 1 to (n-1) do
begin
    // find the index 'loc' of the minimum number
    // in the range $A[i]$ to $A[n]$
    swap $A[i]$ and $A[loc]$
end
```

```
for i ← 1 to (n-1) do
begin    // find index 'loc' in the range $A[i]$ to $A[n]$
    loc ← i
    for j ← i+1 to n do



    swap $A[i]$ and $A[loc]$
end
```

Selection sort algorithm

```
for i ← 1 to (n-1) do
begin
    // find the index 'loc' of the minimum number
    // in the range $A[i]$ to $A[n]$
    swap $A[i]$ and $A[loc]$
end
```

```
for i ← 1 to (n-1) do
begin    // find index 'loc' in the range $A[i]$ to $A[n]$
    loc ← i
    for j ← i+1 to n do
        if $A[j] < A[loc]$ then
            loc ← j
    swap $A[i]$ and $A[loc]$
end
```

Selection sort algorithm - Using nested while loops

```
for i $\leftarrow$ 1 to (n-1) do
begin
    loc $\leftarrow$ i
    for j $\leftarrow$ i+1 to n do
        if $A[j] < A[loc]$ then
            loc $\leftarrow$ j
    swap $A[i]$ and $A[loc]$
end
```

```
i $\leftarrow$ 1
while i $\leq$ n do
begin
    loc $\leftarrow$ i
    j $\leftarrow$ i + 1
    while j $\leq$ n do
    begin
        if $A[j] < A[loc]$ then
            loc $\leftarrow$ j
        j $\leftarrow$ j + 1
    end
    swap $A[i]$ and $A[loc]$
    i $\leftarrow$ i + 1
end
```

Selection sort algorithm - Time complexity

The algorithm consists of a nested for-loop.

For each iteration of the outer i-loop, there is an inner j-loop.

```
for i ← 1 to (n-1) do
begin    // find index 'loc' in the range $A[i]$ to $A[n]$
    loc ← i
    for j ← i+1 to n do
        if $A[j] < A[loc]$ then
            loc ← j
    swap $A[i]$ and $A[loc]$
end
```

Selection sort algorithm - Time complexity

The algorithm consists of a nested for-loop.

For each iteration of the outer i-loop, there is an inner j-loop.

```
for i ← 1 to (n-1) do
begin    // find index 'loc' in the range $A[i]$ to $A[n]$
    loc ← i
    for j ← i+1 to n do
        if $A[j] < A[loc]$ then
            loc ← j
    swap $A[i]$ and $A[loc]$
end
```

i	# of < comparisons
1	n-1
2	n-2
⋮	⋮
n-1	1

Selection sort algorithm - Time complexity

The algorithm consists of a nested for-loop.

For each iteration of the outer i-loop, there is an inner j-loop.

```
for i ← 1 to (n-1) do
begin    // find index 'loc' in the range $A[i]$ to $A[n]$
    loc ← i
    for j ← i+1 to n do
        if $A[j] < A[loc]$ then
            loc ← j
    swap $A[i]$ and $A[loc]$
end
```

i	# of < comparisons
1	n-1
2	n-2
⋮	⋮
n-1	1

Total number of comparisons

$$= (n-1) + (n-2) + \cdots + 2 + 1$$

$$= \frac{n(n-1)}{2}$$

Selection sort algorithm - Time complexity $O(n^2)$-time

The algorithm consists of a nested for-loop.

For each iteration of the outer i-loop, there is an inner j-loop.

```
for i ← 1 to (n-1) do
begin     // find index 'loc' in the range $A[i]$ to $A[n]$
    loc ← i
    for j ← i+1 to n do
        if $A[j] < A[loc]$ then
            loc ← j
    swap $A[i]$ and $A[loc]$
end
```

i	# of < comparisons
1	n-1
2	n-2
$\vdots$	$\vdots$
n-1	1

Total number of comparisons

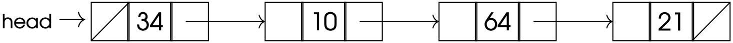
$$= (n-1) + (n-2) + \cdots + 2 + 1$$

$$= \frac{n(n-1)}{2}$$

Selection Sort with linked list. . .

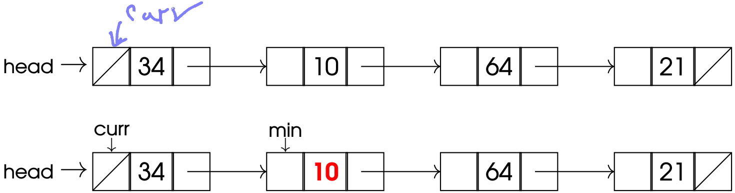
Selection sort with linked list - Example

underlined: being considered *italic*: sorted



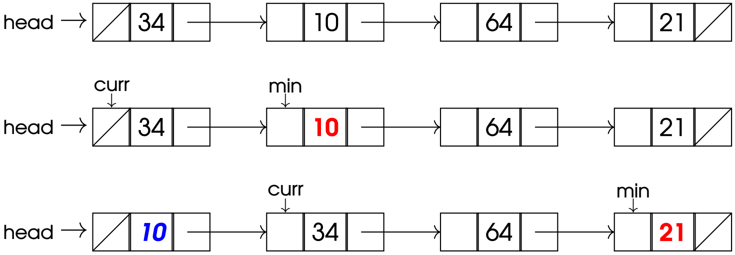
Selection sort with linked list - Example

underlined: being considered *italic*: sorted



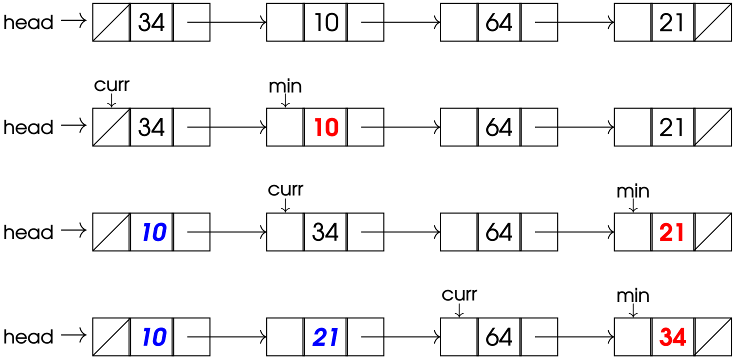
Selection sort with linked list - Example

underlined: being considered *italic*: sorted



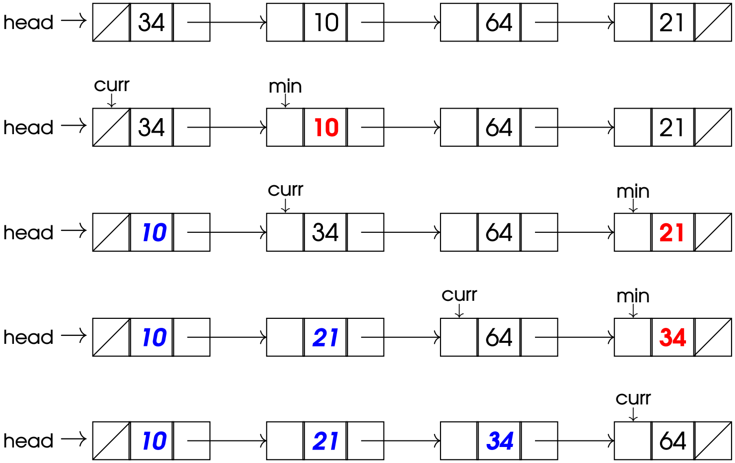
Selection sort with linked list - Example

underlined: being considered *italic*: sorted



Selection sort with linked list - Example

underlined: being considered *italic*: sorted



To find min, traverse from curr to tail to find min node

Selection sort with linked list

- First of all, if head is NIL, then the list is empty & nothing to sort.

```
        if head == NIL then Empty list and STOP!
```

Selection sort with linked list

- First of all, if head is NIL, then the list is empty & nothing to sort.

```
if head == NIL then Empty list and STOP!
```

- Consider the first round to find the minimum in the list
    - Recall sequential searching a list

```
while node $\neq$ NIL do
    node $\leftarrow$ node.next
```

Selection sort with linked list

- First of all, if head is NIL, then the list is empty & nothing to sort.

```
if head == NIL then Empty list and STOP!
```

- Consider the first round to find the minimum in the list
  - Recall sequential searching a list

```
while node $\neq$ NIL do
    node $\leftarrow$ node.next
```

  - Recall finding minimum algorithm, we need a pointer min

```
        **if node.data < min.data then min $\leftarrow$ node**
```

Selection sort with linked list

c.f. array, min is loc; node is j

- First of all, if head is NIL, then the list is empty & nothing to sort.

```
if head == NIL then Empty list and STOP!
```

- Consider the first round to find the minimum in the list
  - Recall sequential searching a list

```
while node ≠ NIL do
    node ← node.next
```

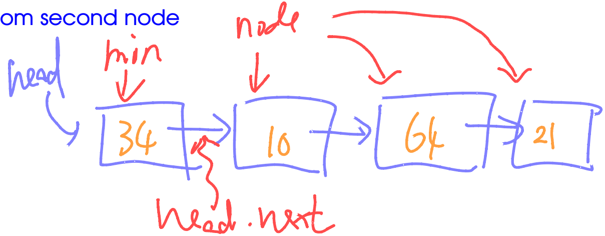
  - Recall finding minimum algorithm, we need a pointer min

```
    if node.data < min.data then min ← node
```

  - Combining we have

```
min ← head // min start from the beginning
node ← head.next // node start from second node
while node ≠ NIL do
begin
    if node.data < min.data then
        min ← node
    node ← node.next
end
```

```
min.data is 34
node.data is 10
IF-T => min.data is 10
node.data is 64
IF-F => min unchanged
node.data is 21
IF-F => min unchanged
```



Selection sort with linked list

- First of all, if head is NIL, then the list is empty & nothing to sort.

```
    if head == NIL then Empty list and STOP!
```

- Consider the first round to find the minimum in the list
  - Recall sequential searching a list

```
while node ≠ NIL do
    node ← node.next
```

  - Recall finding minimum algorithm, we need a pointer min

```
    if node.data < min.data then min ← node
```

  - Combining we have

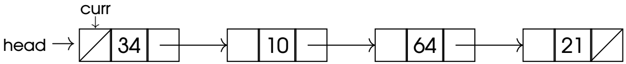
```
min ← head // min start from the beginning
node ← head.next // node start from second node
while node ≠ NIL do
begin
    if node.data < min.data then
        min ← node
    node ← node.next
end
```

- What about next round?

Selection sort with linked list - adding the outer loop

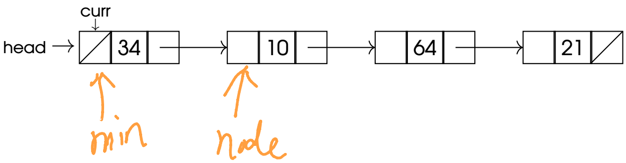
```
if head == NIL then
    Empty list and STOP
curr ← head
```

`c.f. array, curr is i`



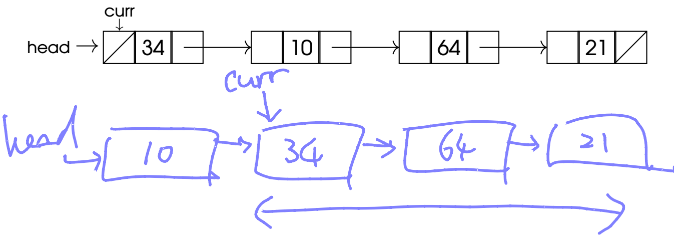
Selection sort with linked list - adding the outer loop

```
if head == NIL then
    Empty list and STOP
curr ← head

    min ← curr
    node ← curr.next
    while node ≠ NIL do
    begin
        if node.data < min.data then
            min ← node
        node ← node.next
    end
```



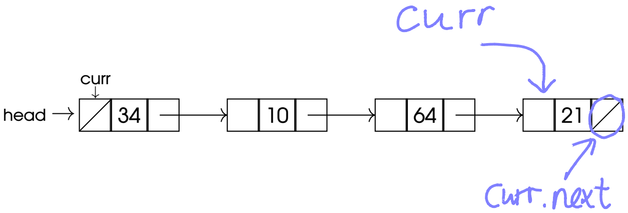
Selection sort with linked list - adding the outer loop

```
if head == NIL then
    Empty list and STOP
curr ← head


    min ← curr
    node ← curr.next
    while node ≠ NIL do
    begin
        if node.data < min.data then
            min ← node
        node ← node.next
    end
    swapnode(curr, min)        c.f. swap A[i] and A[loc]
    curr ← curr.next
                   i <- i+1
```



Selection sort with linked list - adding the outer loop

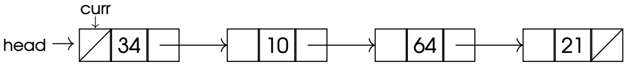
```
if head == NIL then
    Empty list and STOP
curr ← head
while curr.next ≠ NIL do
begin
    min ← curr
    node ← curr.next
    while node ≠ NIL do
    begin
        if node.data < min.data then
            min ← node
        node ← node.next
    end
    swapnode(curr, min)
    curr ← curr.next
end
```



Selection sort with linked list - adding the outer loop

```
if head == NIL then
    Empty list and STOP
curr ← head
while curr.next ≠ NIL do
begin
    min ← curr
    node ← curr.next
    while node ≠ NIL do
    begin
        if node.data < min.data then
            min ← node
        node ← node.next
    end
    swapnode(curr, min)
    curr ← curr.next
end
```

$O(n^2)$



Time complexity?

Time complexity of Selection Sort Algorithm

Which of the following statements is/are correct? Assume we have n numbers to be sorted by selection sort algorithm.

In the best case, selection sort takes 0 swap operation.

In the worst case, selection sort takes $O(n)$ swap operations.

Selection sort takes $O(n^2)$ comparisons.

The worst case time complexity of selection sort is $O(n^2)$.

Summary: Selection Sort Algorithm

Next: Insertion Sort Algorithm

For note taking