Shell Sort, Heap, Heap Sort & Heap Variants

Kuan-Yu Chen (陳冠宇)

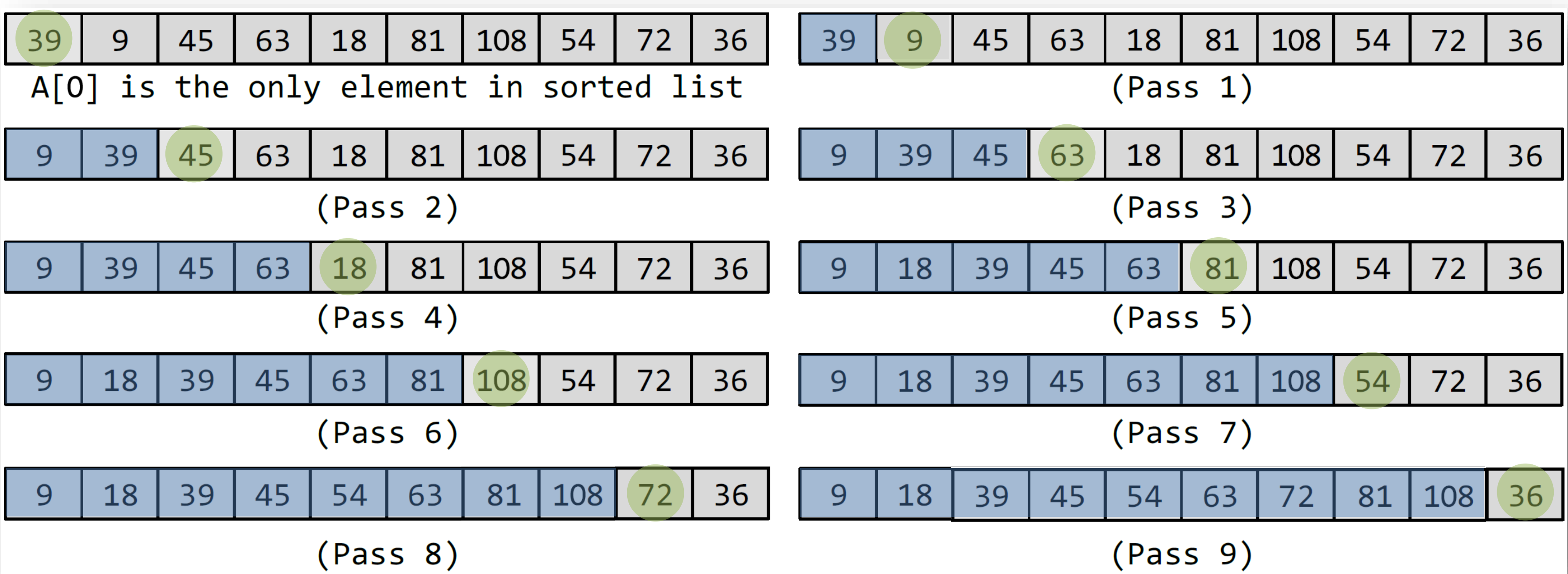
Sorting

- 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**, Shell
  - **Quick, Radix,** Heap

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



Example.

- Sort the 15 elements using shell sort

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

  - The first pass: $gap = \frac{15+1}{2} = 8$

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: $gap = \frac{8+1}{2} = 4.5$

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

– The third pass: $gap = \frac{5+1}{2} = 3$

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

Result:

9	19	7
22	27	36
33	45	54
41	63	59
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 GAP_SIZE=N
Step 2: Repeat Steps 3 to 5 while GAP_SIZE > 1
Step 3:         SET GAP_SIZE = (GAP_SIZE + 1) / 2
Step 4:         for I = 0 to I < GAP_SIZE
Step 5:                 insertion sort()
Step 6: END
```

insertion sort()

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

https://www.youtube.com/watch?v=CmPA7zE8mx0

Shell Sort..

$gap = 5$

$gap = 3$

$gap = 1$

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

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

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

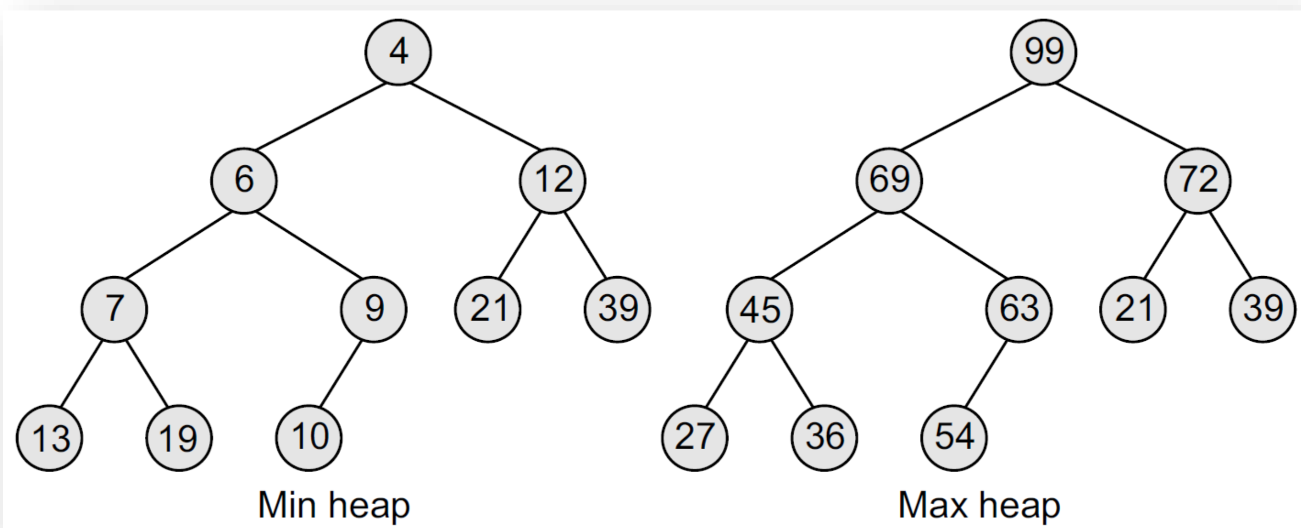
Binary Heap

- 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

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



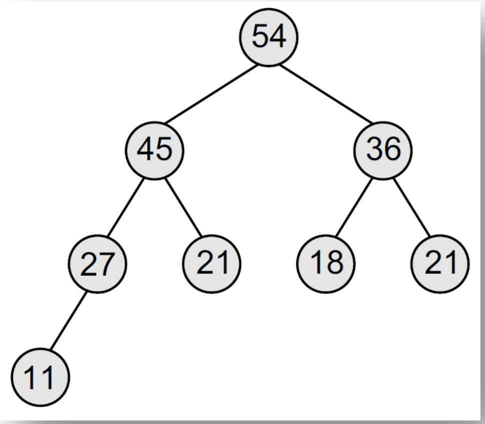
Binary Heap – Insertion

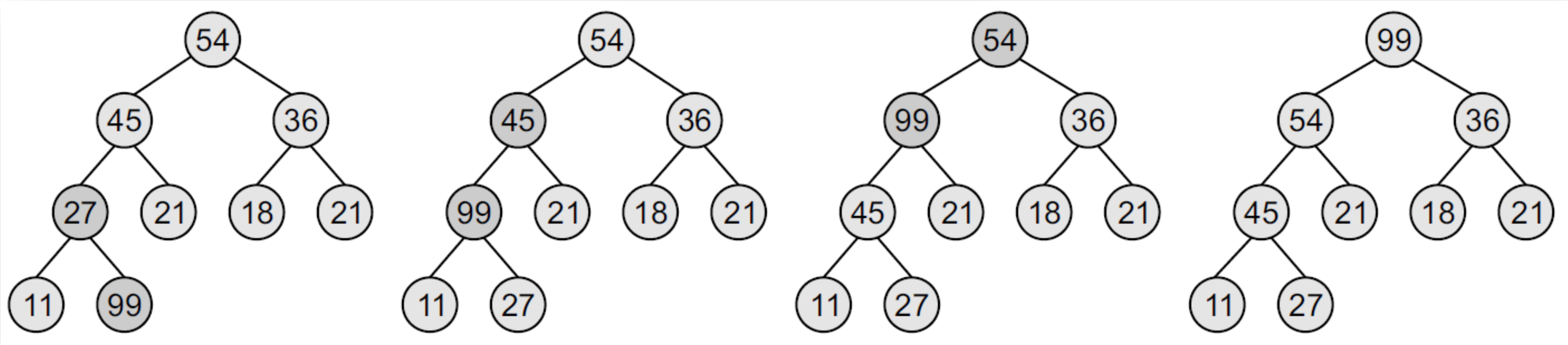
- Inserting a new value into a binary heap is done in the following two steps:
  - Consider a max heap H with n elements

  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





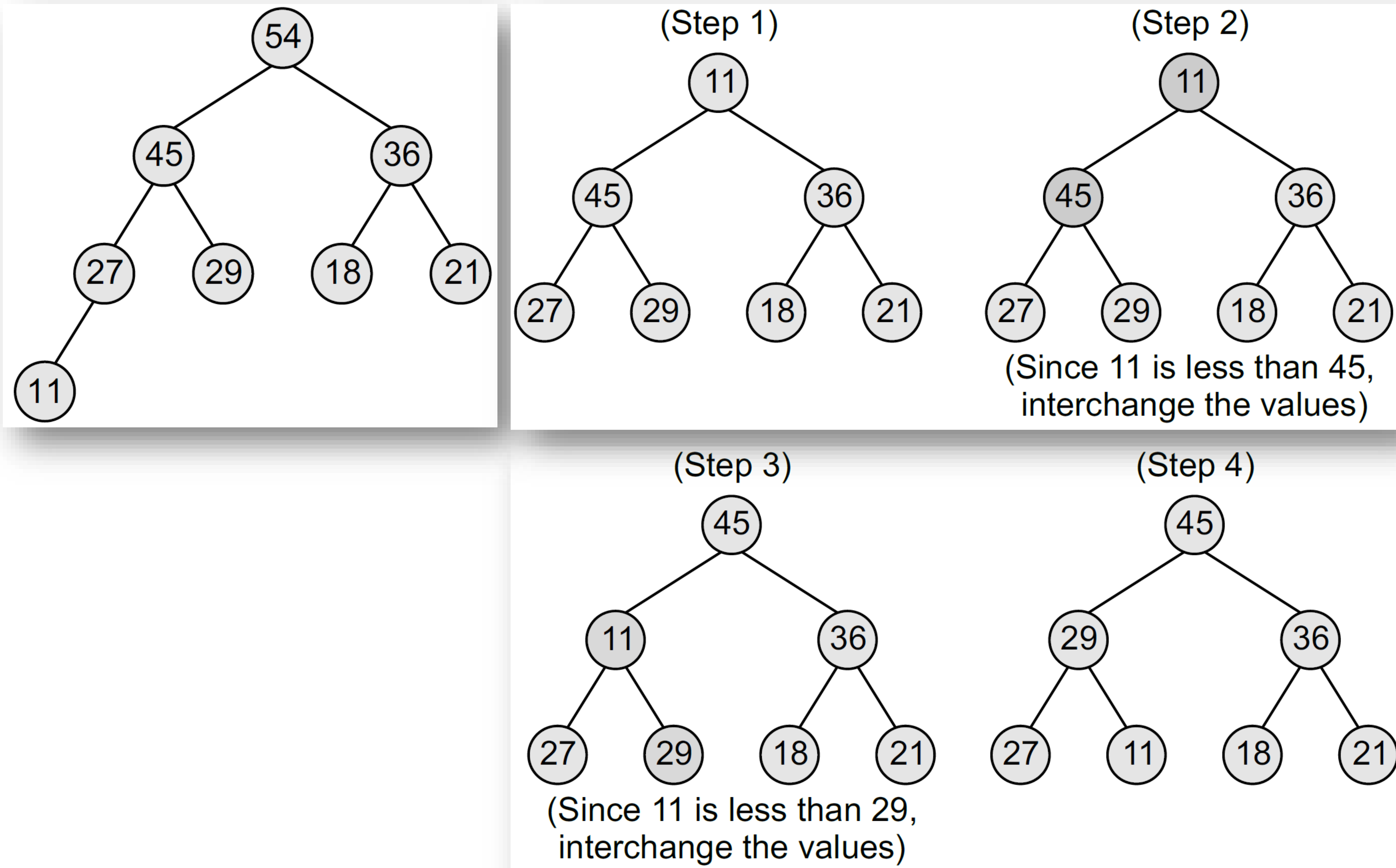
Binary Heap – Deletion

- 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 an element from the given **max** heap *H*

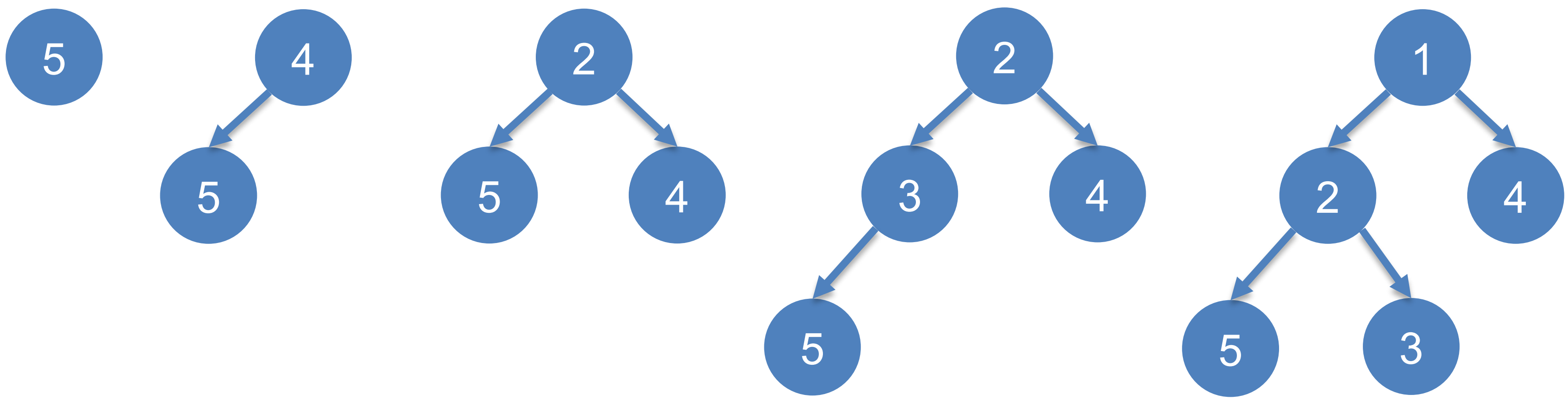


Heap Sort

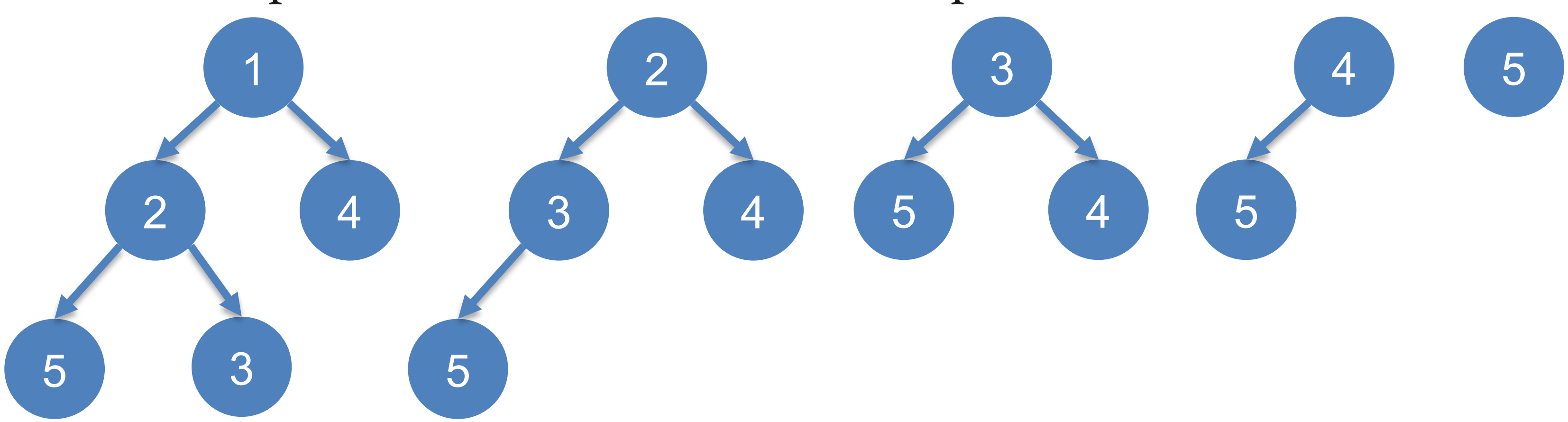
- Given an array ARR with n elements, the heap sort algorithm can be used to sort ARR in two phases
  - In phase 1, build a binary heap H using the elements of ARR
  - In phase 2, repeatedly delete an element from the heap

Example

- Sort the given elements using heap sort algorithm
  - 5, 4, 2, 3, 1
    - Step 1: build a minimum heap

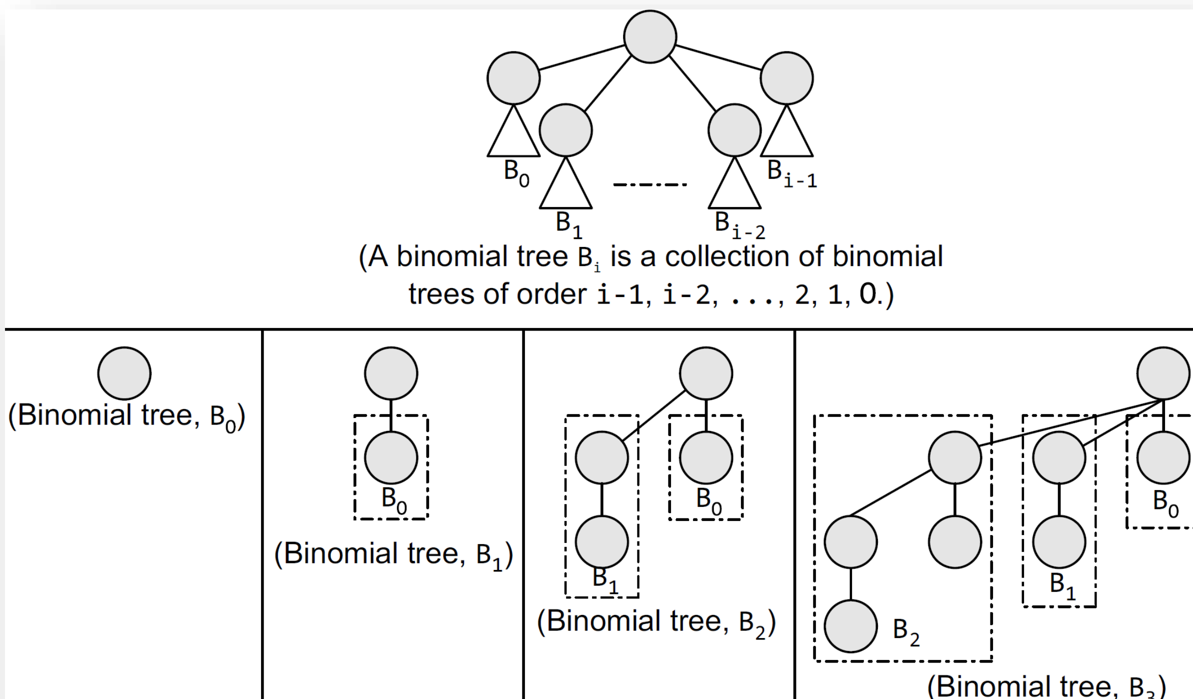


    - Step 2: delete elements from the heap



Binomial Tree

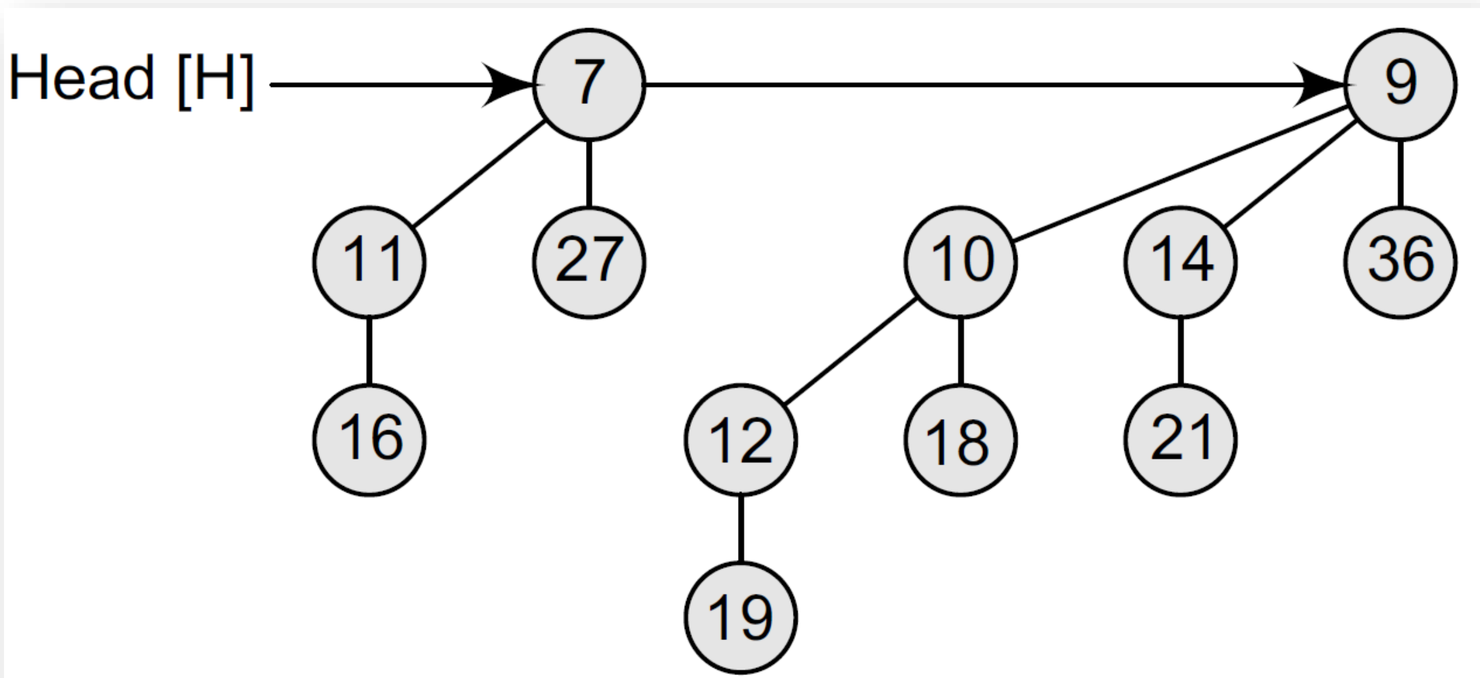
- A **binomial tree** is an ordered tree
  - A binomial tree B_i with order i has 2^i nodes
    - It contains a root node whose children are the root nodes of binomial trees of order $i-1,\ i-2, \cdots,\ 2,\ 1,$ and 0
    - The height of a binomial tree B_i is i
  - A binomial tree of order 0 has a single node



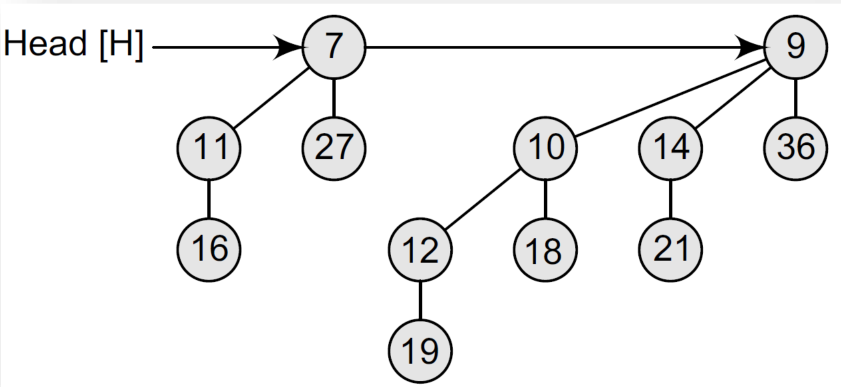
Binomial Heap

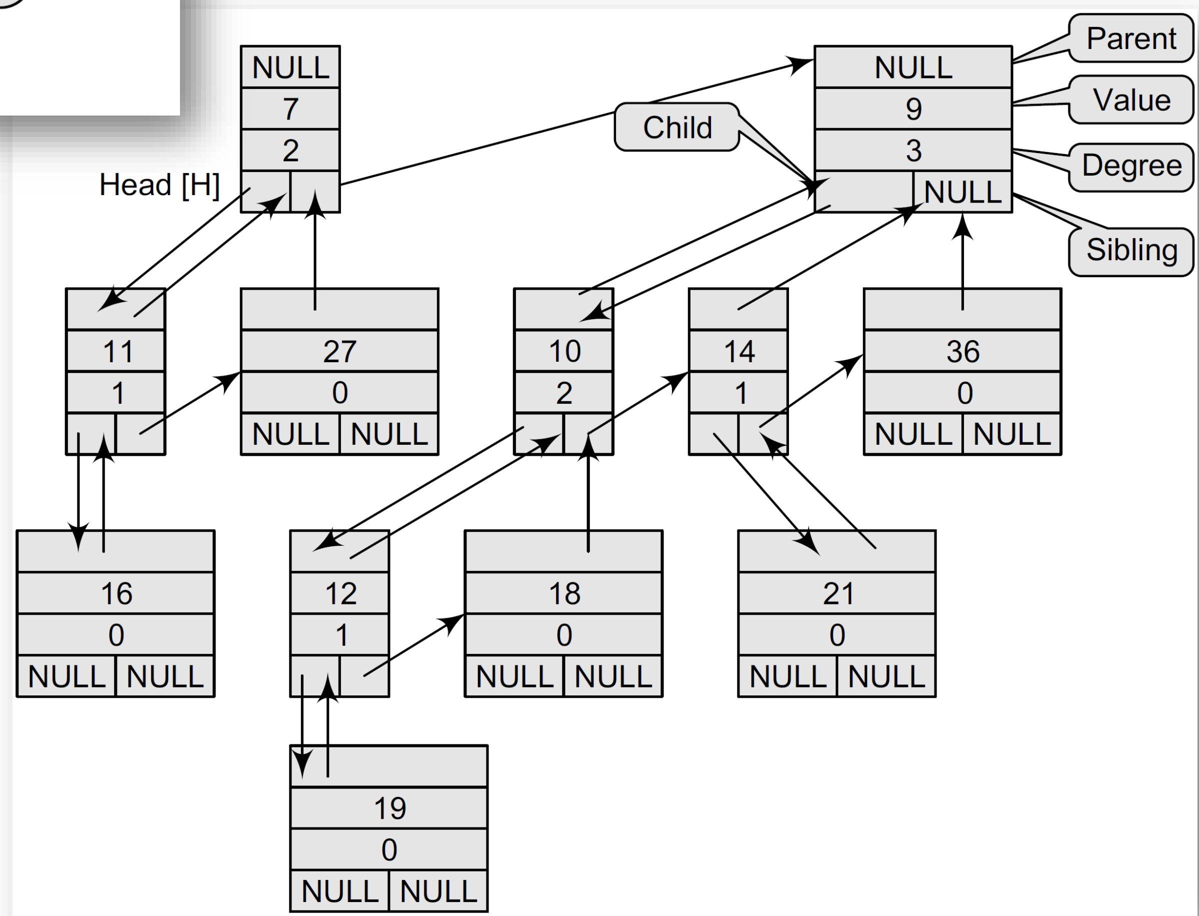
- A **binomial heap** H is a set of **binomial trees**
  - Every binomial tree in H satisfies the **minimum heap** property

B_2 B_3



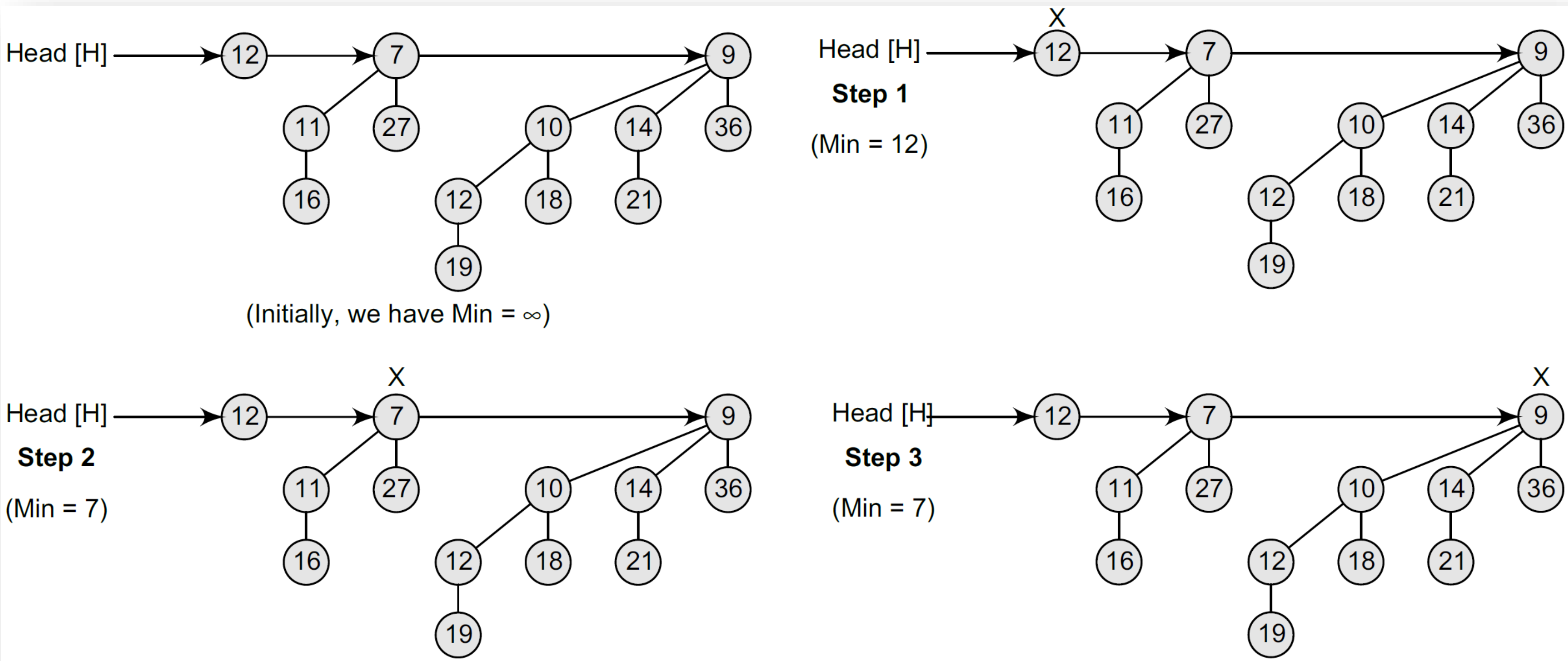
Binomial Heap with Linked List





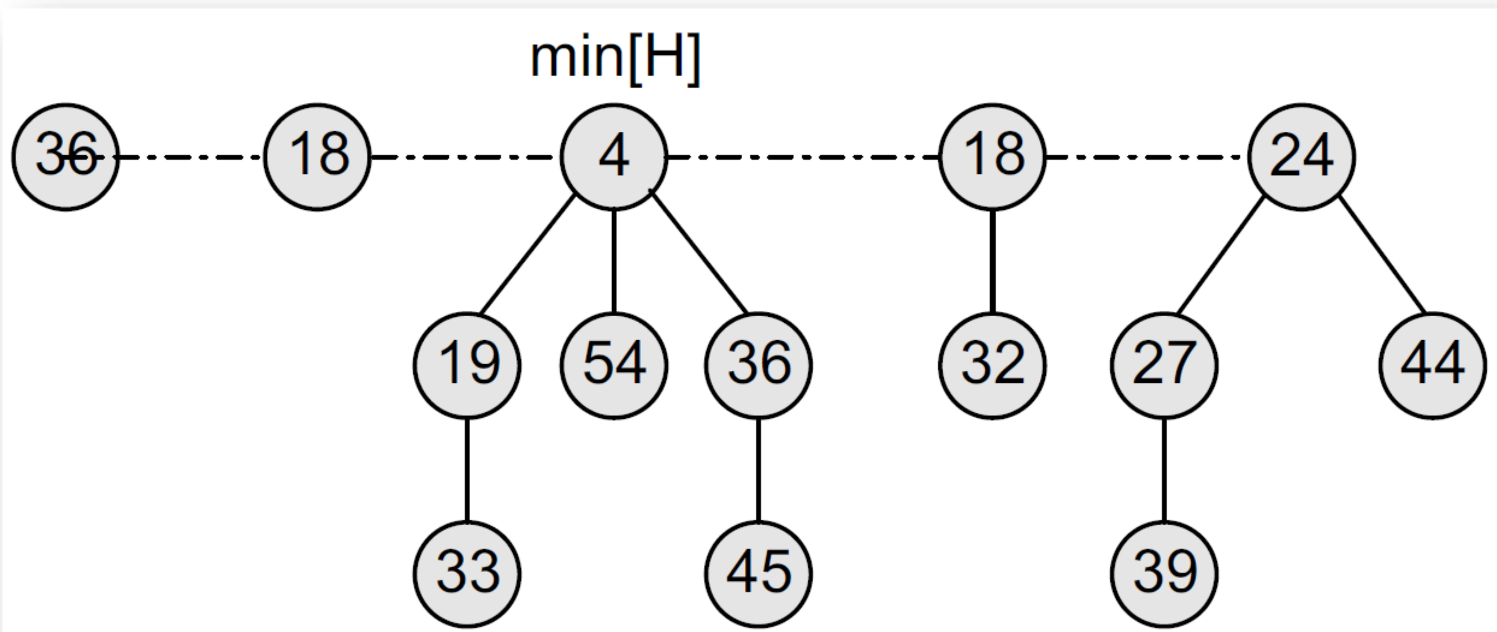
Minimum Value in Binomial Heap

- Since a binomial heap is heap-ordered, the node with the minimum value in a particular binomial tree will appear as a root node in the binomial heap



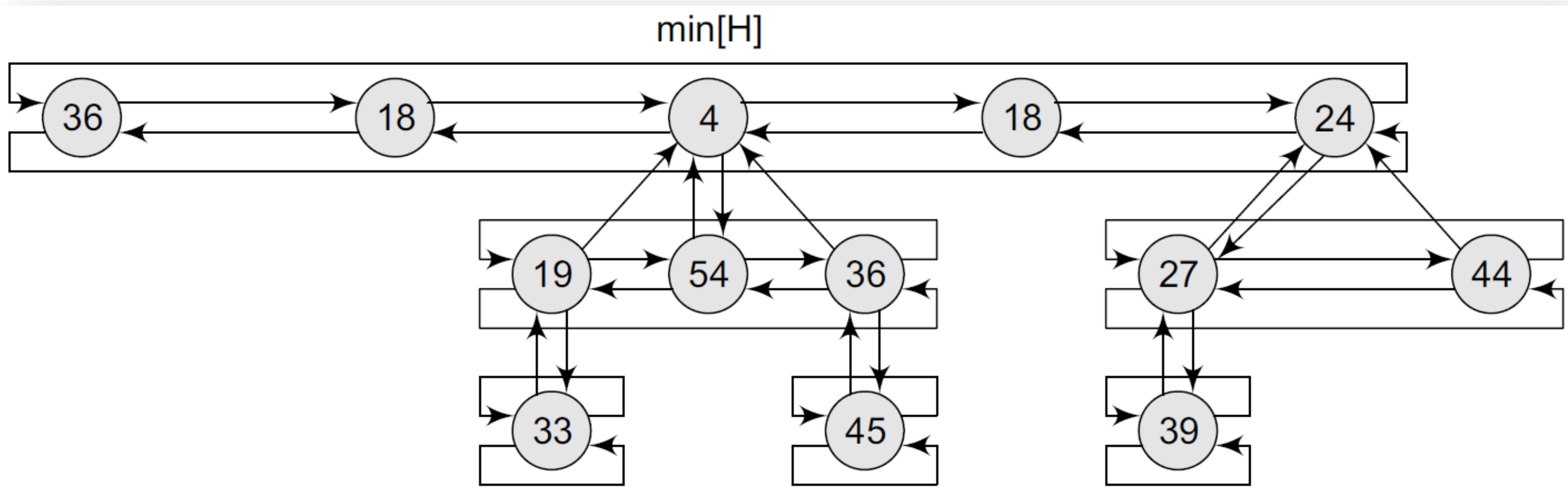
Fibonacci Heaps.

- A Fibonacci heap is a collection of trees
  - It is loosely based on binomial heaps
    - Fibonacci heaps differ from binomial heaps as they have a more relaxed structure
  - The trees in a Fibonacci heap are **not** constrained to be binomial trees



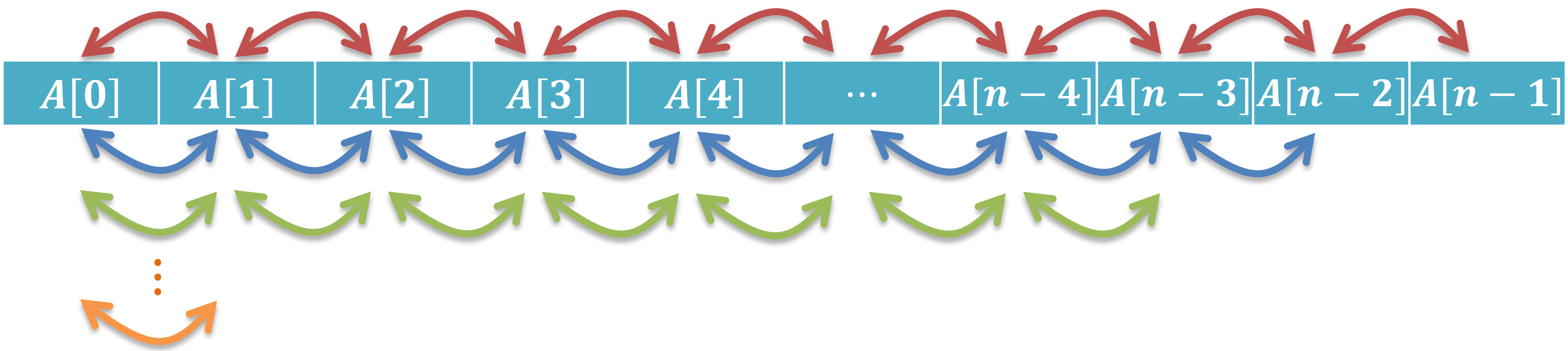
Fibonacci Heaps..

- Fibonacci heap H is generally accessed by a pointer called min$[H]$ which points to the root that has a minimum value
  - If the Fibonacci heap H is empty, then min$[H] = NULL$



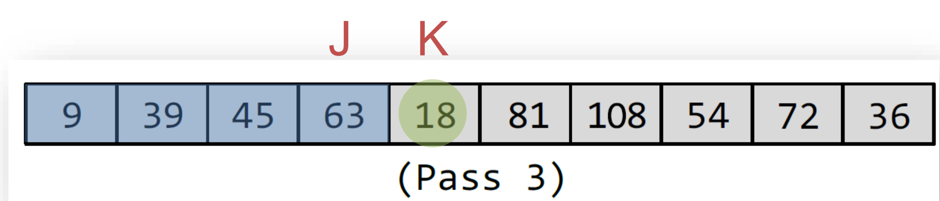
Review.

- Bubble Sort



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

- Insertion Sort



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

Review...

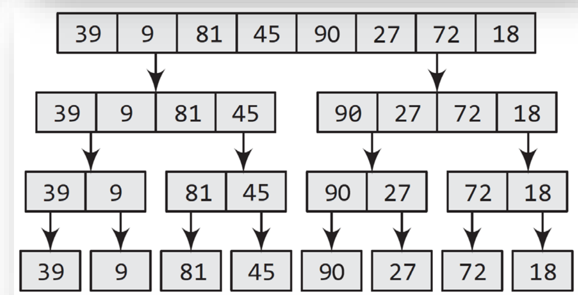
- Selection Sort
  - Average/Best/Worst Case: $\mathbf{O}(n^2)$

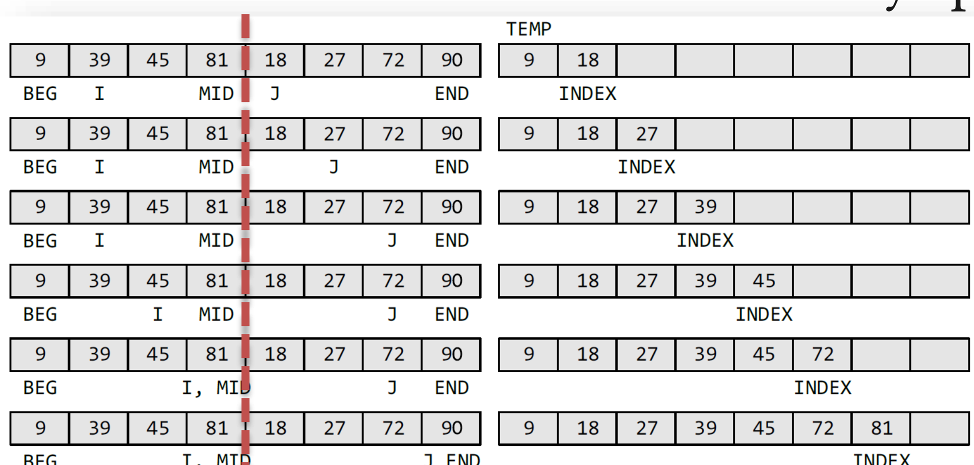
PASS	ARR[0]	ARR[1]	ARR[2]	ARR[3]	ARR[4]	ARR[5]	ARR[6]	ARR[7]
1	9	39	81	45	90	27	72	18
2	9	18	81	45	90	27	72	39
3	9	18	27	45	90	81	72	39
4	9	18	27	39	90	81	72	45
5	9	18	27	39	45	81	72	90
6	9	18	27	39	45	72	81	90
7	9	18	27	39	45	72	81	90

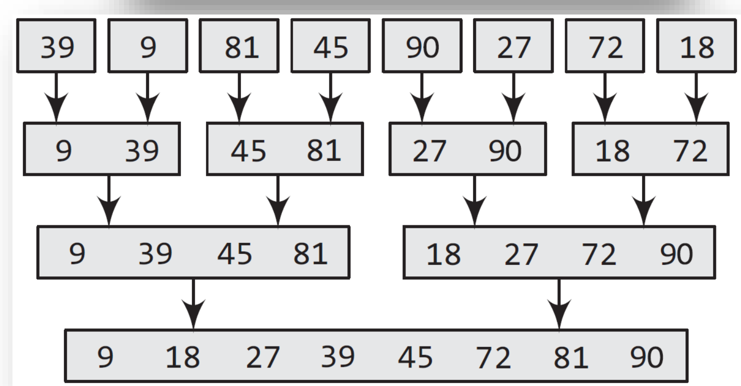
```
SMALLEST (ARR, K, N, POS)

Step 1: [INITIALIZE] SET SMALL = ARR[K]
Step 2: [INITIALIZE] SET POS = K
Step 3: Repeat for J = K+1 to N-1
            IF SMALL > ARR[J]
                  SET SMALL = ARR[J]
                  SET POS = J
            [END OF IF]
        [END OF LOOP]
Step 4: RETURN POS
```

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







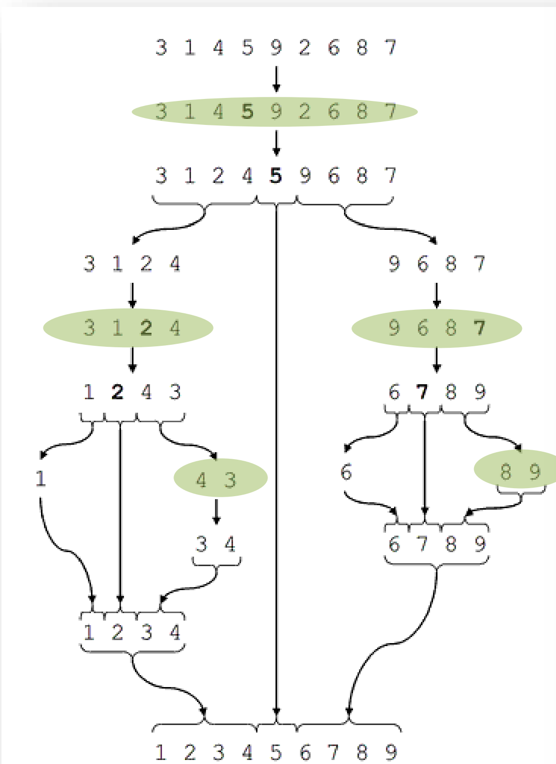
https://stackoverflow.com/questions/39665299/understanding-quicksort

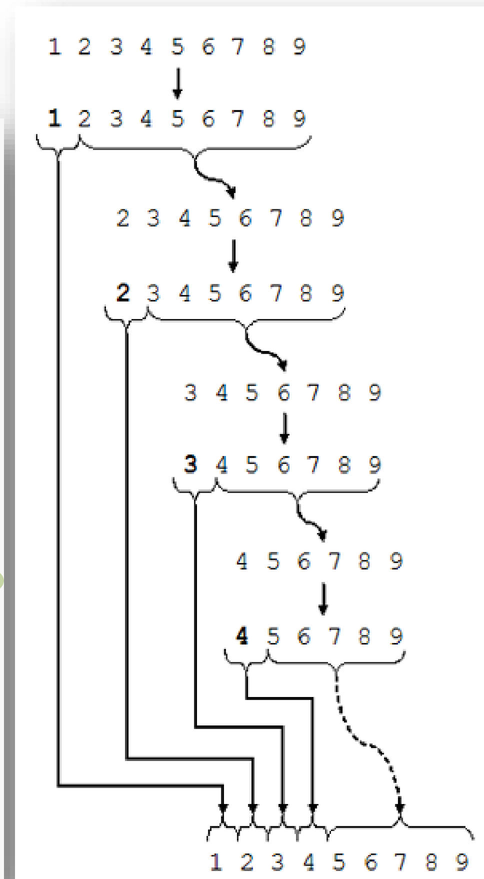
Review….

- Quick Sort

  - Best Case: $\mathbf{O}(n\log n)$
  - Average 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

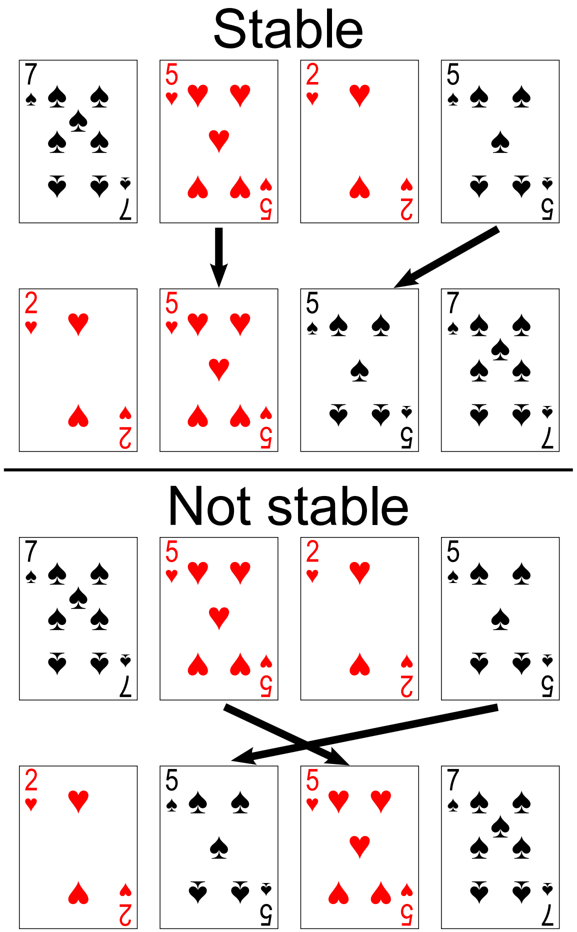
Number	0	1	2	3	4	5	6	7	8	9
911		911								
472								472		
123			123							
654						654				
924			924							
345					345					
555						555				
567							567			
808	808									

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

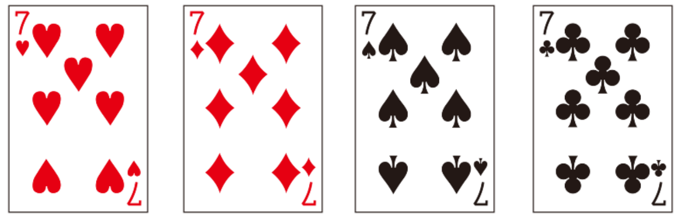
Stable Sorting Algorithms.

- Stable sort algorithms sort equal elements in the same order that they appear in the input
  - https://en.wikipedia.org/wiki/Sorting_algorithm#Stability

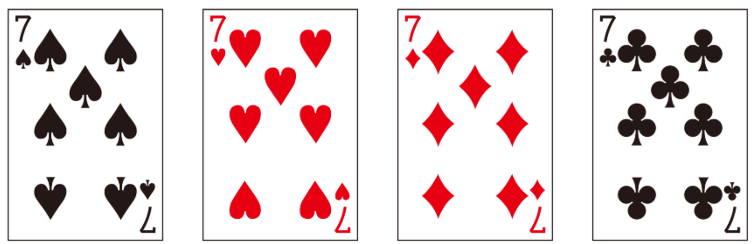


Stable Sorting Algorithms..

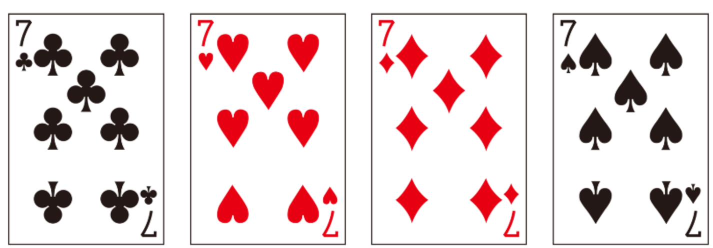
- Why is the property important?
  - Please sort the four cards by referring to their numbers and suits



    - First, sort by suits



    - Then, sort by numbers
      - For unstable sorting algorithm, we may get:



Comparison & Non-comparison

- Comparison sorts
  - A comparison sort cannot perform better than $\mathbf{O}(n\log n)$ on average
    - Bubble, Insertion, Tree, Selection, Merge, Shell, Quick, Heap sorts
- Non-comparison sort
  - Radix sort

Questions?

kychen@mail.ntust.edu.tw