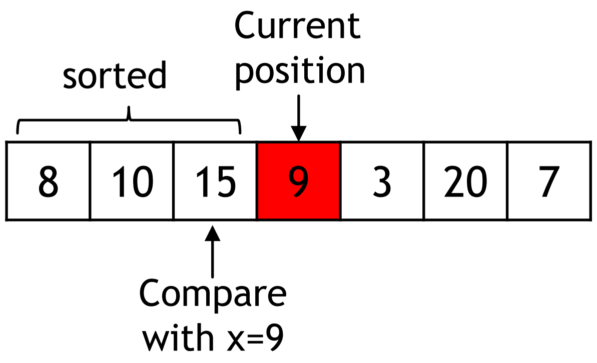
CSI 2110 Tutorial (Section A)

Yiheng Zhao
yzhao137@uottawa.ca
Office Hour: Fri 13:00-14:00
Place: STE 5000G

Review: Insertion Sort

Insert the current element into appropriate position.



sorted

Current position

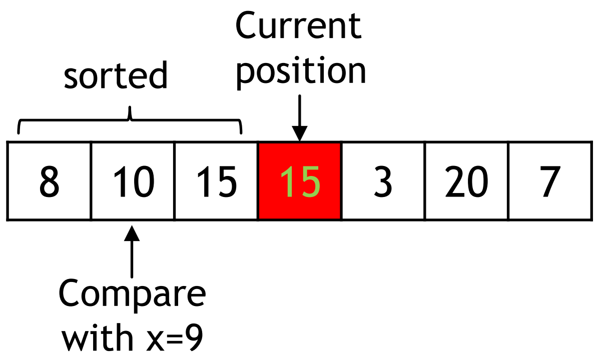
8	10	15	9	3	20	7

Compare with x=9

```
for i=1 to n-1
    x = A[i]
    j = i-1
    while x.key < A[j].key and j >= 0
        A[j+1] = A[j]
        j = j-1
    A[j+1] = x
```

Review: Insertion Sort

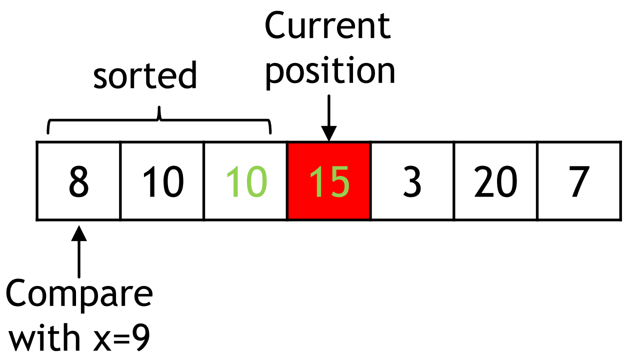
Insert the current element into appropriate position.



```
for i=1 to n-1
    x = A[i]
    j = i-1
    while x.key < A[j].key and j >= 0
        A[j+1] = A[j]
        j = j-1
    A[j+1] = x
```

Review: Insertion Sort

Insert the current element into appropriate position.



sorted

Current position

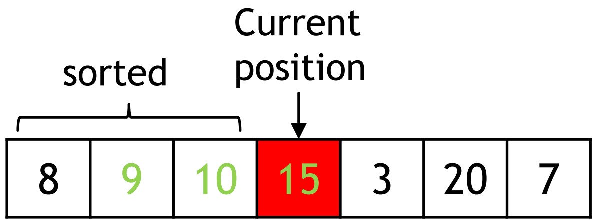
8	10	10	15	3	20	7

Compare with x=9

```
for i=1 to n-1
    x = A[i]
    j = i-1
    while x.key < A[j].key and j >= 0
        A[j+1] = A[j]
        j = j-1
    A[j+1] = x
```

Review: Insertion Sort

Insert the current element into appropriate position.



sorted

Current position

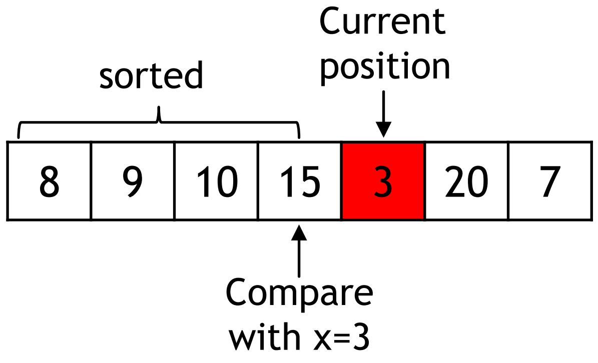
8	9	10	15	3	20	7

Find the position with x=9

```
for i=1 to n-1
    x = A[i]
    j = i-1
    while x.key < A[j].key and j >= 0
        A[j+1] = A[j]
        j = j-1
    A[j+1] = x
```

Review: Insertion Sort

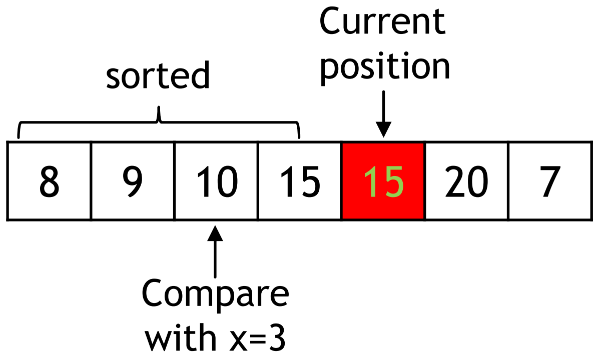
Insert the current element into appropriate position.



```
for i=1 to n-1
    x = A[i]
    j = i-1
    while x.key < A[j].key and j >= 0
        A[j+1] = A[j]
        j = j-1
    A[j+1] = x
```

Review: Insertion Sort

Insert the current element into appropriate position.



Current position

sorted

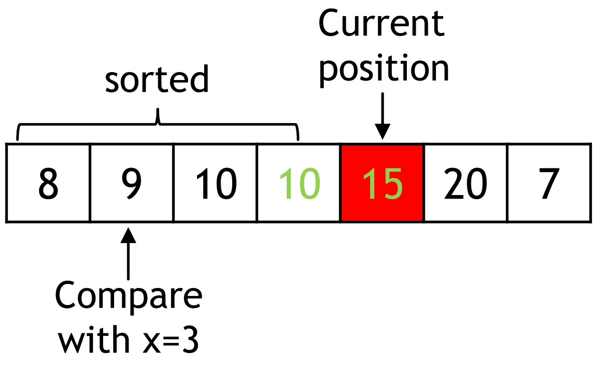
8	9	10	15	15	20	7

Compare with x=3

```
for i=1 to n-1
    x = A[i]
    j = i-1
    while x.key < A[j].key and j >= 0
        A[j+1] = A[j]
        j = j-1
    A[j+1] = x
```

Review: Insertion Sort

Insert the current element into appropriate position.



sorted

Current position

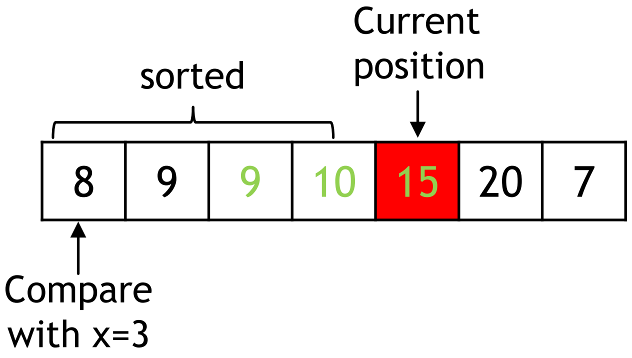
8	9	10	10	15	20	7

Compare with x=3

```
for i=1 to n-1
    x = A[i]
    j = i-1
    while x.key < A[j].key and j >= 0
        A[j+1] = A[j]
        j = j-1
    A[j+1] = x
```

Review: Insertion Sort

Insert the current element into appropriate position.



sorted

Current position

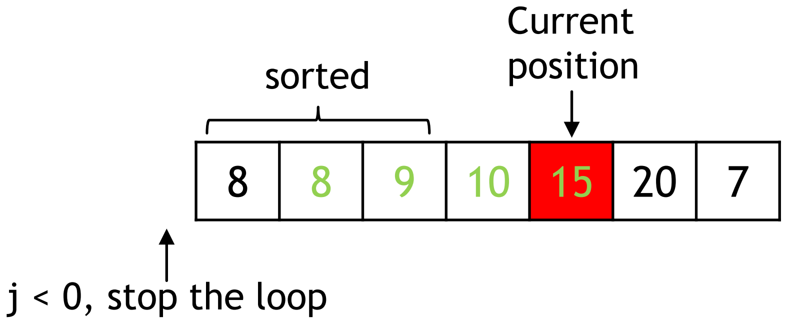
8	9	9	10	15	20	7

Compare with x=3

```
for i=1 to n-1
    x = A[i]
    j = i-1
    while x.key < A[j].key and j >= 0
        A[j+1] = A[j]
        j = j-1
    A[j+1] = x
```

Review: Insertion Sort

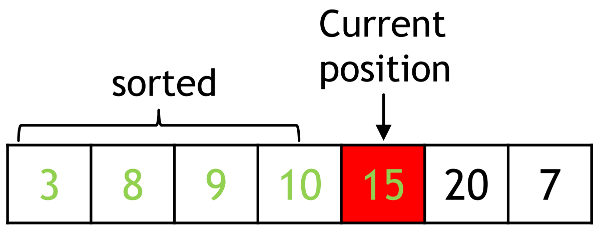
Insert the current element into appropriate position.



```
for i=1 to n-1
    x = A[i]
    j = i-1
    while x.key < A[j].key and j >= 0
        A[j+1] = A[j]
        j = j-1
    A[j+1] = x
```

Review: Insertion Sort

Insert the current element into appropriate position.



sorted

Current position

3	8	9	10	15	20	7

Insert x=3 to the front

```
for i=1 to n-1
    x = A[i]
    j = i-1
    while x.key < A[j].key and j >= 0
        A[j+1] = A[j]
        j = j-1
    A[j+1] = x
```

Complexity:

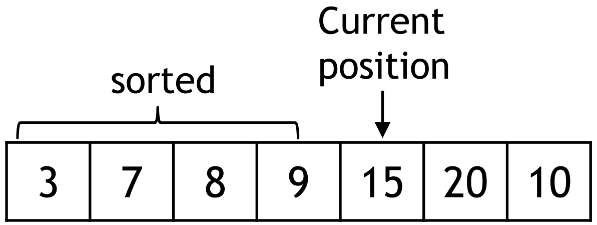
Min (in order): O(n)

Max (in reverse order): $O(n^2)$

1. Considering the following array with n=10 elements, use **insertion sort algorithm** to sort the array (shows the state after each insertion)

Index	0	1	2	3	4	5	6	7	8	9
Key	2	4	58	10	19	17	22	77	54	67
Key	2	4	58	10	19	17	22	77	54	67
Key	2	4	58	10	19	17	22	77	54	67
Key	2	4	58	10	19	17	22	77	54	67
Key	2	4	10	58	19	17	22	77	54	67
Key	2	4	10	19	58	17	22	77	54	67
Key	2	4	10	17	19	58	22	77	54	67
Key	2	4	10	17	19	22	58	77	54	67
Key	2	4	10	17	19	22	58	77	54	67
Key	2	4	10	17	19	22	54	58	77	67
Key	2	4	10	17	19	22	54	58	67	77

Review: Selection Sort



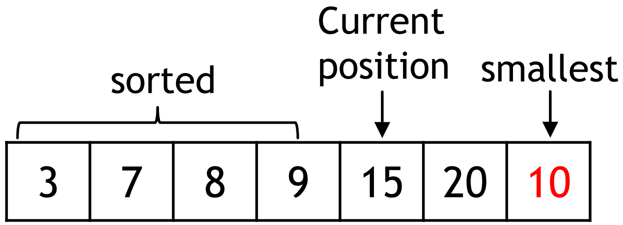
sorted

Current position

3	7	8	9	15	20	10

```
for i=0 to n-2
    k = i
    x = A[i]
    for j=i+1 to n-1
        if A[j].key < x.key
            k = j
            x = A[j]
    A[k] = A[i]
    A[i] = x
```

Review: Selection Sort



sorted

Current position

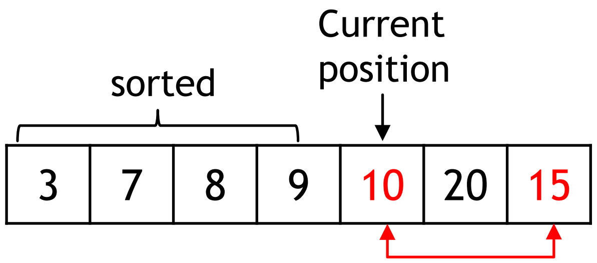
smallest

3	7	8	9	15	20	10

Find the smallest element in unsorted list

```
for i=0 to n-2
    k = i
    x = A[i]
    for j=i+1 to n-1
        if A[j].key < x.key
            k = j
            x = A[j]
    A[k] = A[i]
    A[i] = x
```

Review: Selection Sort



Current position

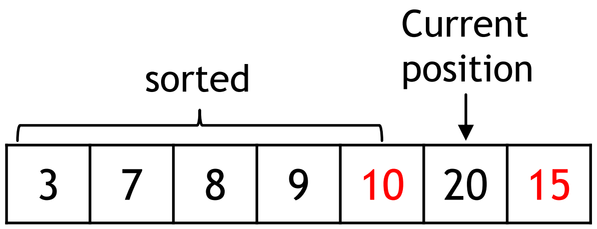
sorted

3	7	8	9	10	20	15

Swap the smallest element
with the one in current position

```
for i=0 to n-2
    k = i
    x = A[i]
    for j=i+1 to n-1
        if A[j].key < x.key
            k = j
            x = A[j]
    A[k] = A[i]
    A[i] = x
```

Review: Selection Sort



Move to the next position

```
for i=0 to n-2
    k = i
    x = A[i]
    for j=i+1 to n-1
        if A[j].key < x.key
            k = j
            x = A[j]
    A[k] = A[i]
    A[i] = x
```

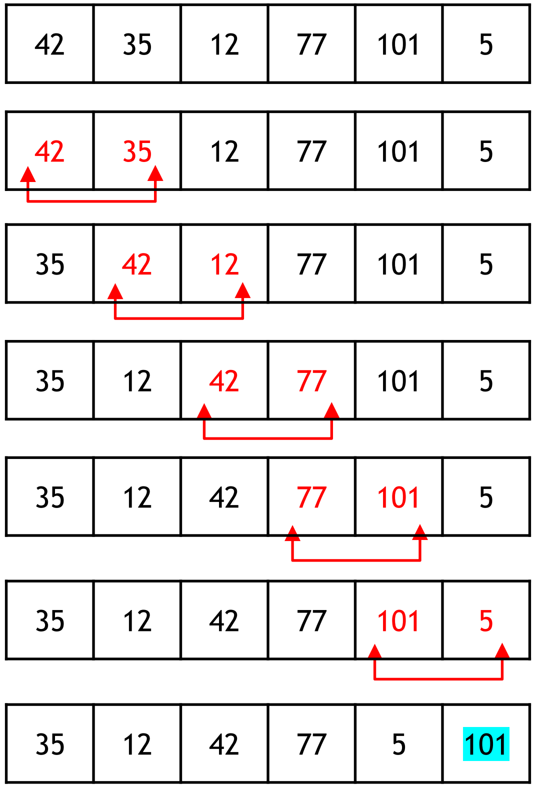
Complexity: $O(n^2)$

2. Considering the following array with n=10 elements, use **Selection sort algorithm** to sort the array (shows the state after each step)

Index	0	1	2	3	4	5	6	7	8	9
Key	2	4	58	10	19	17	22	77	54	67
Key	2	4	58	10	19	17	22	77	54	67
Key	2	4	58	10	19	17	22	77	54	67
Key	2	4	10	58	19	17	22	77	54	67
Key	2	4	10	17	19	58	22	77	54	67
Key	2	4	10	17	19	58	22	77	54	67
Key	2	4	10	17	19	22	58	77	54	67
Key	2	4	10	17	19	22	54	77	58	67
Key	2	4	10	17	19	22	58	58	77	67
Key	2	4	10	17	19	22	54	58	67	77
Key	2	4	10	17	19	22	54	58	67	77

Review: Bubble Sort

Loop 1:

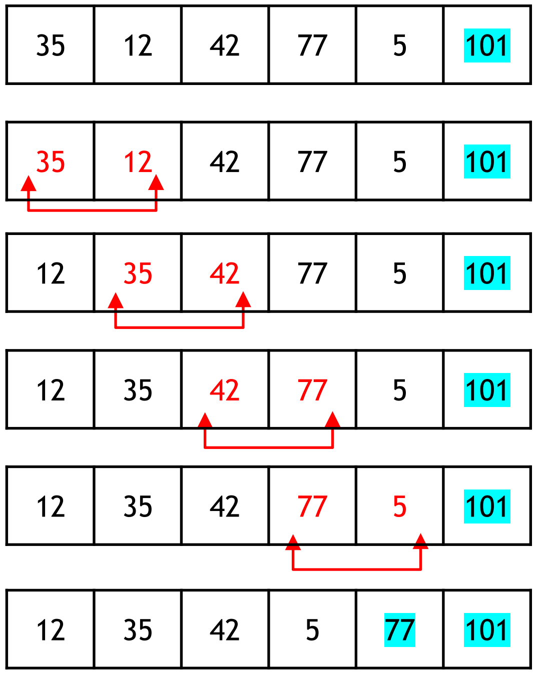


42	35	12	77	101	5
42	35	12	77	101	5
35	42	12	77	101	5
35	12	42	77	101	5
35	12	42	77	101	5
35	12	42	77	101	5
35	12	42	77	5	101

```
j = 0
swapped = true
While swapped
    swapped = false
    j = j+1
    for i=0 to n-j-1
        if A[i].key > A[i+1].key
            swap(A, i, i+1)
            swapped = true
```

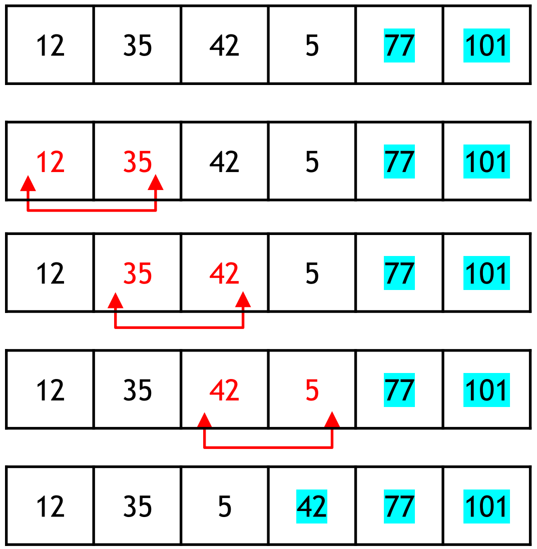
Review: Bubble Sort

Loop 2:



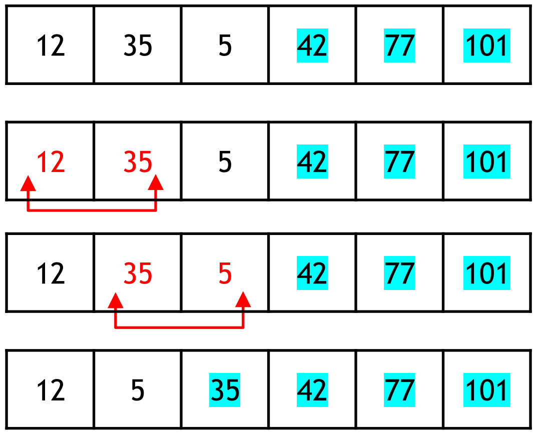
35	12	42	77	5	101
35	12	42	77	5	101
12	35	42	77	5	101
12	35	42	77	5	101
12	35	42	77	5	101
12	35	42	5	77	101

Loop 3:



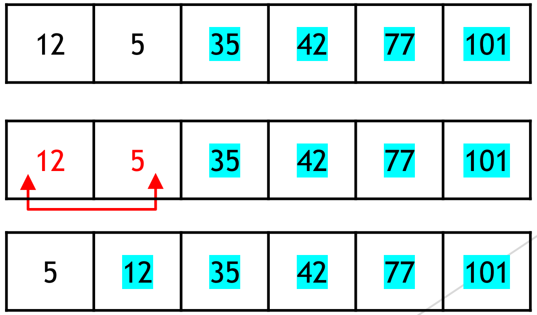
12	35	42	5	77	101
12	35	42	5	77	101
12	35	42	5	77	101
12	35	42	5	77	101
12	35	5	42	77	101

Loop 4:



12	35	5	42	77	101
12	35	5	42	77	101
12	35	5	42	77	101
12	5	35	42	77	101

Loop 5:



12	5	35	42	77	101
12	5	35	42	77	101
5	12	35	42	77	101

Complexity (compare):

Min (in order): O(n)

Max (in reverse order): $O(n^2)$

3. Considering the following array with n=10 elements, use **Bubble sort algorithm** to sort the array (shows the state after each loop)

	Index	0	1	2	3	4	5	6	7	8	9
initial	Key	2	4	58	10	19	17	22	77	54	67
Loop 1	Key	2	4	10	19	17	22	58	54	67	77
Loop 2	Key	2	4	10	17	19	22	54	58	67	77
Loop 3	Key	2	4	10	17	19	22	54	58	67	77

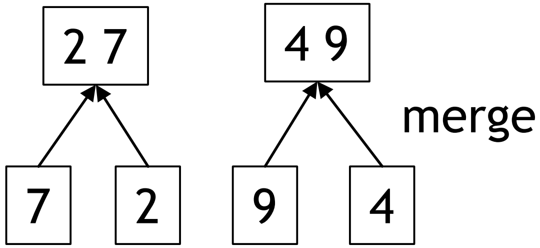
No swap, stop

Review: Merge Sort

```
mergeSort(A)
    if A.size() > 1
        (A1, A2) = partition(A, n/2)
        mergeSort(A1)
        mergeSort(A2)
        A = merge(A1, A2)
```

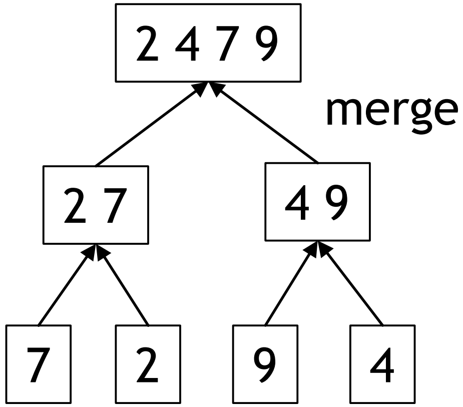
7	2	9	4

Review: Merge Sort



```
mergeSort(A)
    if A.size() > 1
        (A1, A2) = partition(A, n/2)
        mergeSort(A1)
        mergeSort(A2)
        A = merge(A1, A2)
```

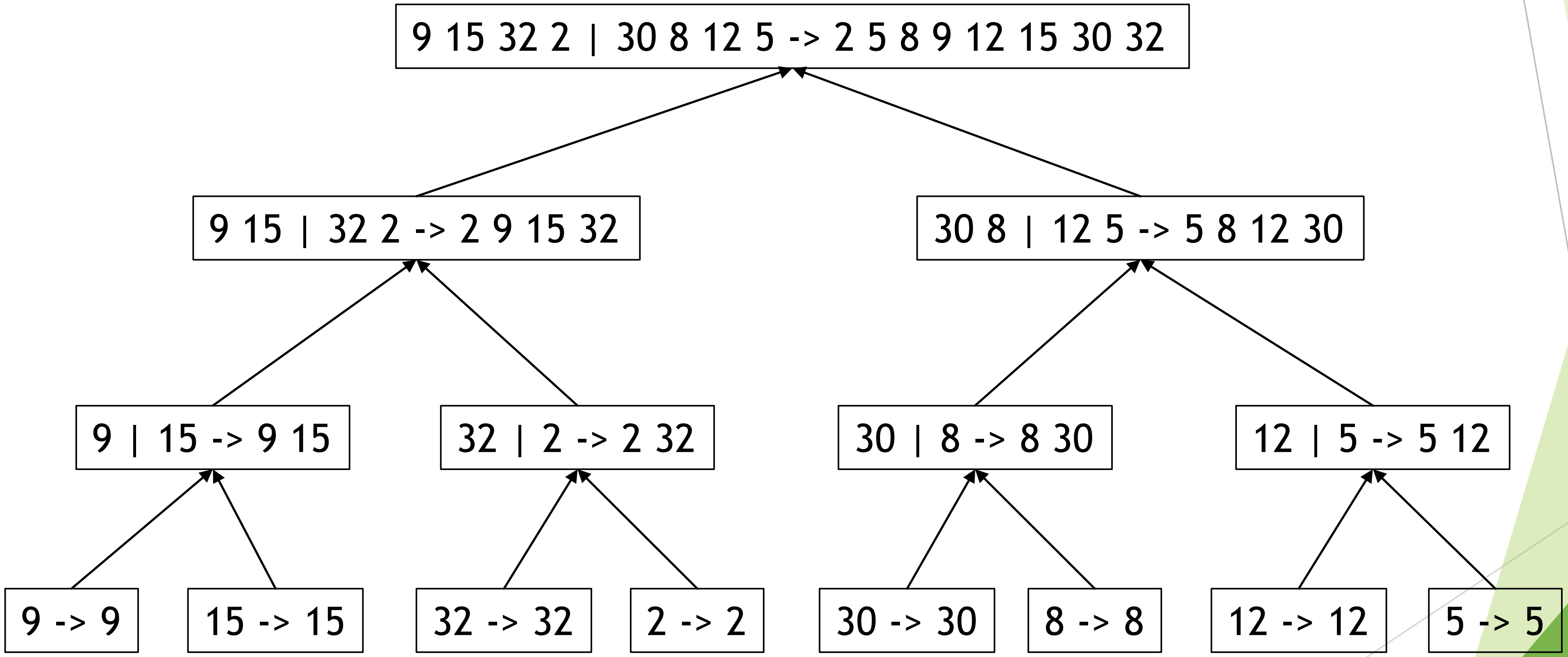
Review: Merge Sort



```
mergeSort(A)
    if A.size() > 1
        (A1, A2) = partition(A, n/2)
        mergeSort(A1)
        mergeSort(A2)
        A = merge(A1, A2)
```

Complexity : O(nlogn)

4. Draw the merge-sort tree with the following array:
(Only the nodes for the first partition are shown)



Review: Quick Sort

7 2 9 4 3 7 6 1

1. Random select a position key(i)
2. Divide the array into three parts (l, e, h):
    1) Elements smaller than key(i)
    2) Elements equals to key(i)
    3) Elements larger than key(i)

Repeat step 1 and 2 to array l and array h

```
inPlaceQuickSort(A, l, r)
    if l >= r
        return
    i = random(l, r)
    (h, k) = inPlacePartition(A, i, l, r)
     inPlaceQuickSort(A, l, h-1)
     inPlaceQuickSort(A, k+1, r)

inPlacePartition(A, p, s, e)
    l = s
    r = e-1
    swap(A, p, e), p = e  // pivot swap to the last pos
    while l <= r
        while A[l] <A[p] and r >=l
            l = l+1
        while A[r] >= A[p] and r >=l
            r = r-1
        if l<r: swap(A, l, r)
    swap(A, l, p)
    return r+1, l
```

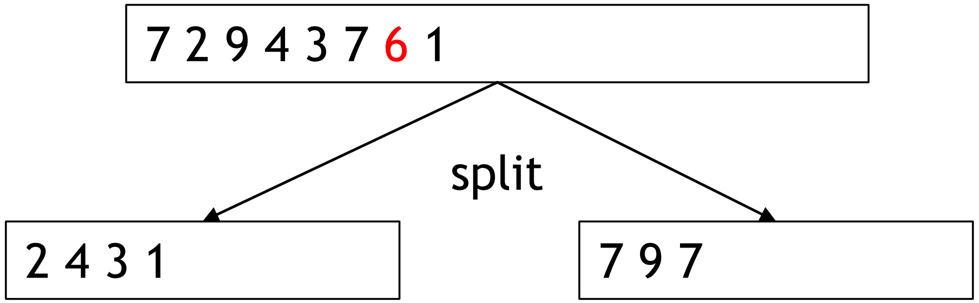
Review: Quick Sort

7 2 9 4 3 7 6 1

```
inPlaceQuickSort(A, l, r)
    if l >= r
        return
    i = random(l, r)
    (h, k) = inPlacePartition(A, i, l, r)
     inPlaceQuickSort(A, l, h-1)
     inPlaceQuickSort(A, k+1, r)


inPlacePartition(A, p, s, e)
    l = s
    r = e-1
    swap(A, p, e), p = e  // pivot swap to the last pos
    while l <= r
        while A[l] <A[p] and r >=l
            l = l+1
        while A[r] >= A[p] and r >=l
            r = r-1
    swap(A, l, p)
    return r+1, l
```

Review: Quick Sort



7 2 9 4 3 7 6 1

split

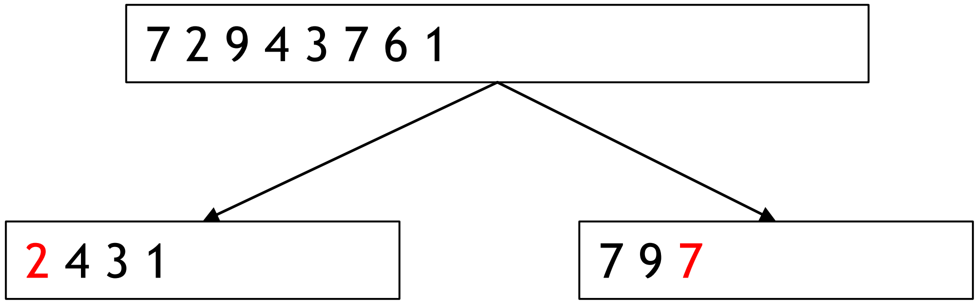
2 4 3 1

7 9 7

```
inPlaceQuickSort(A, l, r)
    if l >= r
        return
    i = random(l, r)
    (h, k) = inPlaceQuickPartition(A, i, l, r)
     inPlaceQuickSort(A, l, h-1)
     inPlaceQuickSort(A, k+1, r)


inPlacePartition(A, p, s, e)
    l = s
    r = e-1
    swap(A, p, e), p = e  // pivot swap to the last pos
    while l <= r
        while A[l] <A[p] and r >=l
            l = l+1
        while A[r] >= A[p] and r >=l
            r = r-1
    swap(A, l, p)
    return r+1, l
```

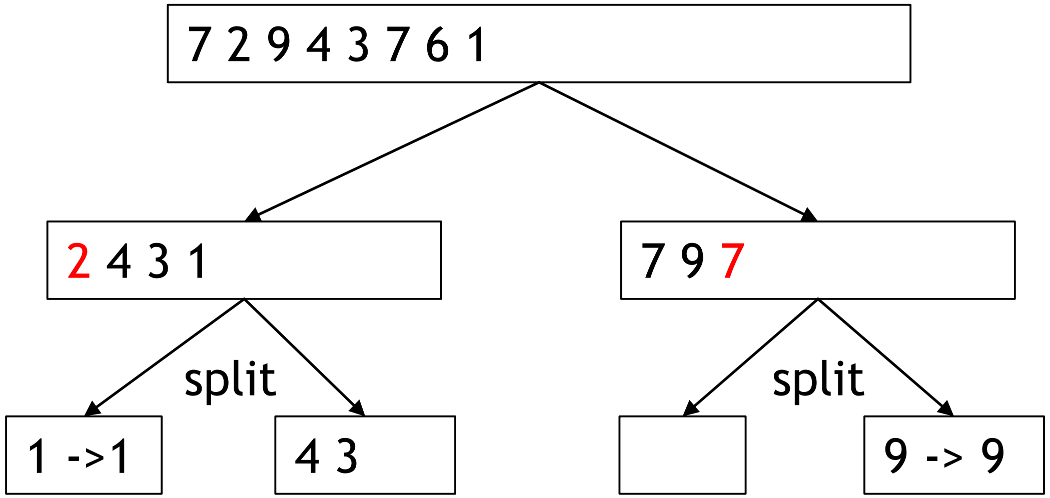
Review: Quick Sort



```
inPlaceQuickSort(A, l, r)
    if l >= r
        return
    i = random(l, r)
    (h, k) = inPlacePartition(A, i, l, r)
    inPlaceQuickSort(A, l, h-1)
    inPlaceQuickSort(A, k+1, r)


inPlacePartition(A, p, s, e)
    l = s
    r = e-1
    swap(A, p, e), p = e  // pivot swap to the last pos
    while l <= r
        while A[l] <A[p] and r >=l
            l = l+1
        while A[r] >= A[p] and r >=l
            r = r-1
    swap(A, l, p)
    return r+1, l
```

Review: Quick Sort



7 2 9 4 3 7 6 1

2 4 3 1

7 9 7

split

split

1 ->1

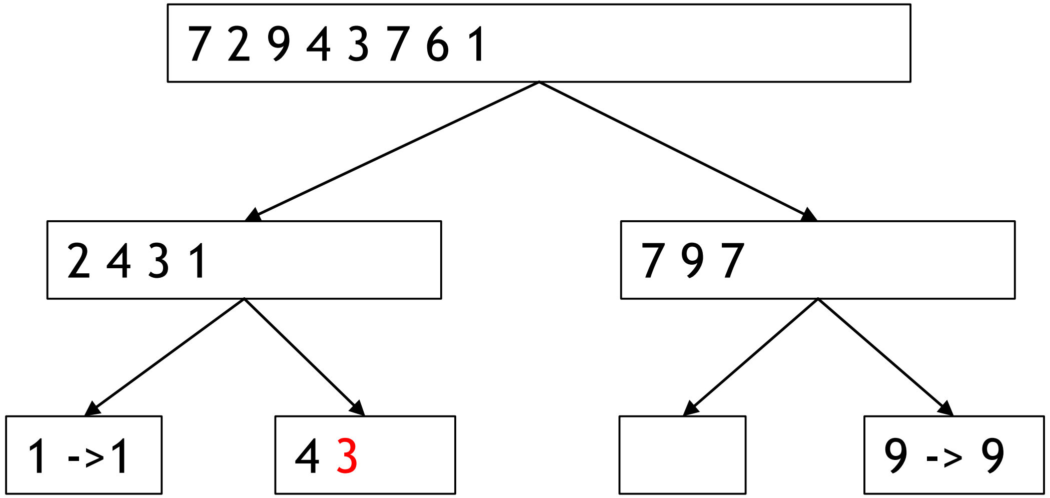
4 3

9 -> 9

```
inPlaceQuickSort(A, l, r)
    if l >= r
        return
    i = random(l, r)
    (h, k) = inPlacePartition(A, i, l, r)
     inPlaceQuickSort(A, l, h-1)
     inPlaceQuickSort(A, k+1, r)


inPlacePartition(A, p, s, e)
    l = s
    r = e-1
    swap(A, p, e), p = e  // pivot swap to the last pos
    while l <= r
        while A[l] <A[p] and r >=l
            l = l+1
        while A[r] >= A[p] and r >=l
            r = r-1
    swap(A, l, p)
    return r+1, l
```

Review: Quick Sort



7 2 9 4 3 7 6 1

2 4 3 1

7 9 7

1 ->1

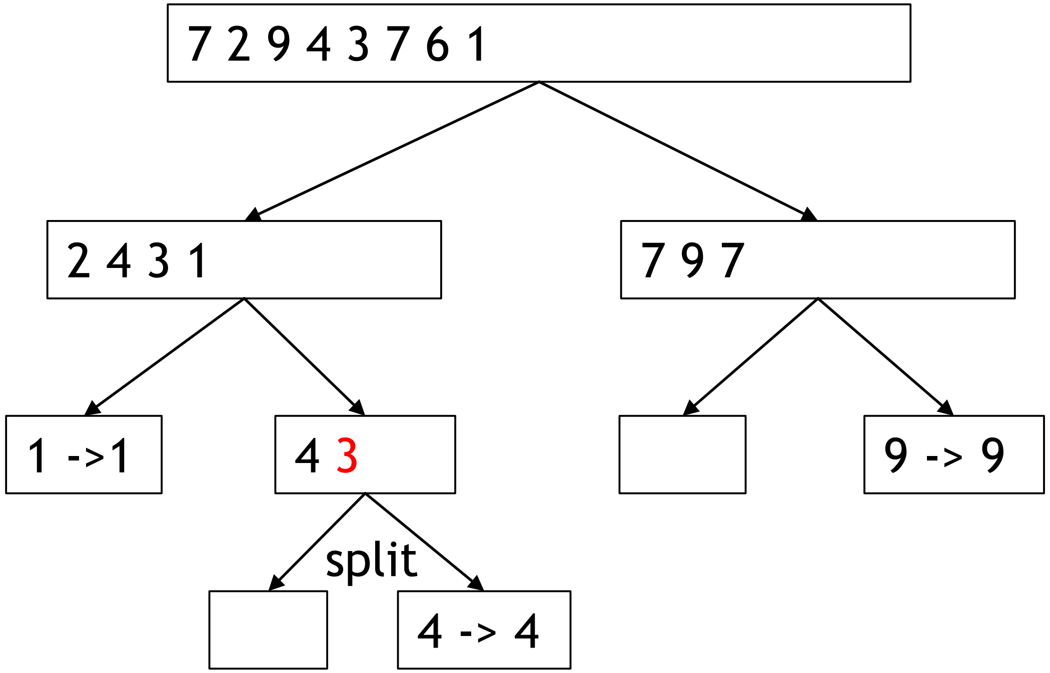
4 3

9 -> 9

```
inPlaceQuickSort(A, l, r)
    if l >= r
        return
    i = random(l, r)
    (h, k) = inPlacePartition(A, i, l, r)
     inPlaceQuickSort(A, l, h-1)
     inPlaceQuickSort(A, k+1, r)


inPlacePartition(A, p, s, e)
    l = s
    r = e-1
    swap(A, p, e), p = e  // pivot swap to the last pos
    while l <= r
        while A[l] <A[p] and r >=l
            l = l+1
        while A[r] >= A[p] and r >=l
            r = r-1
    swap(A, l, p)
    return r+1, l
```

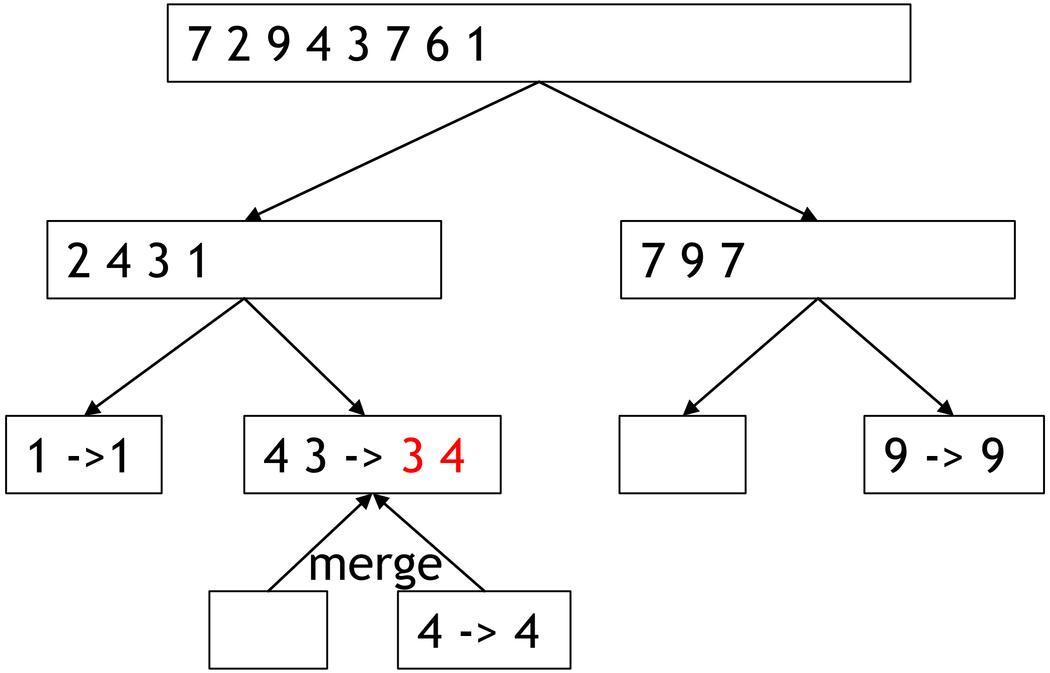
Review: Quick Sort



```
inPlaceQuickSort(A, l, r)
    if l >= r
        return
    i = random(l, r)
    (h, k) = inPlacePartition(A, i, l, r)
     inPlaceQuickSort(A, l, h-1)
     inPlaceQuickSort(A, k+1, r)


inPlacePartition(A, p, s, e)
    l = s
    r = e-1
    swap(A, p, e), p = e  // pivot swap to the last pos
    while l <= r
        while A[l] <A[p] and r >=l
            l = l+1
        while A[r] >= A[p] and r >=l
            r = r-1
    swap(A, l, p)
    return r+1, l
```

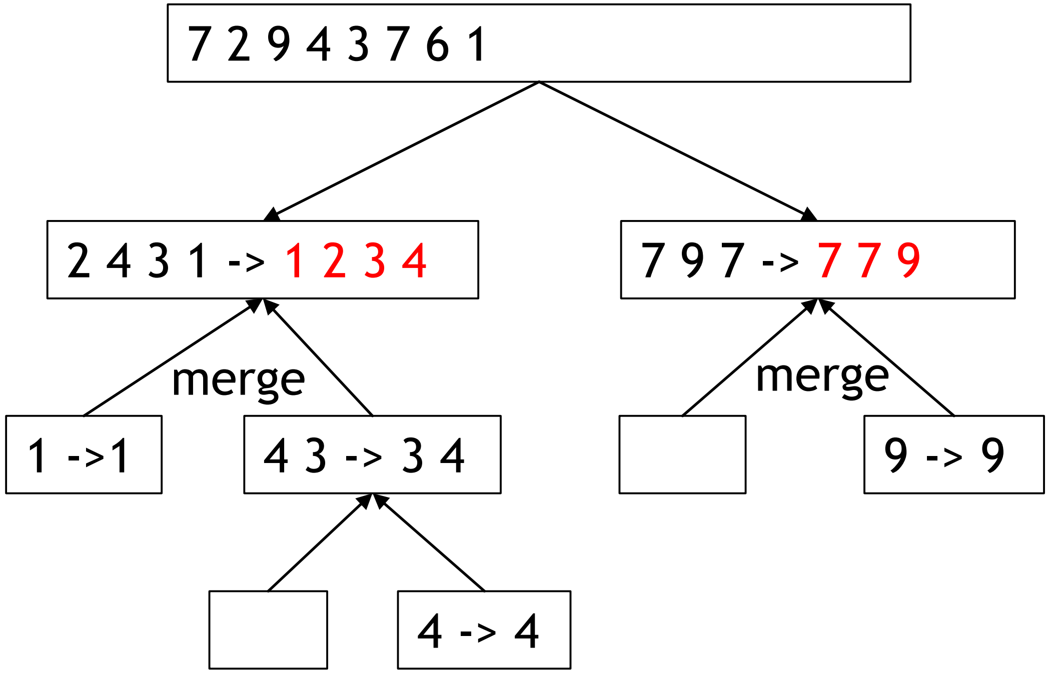
Review: Quick Sort



```
inPlaceQuickSort(A, l, r)
    if l >= r
        return
    i = random(l, r)
    (h, k) = inPlacePartition(A, i, l, r)
    inPlaceQuickSort(A, l, h-1)
    inPlaceQuickSort(A, k+1, r)


inPlacePartition(A, p, s, e)
    l = s
    r = e-1
    swap(A, p, e), p = e  // pivot swap to the last pos
    while l <= r
        while A[l] <A[p] and r >=l
            l = l+1
        while A[r] >= A[p] and r >=l
            r = r-1
    swap(A, l, p)
    return r+1, l
```

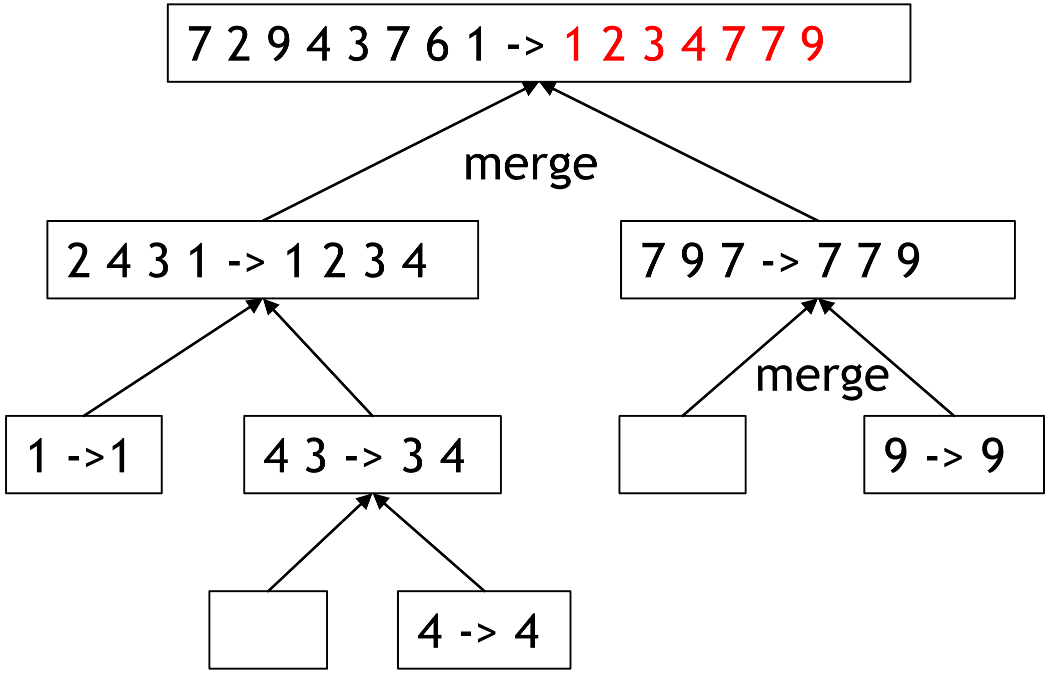
Review: Quick Sort



```
inPlaceQuickSort(A, l, r)
    if l >= r
        return
    i = random(l, r)
    (h, k) = inPlacePartition(A, i, l, r)
     inPlaceQuickSort(A, l, h-1)
     inPlaceQuickSort(A, k+1, r)


inPlacePartition(A, p, s, e)
    l = s
    r = e-1
    swap(A, p, e), p = e  // pivot swap to the last pos
    while l <= r
        while A[l] <A[p] and r >=l
            l = l+1
        while A[r] >= A[p] and r >=l
            r = r-1
    swap(A, l, p)
    return r+1, l
```

Review: Quick Sort



```
inPlaceQuickSort(A, l, r)
    if l >= r
        return
    i = random(l, r)
    (h, k) = inPlacePartition(A, i, l, r)
     inPlaceQuickSort(A, l, h-1)
     inPlaceQuickSort(A, k+1, r)


inPlacePartition(A, p, s, e)
    l = s
    r = e-1
    swap(A, p, e), p = e  // pivot swap to the last pos
    while l <= r
        while A[l] <A[p] and r >=l
            l = l+1
        while A[r] >= A[p] and r >=l
            r = r-1
    swap(A, l, p)
    return r+1, l
```

5. Suppose that Quicksort in-place is used to sort the following array where the pivot is always chosen to be the last number. Before recursively calling Quicksort in-place, the keys must be partitioned around the pivot. Write the content of the array after each swap.

												pivot
	1	2	3	4	10	15	25	20	1	30	5	18
	l										r	
Step 1:	1	2	3	4	10	15	25	20	1	30	5	18
Step 2:												
Step 3:												

5. Suppose that Quicksort in-place is used to sort the following array where the pivot is always chosen to be the last number. Before recursively calling Quicksort in-place, the keys must be partitioned around the pivot. Write the content of the array after each swap.

											pivot
1	2	3	4	10	15	25	20	1	30	5	18

		l									r	
Step 1:	1	2	3	4	10	15	25	20	1	30	5	18

Step 2:												

Step 3:												

5. Suppose that Quicksort in-place is used to sort the following array where the pivot is always chosen to be the last number. Before recursively calling Quicksort in-place, the keys must be partitioned around the pivot. Write the content of the array after each swap.

											pivot
1	2	3	4	10	15	25	20	1	30	5	18

Step 1:

		l								r	
1	2	3	4	10	15	25	20	1	30	5	18

Step 2:

Step 3:

5. Suppose that Quicksort in-place is used to sort the following array where the pivot is always chosen to be the last number. Before recursively calling Quicksort in-place, the keys must be partitioned around the pivot. Write the content of the array after each swap.

												pivot
	1	2	3	4	10	15	25	20	1	30	5	18
				l							r	
Step 1:	1	2	3	4	10	15	25	20	1	30	5	18
Step 2:												
Step 3:												

5. Suppose that Quicksort in-place is used to sort the following array where the pivot is always chosen to be the last number. Before recursively calling Quicksort in-place, the keys must be partitioned around the pivot. Write the content of the array after each swap.

											pivot
1	2	3	4	10	15	25	20	1	30	5	18

Step 1:

				l						r	
1	2	3	4	10	15	25	20	1	30	5	18

Step 2:

Step 3:

5. Suppose that Quicksort in-place is used to sort the following array where the pivot is always chosen to be the last number. Before recursively calling Quicksort in-place, the keys must be partitioned around the pivot. Write the content of the array after each swap.

												pivot
	1	2	3	4	10	15	25	20	1	30	5	18
						l					r	
Step 1:	1	2	3	4	10	15	25	20	1	30	5	18
Step 2:												
Step 3:												

5. Suppose that Quicksort in-place is used to sort the following array where the pivot is always chosen to be the last number. Before recursively calling Quicksort in-place, the keys must be partitioned around the pivot. Write the content of the array after each swap.

												pivot	
	1	2	3	4	10	15	25	20	1	30	5	18	
							l				r		
Step 1:	1	2	3	4	10	15	25	20	1	30	5	18	Swap(l, r)
Step 2:													
Step 3:													

5. Suppose that Quicksort in-place is used to sort the following array where the pivot is always chosen to be the last number. Before recursively calling Quicksort in-place, the keys must be partitioned around the pivot. Write the content of the array after each swap.

											pivot
1	2	3	4	10	15	25	20	1	30	5	18

Step 1:

						l				r	
1	2	3	4	10	15	5	20	1	30	25	18

Step 2:

							l		r		
1	2	3	4	10	15	5	20	1	30	25	18

Step 3:

5. Suppose that Quicksort in-place is used to sort the following array where the pivot is always chosen to be the last number. Before recursively calling Quicksort in-place, the keys must be partitioned around the pivot. Write the content of the array after each swap.

											pivot
1	2	3	4	10	15	25	20	1	30	5	18

Step 1:

						l				r	
1	2	3	4	10	15	5	20	1	30	25	18

Step 2:

							l	r			
1	2	3	4	10	15	5	20	1	30	25	18

Swap(l, r)

Step 3:

5. Suppose that Quicksort in-place is used to sort the following array where the pivot is always chosen to be the last number. Before recursively calling Quicksort in-place, the keys must be partitioned around the pivot. Write the content of the array after each swap.

											pivot
1	2	3	4	10	15	25	20	1	30	5	18

Step 1:

						l				r	
1	2	3	4	10	15	5	20	1	30	25	18

Step 2:

							l	r			
1	2	3	4	10	15	5	1	20	30	25	18

Step 3:

								l/r			
1	2	3	4	10	15	5	1	20	30	25	18

5. Suppose that Quicksort in-place is used to sort the following array where the pivot is always chosen to be the last number. Before recursively calling Quicksort in-place, the keys must be partitioned around the pivot. Write the content of the array after each swap.

											pivot
1	2	3	4	10	15	25	20	1	30	5	18

Step 1:

						l				r	
1	2	3	4	10	15	5	20	1	30	25	18

Step 2:

							l	r			
1	2	3	4	10	15	5	1	20	30	25	18

Step 3:

								r			
1	2	3	4	10	15	5	1	18	30	25	20
								l			

Swap(l, pivot)