Quick, Heap and Shell Sorts

Kuan-Yu Chen (陳冠宇)

2020/11/30 @ TR-313, NTUST

Sorting

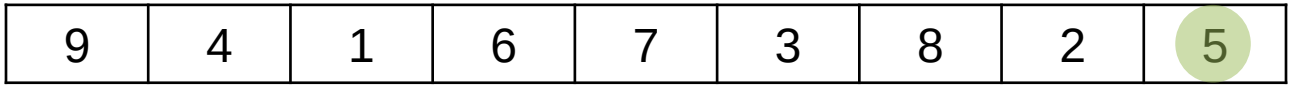
- Sorting means arranging the elements of an array so that they are placed in some relevant order which may be either ascending or descending

- A sorting algorithm is defined as an algorithm that puts the elements of a list in a certain order, which can be either numerical order, lexicographical order, or any user-defined order
  - **Bubble, Insertion, Tree**
  - **Selection, Merge, Radix**
  - Quick, Heap, Shell

Quick Sort.

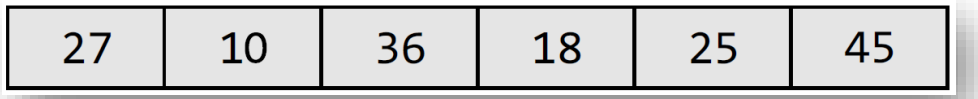
- Quick sort is a widely used sorting algorithm developed by C. A. R. Hoare
  - Quick sort is also known as **partition exchange sort**

- The quick sort algorithm works as follows:
  1. Select an element **pivot** from the array elements
  2. Rearrange the elements in the array in such a way that all elements that are less than the pivot appear before the pivot and all elements greater than the pivot element come after it
  3. Recursively sort the two sub-arrays thus obtained

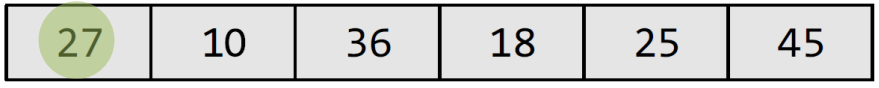


Example.

- Sort the given array using quick sort algorithm



We choose the first element as the pivot.
Set `loc` = 0, `left` = 0, and `right` = 5.



`loc`
`left`

`right`

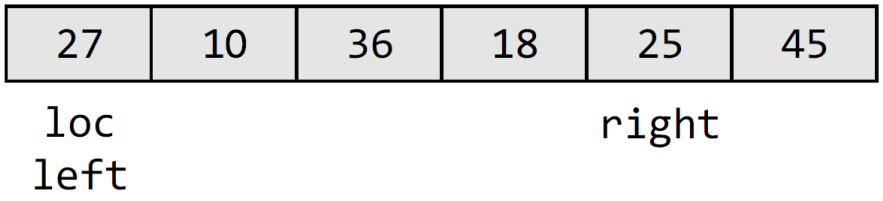
Scan from right to left. Since `a[loc]` < `a[right]`, decrease the value of `right`.

`loc`
`left`

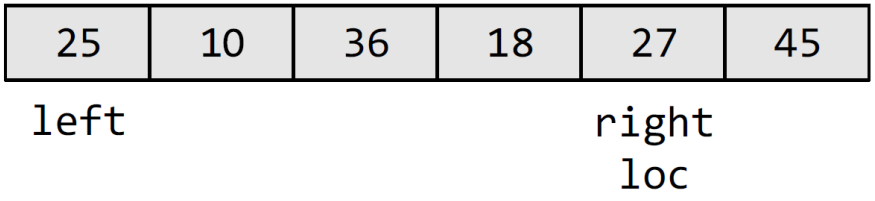
`right`

Example..

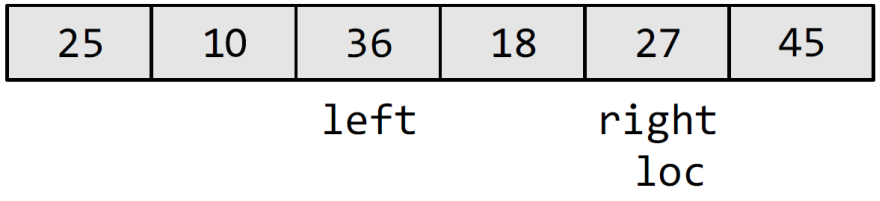
Scan from right to left. Since `a[loc]` < `a[right]`, decrease the value of `right`.



Since `a[loc]` > `a[right]`, interchange the two values and set `loc = right`.

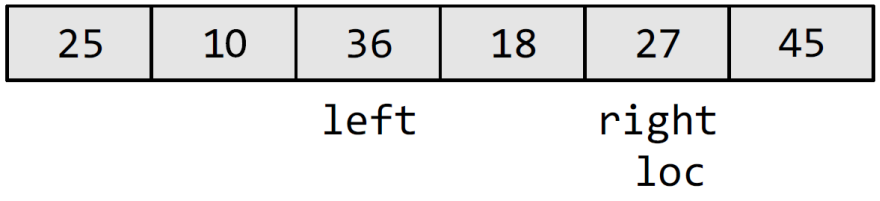


Start scanning from left to right. Since `a[loc]` > `a[left]`, increment the value of `left`.

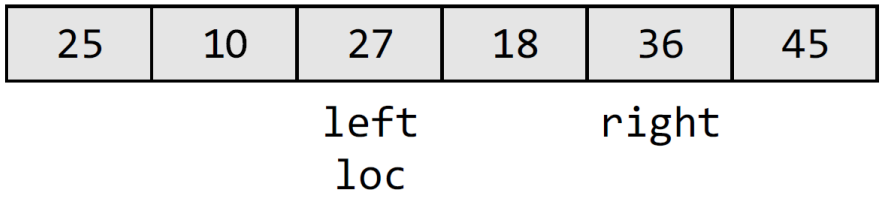


Example…

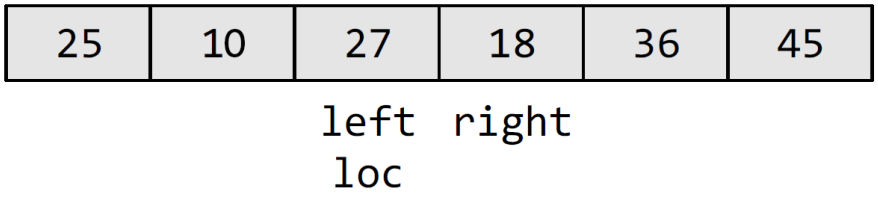
Start scanning from left to right. Since a[loc] > a[left], increment the value of left.



Since a[loc] < a[left], interchange the values and set loc = left.

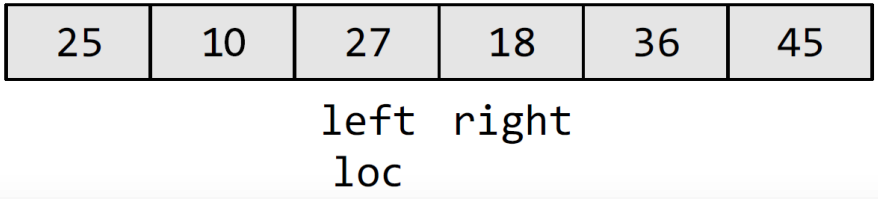


Scan from right to left. Since a[loc] < a[right], decrement the value of right.

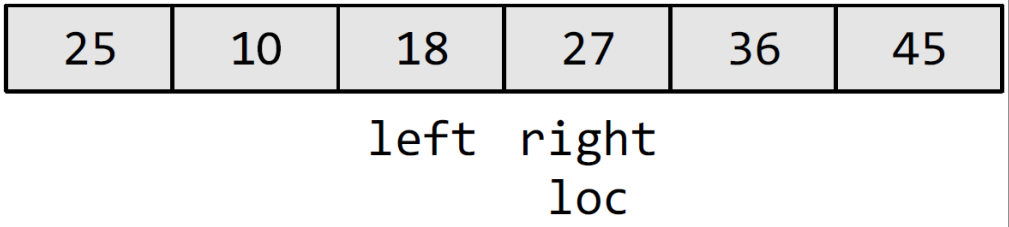


Example….

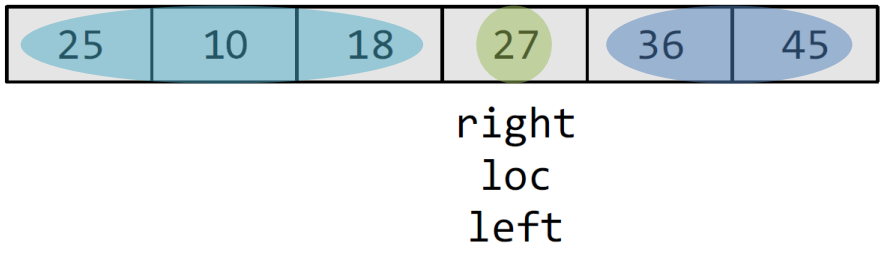
Scan from right to left. Since `a[loc]` < `a[right]`, decrement the value of `right`.



Since `a[loc]` > `a[right]`, interchange the two values and set `loc = right`.



Start scanning from left to right. Since `a[loc]` > `a[left]`, increment the value of `left`.



Quick Sort..

```
QUICKSORT (ARR, BEG, END)

Step 1: IF (BEG < END)
            CALL PARTITION (ARR, BEG, END, LOC)
            CALL QUICKSORT(ARR, BEG, LOC - 1)
            CALL QUICKSORT(ARR, LOC + 1, END)
        [END OF IF]
Step 2: END
```

Quick Sort…

```
PARTITION (ARR, BEG, END, LOC)

Step 1: [INITIALIZE] SET LEFT = BEG, RIGHT = END, LOC = BEG, FLAG = 0
Step 2: Repeat Steps 3 to 6 while FLAG = 0
Step 3: Repeat while ARR[LOC] <= ARR[RIGHT] AND LOC != RIGHT
                SET RIGHT = RIGHT - 1
        [END OF LOOP]
Step 4: IF LOC = RIGHT
                SET FLAG = 1
        ELSE IF ARR[LOC] > ARR[RIGHT]
                SWAP ARR[LOC] with  ARR[RIGHT]
                SET LOC = RIGHT
        [END OF IF]
Step 5: IF FLAG = 0
                Repeat while ARR[LOC] >= ARR[LEFT] AND LOC != LEFT
                SET LEFT = LEFT + 1
                [END OF LOOP]
Step 6:         IF LOC = LEFT
                        SET FLAG = 1
                ELSE IF ARR[LOC] < ARR[LEFT]
                        SWAP ARR[LOC] with  ARR[LEFT]
                        SET LOC = LEFT
                [END OF IF]
        [END OF IF]
Step 7: [END OF LOOP]
Step 8: END
```

Quick Sort… - 1

- Sort the given array using quick sort algorithm

27	10	36	18	25	45

```
PARTITION (ARR, BEG, END, LOC)

Step 1: [INITIALIZE] SET LEFT = BEG, RIGHT = END, LOC = BEG, FLAG = 0
Step 2: Repeat Steps 3 to 6 while FLAG = 0
Step 3: Repeat while ARR[LOC] <= ARR[RIGHT] AND LOC != RIGHT
                SET RIGHT = RIGHT - 1
        [END OF LOOP]
Step 4: IF LOC = RIGHT
                SET FLAG = 1
        ELSE IF ARR[LOC] > ARR[RIGHT]
                SWAP ARR[LOC] with  ARR[RIGHT]
                SET LOC = RIGHT
        [END OF IF]
Step 5: IF FLAG = 0
                Repeat while ARR[LOC] >= ARR[LEFT] AND LOC != LEFT
                SET LEFT = LEFT + 1
                [END OF LOOP]
Step 6:         IF LOC = LEFT
                        SET FLAG = 1
                ELSE IF ARR[LOC] < ARR[LEFT]
                        SWAP ARR[LOC] with  ARR[LEFT]
                        SET LOC = LEFT
                [END OF IF]
        [END OF IF]
Step 7: [END OF LOOP]
Step 8: END
```

We choose the first element as the pivot. Set `loc` = 0, `left` = 0, and `right` = 5.

27	10	36	18	25	45

loc, left — right

Scan from right to left. Since `a[loc]` < `a[right]`, decrease the value of `right`.

27	10	36	18	25	45

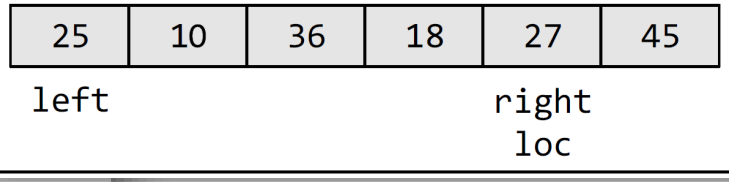
loc, left — right

Quick Sort… - 2

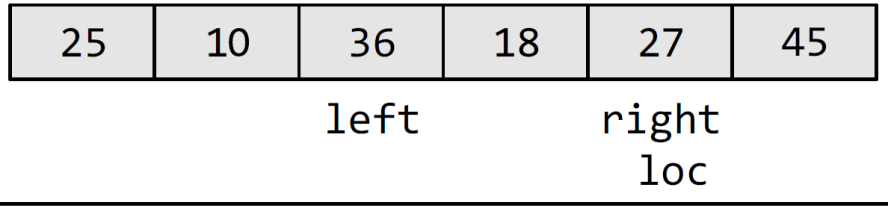
```
PARTITION (ARR, BEG, END, LOC)

Step 1: [INITIALIZE] SET LEFT = BEG, RIGHT = END, LOC = BEG, FLAG = 0
Step 2: Repeat Steps 3 to 6 while FLAG = 0
Step 3: Repeat while ARR[LOC] <= ARR[RIGHT] AND LOC != RIGHT
                SET RIGHT = RIGHT - 1
        [END OF LOOP]
Step 4: IF LOC = RIGHT
                SET FLAG = 1
        ELSE IF ARR[LOC] > ARR[RIGHT]
                SWAP ARR[LOC] with  ARR[RIGHT]
                SET LOC = RIGHT
        [END OF IF]
Step 5: IF FLAG = 0
                Repeat while ARR[LOC] >= ARR[LEFT] AND LOC != LEFT
                SET LEFT = LEFT + 1
                [END OF LOOP]
Step 6:         IF LOC = LEFT
                        SET FLAG = 1
                ELSE IF ARR[LOC] < ARR[LEFT]
                        SWAP ARR[LOC] with  ARR[LEFT]
                        SET LOC = LEFT
                [END OF IF]
        [END OF IF]
Step 7: [END OF LOOP]
Step 8: END
```

Since `a[loc] > a[right]`, interchange the two values and set `loc = right`.



Start scanning from left to right. Since `a[loc] > a[left]`, increment the value of `left`.

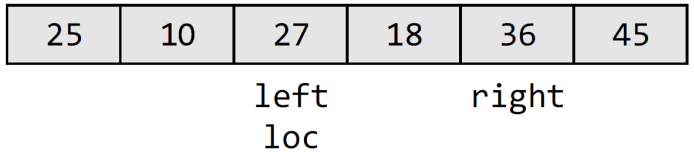


Quick Sort… - 3

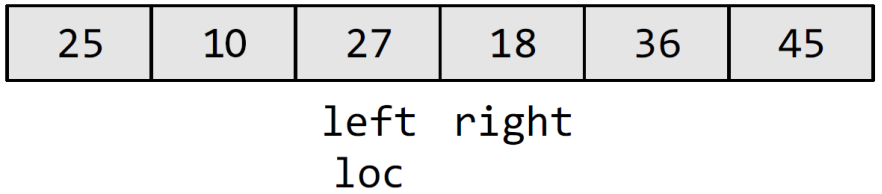
```
PARTITION (ARR, BEG, END, LOC)

Step 1: [INITIALIZE] SET LEFT = BEG, RIGHT = END, LOC = BEG, FLAG = 0
Step 2: Repeat Steps 3 to 6 while FLAG = 0
Step 3: Repeat while ARR[LOC] <= ARR[RIGHT] AND LOC != RIGHT
                SET RIGHT = RIGHT - 1
        [END OF LOOP]
Step 4: IF LOC = RIGHT
                SET FLAG = 1
        ELSE IF ARR[LOC] > ARR[RIGHT]
                SWAP ARR[LOC] with  ARR[RIGHT]
                SET LOC = RIGHT
        [END OF IF]
Step 5: IF FLAG = 0
                Repeat while ARR[LOC] >= ARR[LEFT] AND LOC != LEFT
                SET LEFT = LEFT + 1
                [END OF LOOP]
Step 6:         IF LOC = LEFT
                        SET FLAG = 1
                ELSE IF ARR[LOC] < ARR[LEFT]
                        SWAP ARR[LOC] with  ARR[LEFT]
                        SET LOC = LEFT
                [END OF IF]
        [END OF IF]
Step 7: [END OF LOOP]
Step 8: END
```

Since `a[loc] < a[left]`, interchange the values and set `loc = left`.



Scan from right to left. Since `a[loc] < a[right]`, decrement the value of `right`.

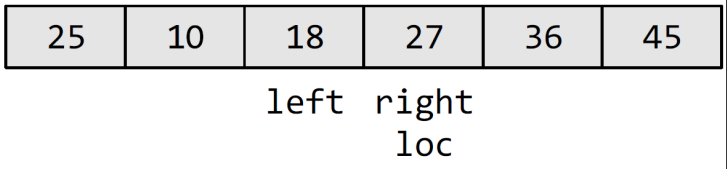


Quick Sort… - 4

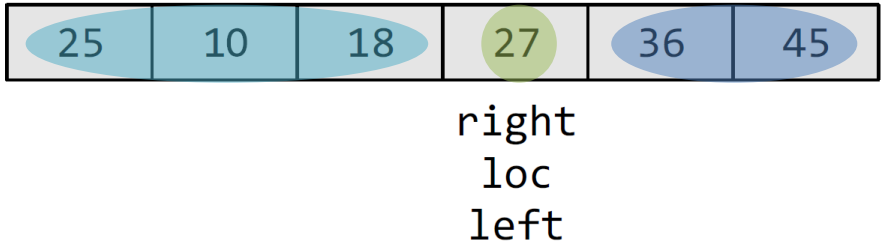
```
PARTITION (ARR, BEG, END, LOC)

Step 1: [INITIALIZE] SET LEFT = BEG, RIGHT = END, LOC = BEG, FLAG = 0
Step 2: Repeat Steps 3 to 6 while FLAG = 0
Step 3: Repeat while ARR[LOC] <= ARR[RIGHT] AND LOC != RIGHT
                SET RIGHT = RIGHT - 1
        [END OF LOOP]
Step 4: IF LOC = RIGHT
                SET FLAG = 1
        ELSE IF ARR[LOC] > ARR[RIGHT]
                SWAP ARR[LOC] with  ARR[RIGHT]
                SET LOC = RIGHT
        [END OF IF]
Step 5: IF FLAG = 0
                Repeat while ARR[LOC] >= ARR[LEFT] AND LOC != LEFT
                SET LEFT = LEFT + 1
                [END OF LOOP]
Step 6:         IF LOC = LEFT
                        SET FLAG = 1
                ELSE IF ARR[LOC] < ARR[LEFT]
                        SWAP ARR[LOC] with  ARR[LEFT]
                        SET LOC = LEFT
                [END OF IF]
        [END OF IF]
Step 7: [END OF LOOP]
Step 8: END
```

Since `a[loc] > a[right]`, interchange the two values and set `loc = right`.



Start scanning from left to right. Since `a[loc] > a[left]`, increment the value of `left`.



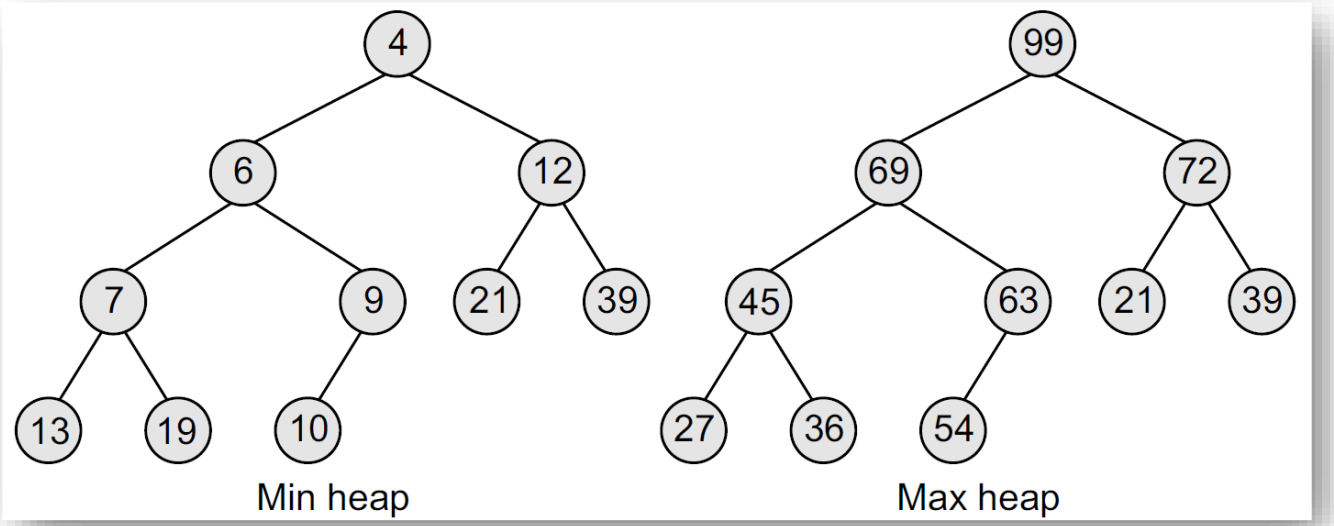
Heap

- A binary heap is a **complete binary tree** in which every node satisfies the heap property
  - Min Heap

$$If\ B\ is\ a\ child\ of\ A, then\ key(B)\ \geq\ key(A)$$

  - Max Heap

$$If\ B\ is\ a\ child\ of\ A, then\ key(A)\ \geq\ key(B)$$

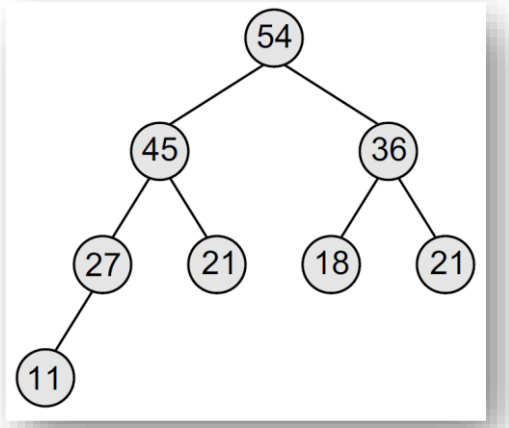


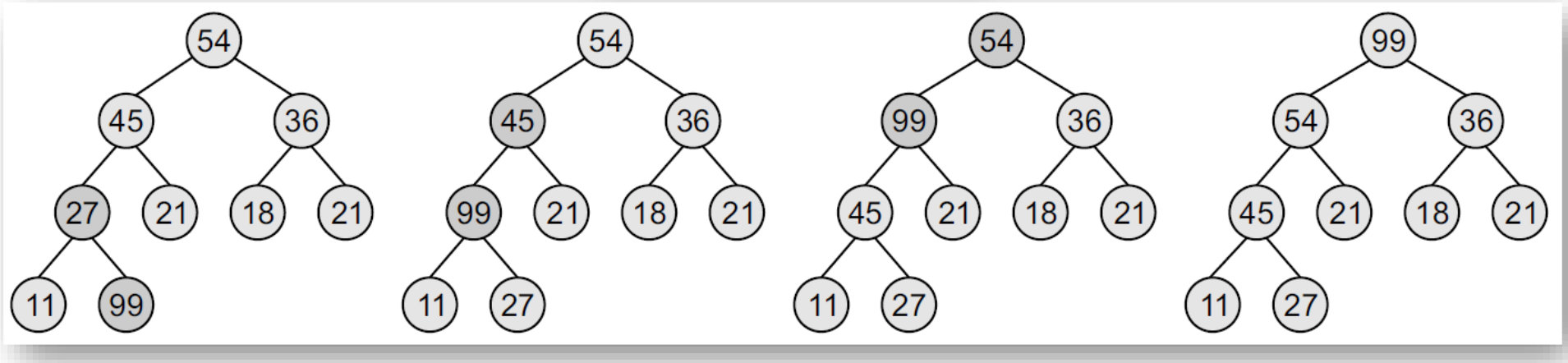
Heap – Insertion

- Inserting a new value into the heap is done in the following two steps:
  - Consider a max heap H with n elements
  1. Add the new value at the bottom of H
  2. Let the new value rise to its appropriate place in H

Example

- Consider a max heap and insert 99 in it





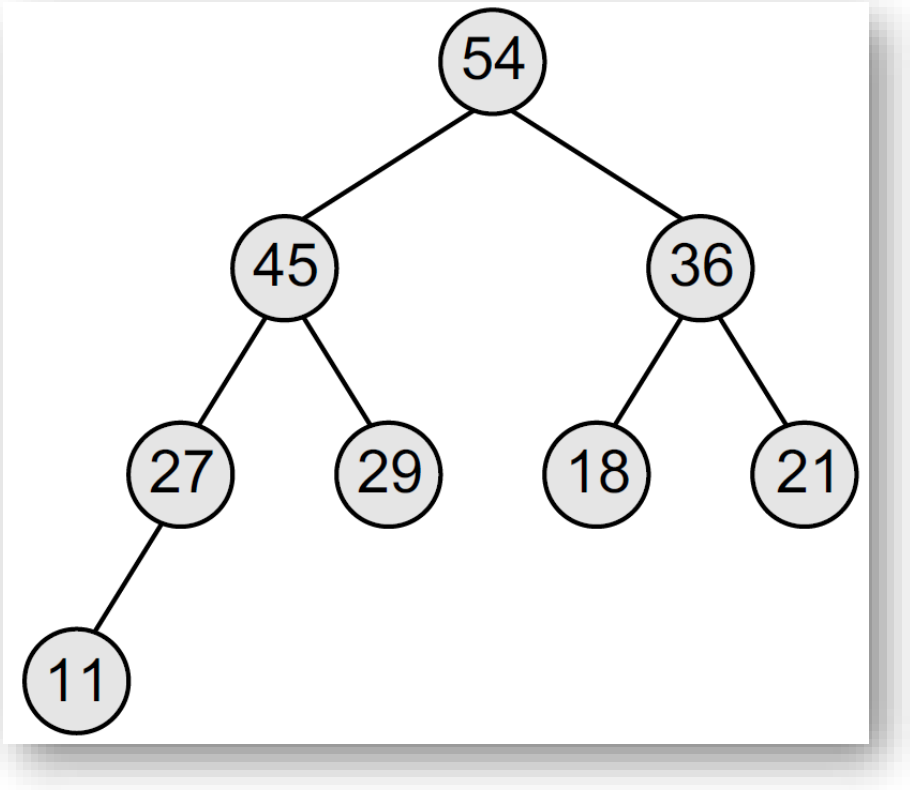
Heap – Deletion

- An element is **always deleted from the root** of the heap

- Consider a max heap H having n elements, deleting an element from the heap is done in the following three steps:
  1. Replace the root node's value with the last node's value
  2. Delete the last node
  3. Sink down the new root node's value so that H satisfies the heap property

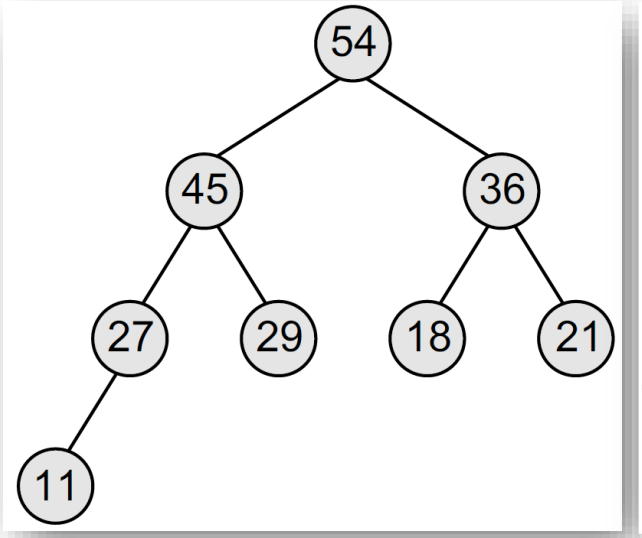
Example.

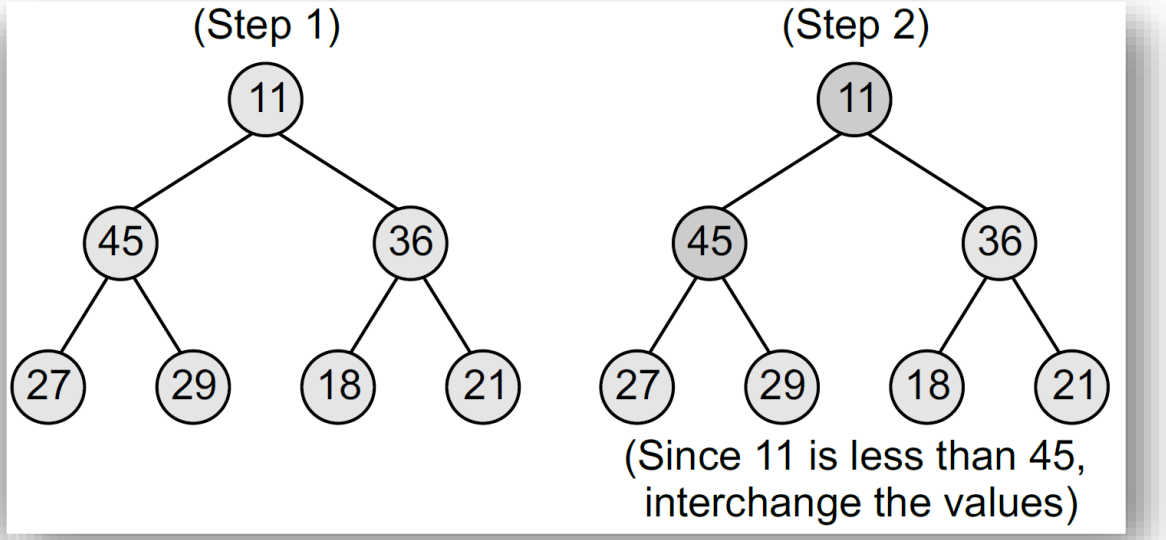
- Delete the root node's value from a given max heap *H*

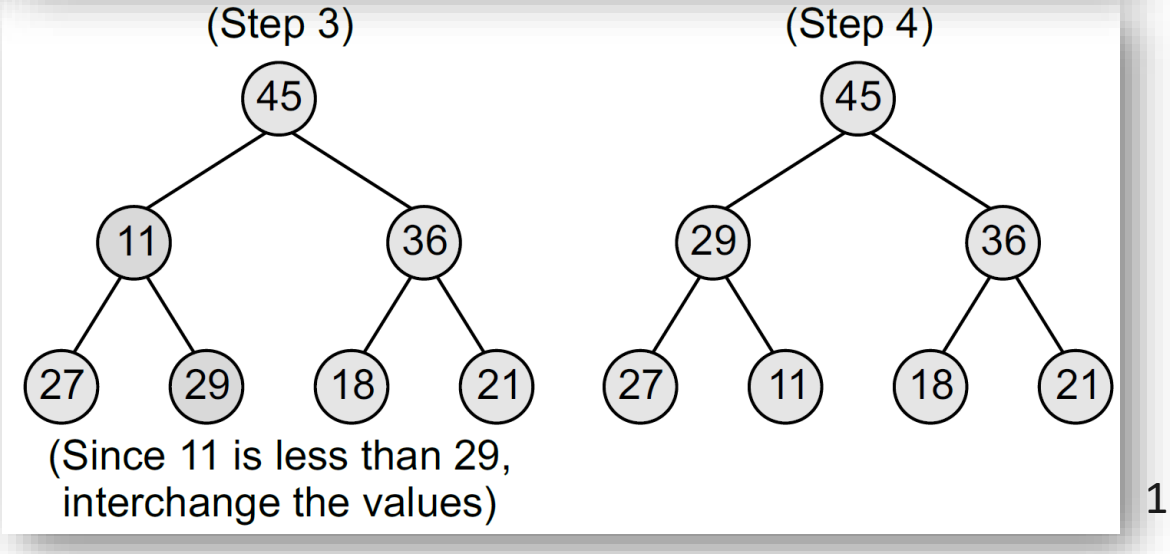


Example..

- Delete the root node's value from a given max heap *H*







Heap Sort

- Given an array ARR with n elements, the heap sort algorithm can be used to sort ARR in two phases
  - In phase 1, build a heap H using the elements of ARR
  - In phase 2, repeatedly delete the root element of the heap formed in phase 1

Shell Sort.

- Shell sort, invented by Donald Shell in 1959, is a sorting algorithm that is a generalization of insertion sort
  - First, insertion sort works well when the input data is "almost sorted"
  - Second, insertion sort is quite inefficient to use as it moves the values just one position at a time

```
Shell_Sort(Arr, N)

Step 1: SET FLAG = 1, GAP_SIZE = N
Step 2: Repeat Steps 3 to 6 while FLAG = 1 OR GAP_SIZE > 1
Step 3:     SET FLAG = 0
Step 4:     SET GAP_SIZE = (GAP_SIZE + 1) / 2
Step 5:     Repeat Step 6 for I = 0 to I < (N - GAP_SIZE)
Step 6:           IF Arr[I + GAP_SIZE] < Arr[I]
                        SWAP Arr[I + GAP_SIZE], Arr[I]
                        SET FLAG = 1
Step 7: END
```

Example.

- Sort the elements using shell sort

```
63,  19,  7,  90,  81,  36,  54,  45,  72,  27,  22,  9,  41,  59,  33
```

  - The first pass: $gap = \frac{15+1}{2}$

Arrange the elements of the array in the form of a table and sort the columns.

63	19	7	90	81	36	54	45
72	27	22	9	41	59	33	

Result:

63	19	7	9	41	36	33	45
72	27	22	90	81	59	54	

The elements of the array can be given as:

```
63,  19,  7,  9,  41,  36,  33,  45,  72,  27,  22,  90,  81,  59,  54
```

Example..

- The second pass: $gap = \frac{8+1}{2}$

					Result:				
63	19	7	9	41	22	19	7	9	27
36	33	45	72	27	36	33	45	59	41
22	90	81	59	54	63	90	81	72	54

The elements of the array can be given as:

22, 19, 7, 9, 27, 36, 33, 45, 59, 41, 63, 90, 81, 72, 54

- The third pass: $gap = \frac{5+1}{2}$

			Result:		
22	19	7	9	19	7
9	27	36	22	27	36
33	45	59	33	45	54
41	63	90	41	63	59
81	72	54	81	72	90

The elements of the array can be given as:

9, 19, 7, 22, 27, 36, 33, 45, 54, 41, 63, 59, 81, 72, 90

Example…

- The last step: $gap = 1$

	Result:
9	7
19	9
7	19
22	22
27	27
36	33
33	36
45	41
54	45
41	54
63	59
59	63
81	72
72	81
90	90

Finally, the elements of the array can be given as:

7, 9, 19, 22, 27, 33, 36, 41, 45, 54, 59, 63, 72, 81, 90

Shell Sort..

```
Shell_Sort(Arr, N)

Step 1: SET FLAG = 1, GAP_SIZE = N
Step 2: Repeat Steps 3 to 6 while FLAG = 1 OR GAP_SIZE > 1
Step 3:     SET FLAG = 0
Step 4:     SET GAP_SIZE = (GAP_SIZE + 1) / 2
Step 5:     Repeat Step 6 for I = 0 to I < (N - GAP_SIZE)
Step 6:           IF Arr[I + GAP_SIZE] < Arr[I]
                        SWAP Arr[I + GAP_SIZE], Arr[I]
                        SET FLAG = 1
Step 7: END
```

- $Gap = 5$

63	19	7	9	41
36	33	45	72	27
22	90	81	59	54

Result:

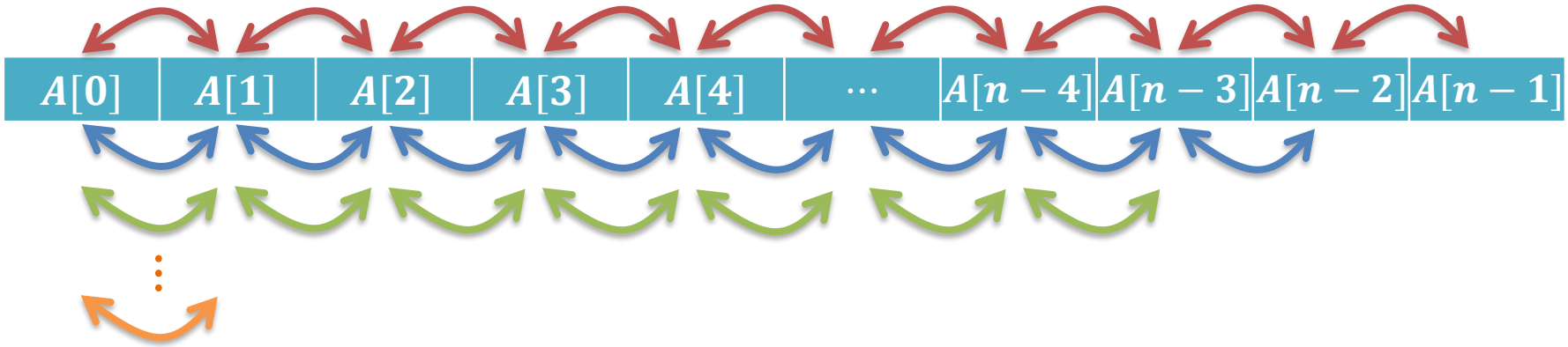
22	19	7	9	27
36	33	45	59	41
63	90	81	72	54

The elements of the array can be given as:

22, 19, 7, 9, 27, 36, 33, 45, 59, 41, 63, 90, 81, 72, 54

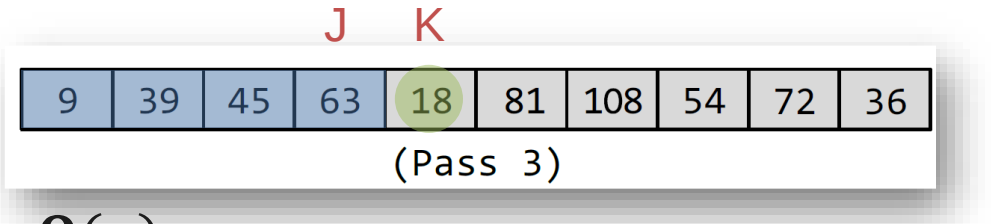
Review.

- Bubble Sort



  - Best/Worst/Average Case: $\mathbf{O}(n^2)$

- Insertion Sort



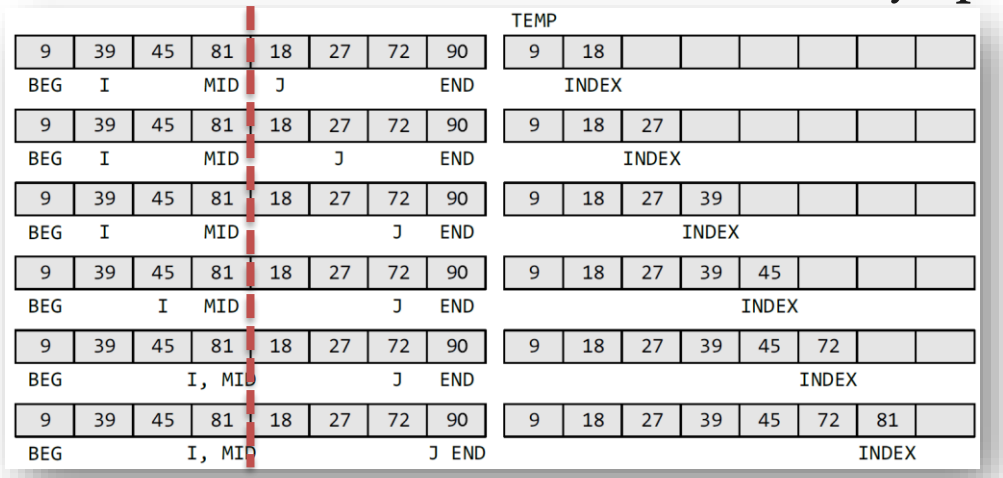
  - Best Case: $\mathbf{O}(n)$
  - Worst Case: $\mathbf{O}(n^2)$

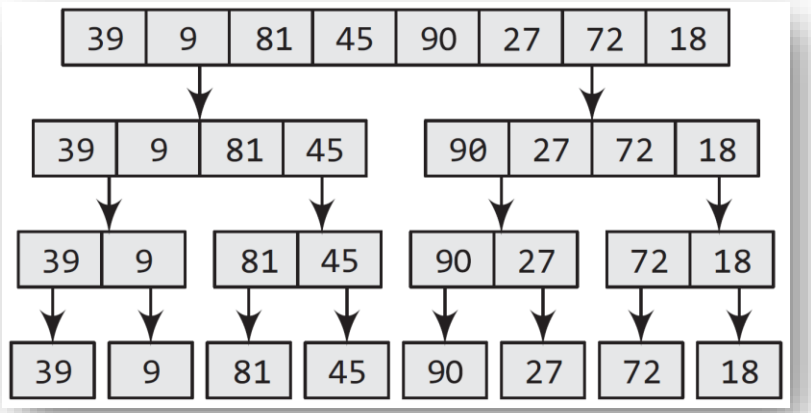
Review…

- Selection Sort
  - Average/Best/Worst Case: $\mathbf{O}(n^2)$

PASS	ARR[0]	ARR[1]	ARR[2]	ARR[3]	ARR[4]	ARR[5]	ARR[6]	ARR[7]
1	9	39	81	45	90	27	72	18
2	9	18	81	45	90	27	72	39
3	9	18	27	45	90	81	72	39
4	9	18	27	39	90	81	72	45
5	9	18	27	39	45	81	72	90
6	9	18	27	39	45	72	81	90
7	9	18	27	39	45	72	81	90

- Merge Sort
  - Average/Best/Worst Case: $\mathbf{O}(n\log n)$
  - It needs an additional memory space





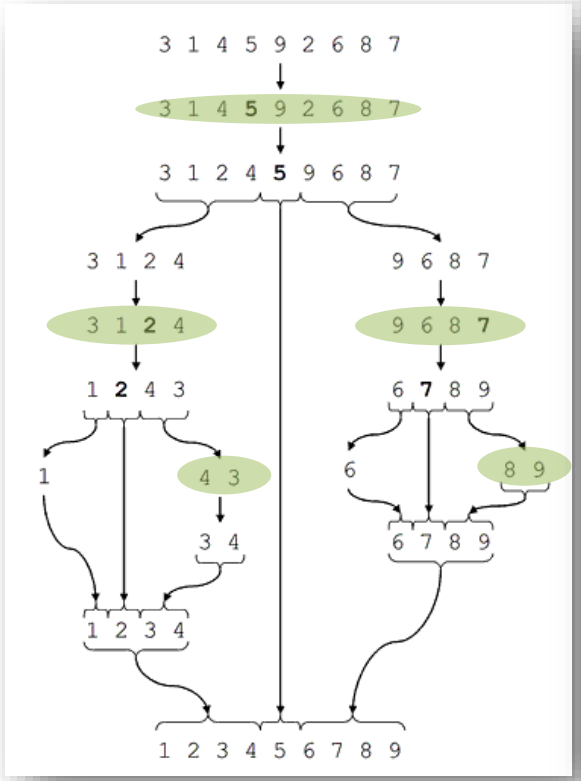
https://stackoverflow.com/questions/39665299/understanding-quicksort

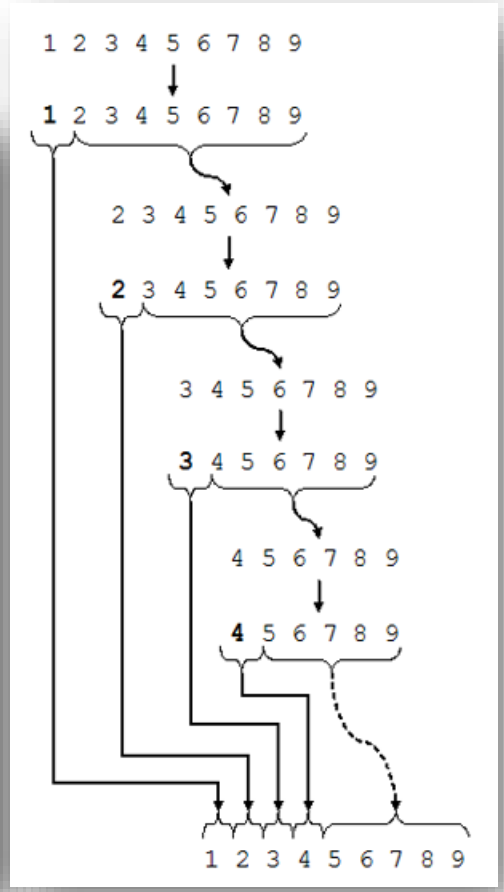
Review….

- Quick Sort

  - Best Case: $\mathbf{O}(n\log n)$
  - Worst Case: $\mathbf{O}(n^2)$

- Tree Sort
  - Best Case: $\mathbf{O}(n\log n)$
    - Add a node is $\mathbf{O}(\log n)$
  - Worst Case: $\mathbf{O}(n^2)$
    - Add a node is $\mathbf{O}(n)$





Review.....

- Radix Sort
  - Best/Worst/Average Case: $\mathbf{O}(kn)$
    - k is the number of digits of the largest element

Number	0	1	2	3	4	5	6	7	8	9
911		911								
472								472		
123			123							
654						654				
924			924							
345					345					
555						555				
567							567			
808	808									

- Shell Sort
  - Best Case: ?
    - The best case for Insertion Sort is $\mathbf{O}(n)$
  - Worst Case: $\mathbf{O}(n^2)$
    - Insertion Sort

- Heap Sort
  - Average/Best/Worst Case: $\mathbf{O}(n\log n)$
    - A Complete Binary Tree

      Balance Tree

Questions?

kychen@mail.ntust.edu.tw