Heap and Shell Sorts & Comparisons

Kuan-Yu Chen (陳冠宇)

2018/11/19 @ TR-212, 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, Selection, Tree**
  - **Merge, Quick, Radix**
  - Heap, Shell

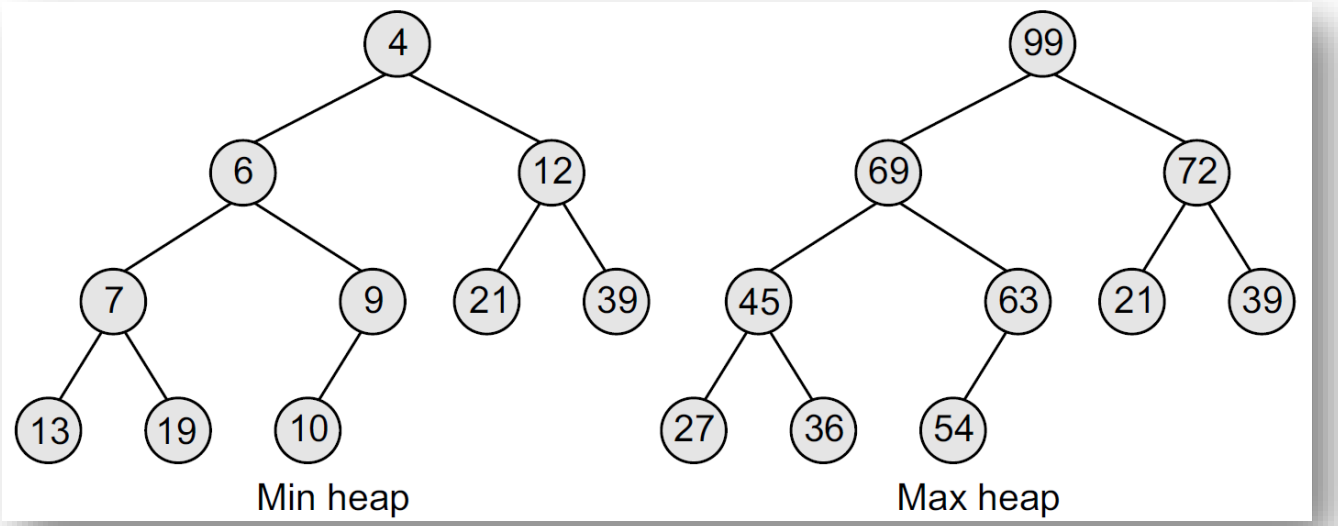
Heap

- 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

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

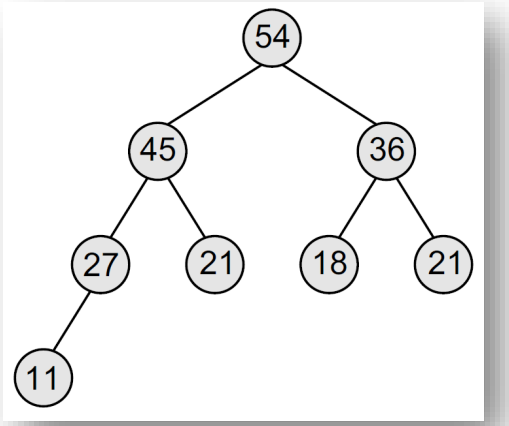


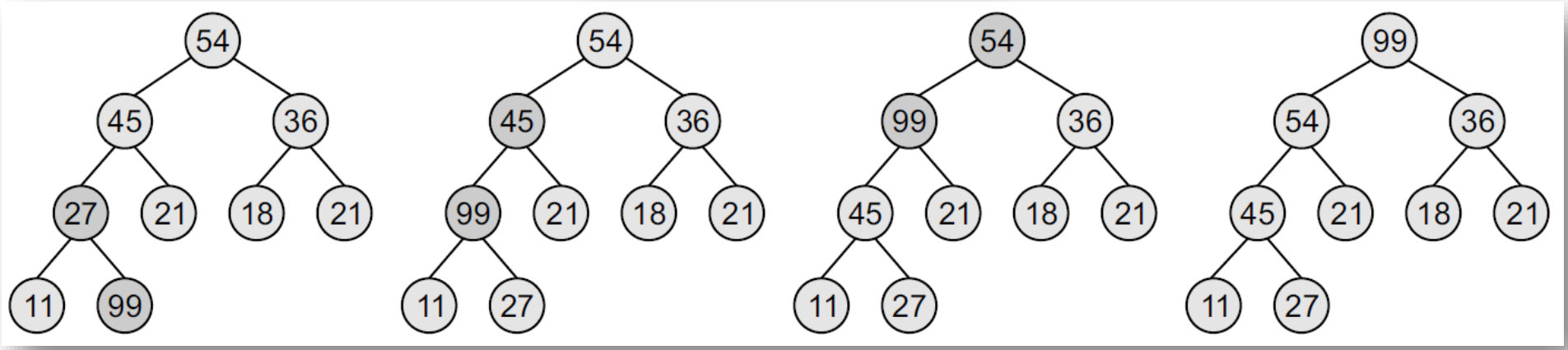
Heap – Insertion

- 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





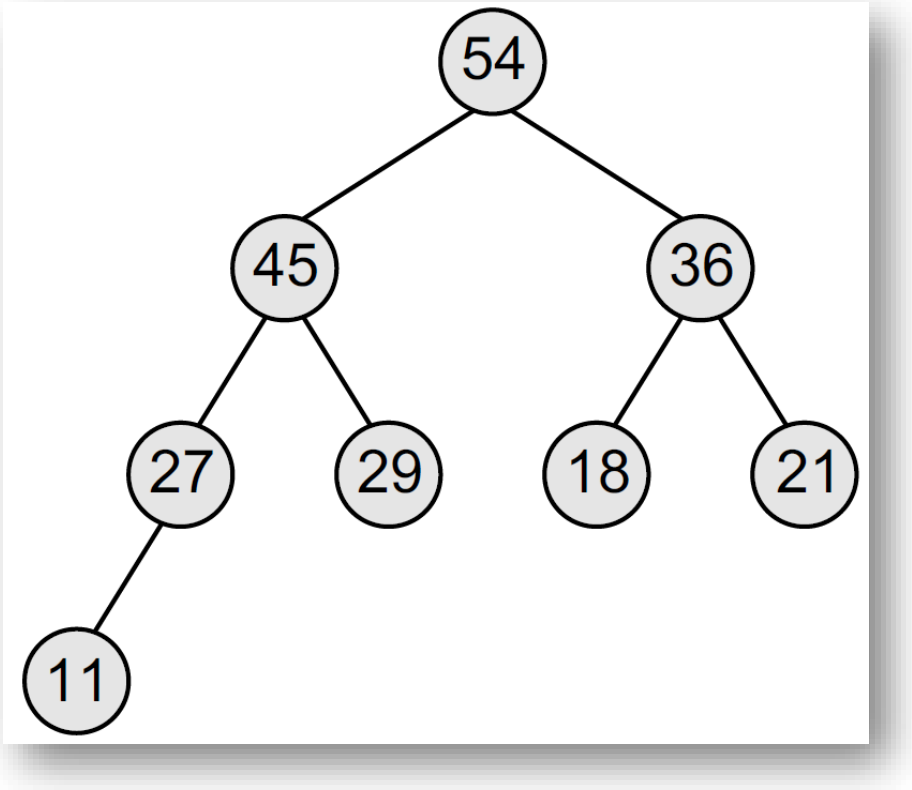
Heap – Deletion

- 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

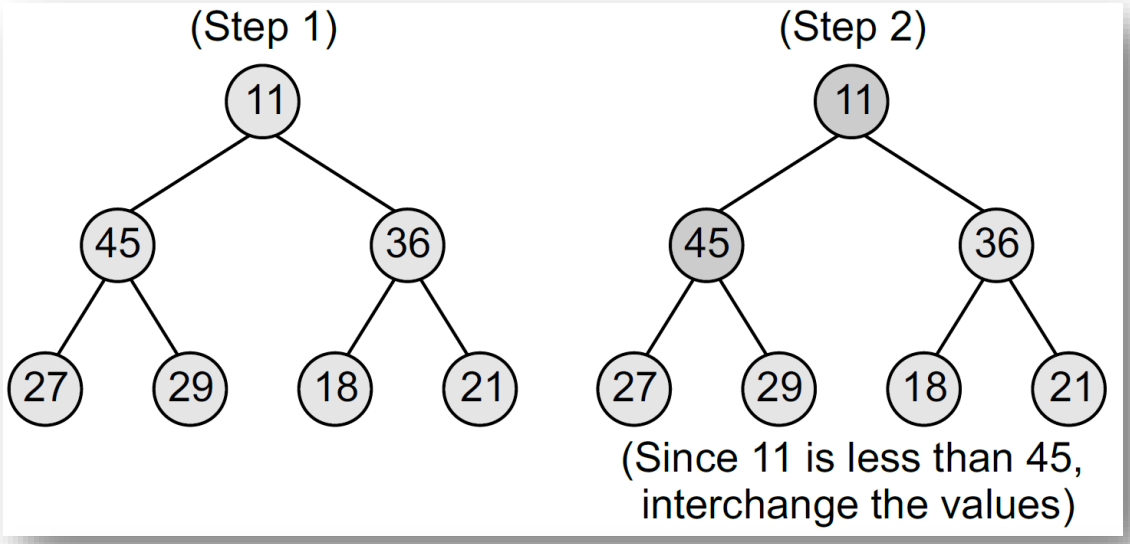
Example.

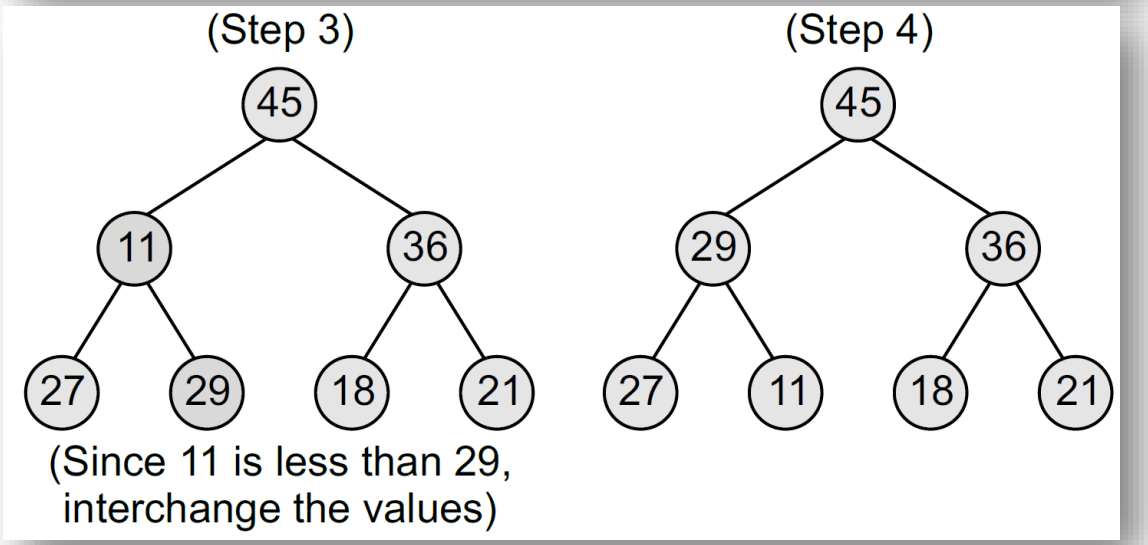
- Delete the root node's value from a given max heap *H*



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 = 0
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: $gap = \frac{9+1}{2}$

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: $gap = \frac{5+1}{2}$

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: $gap = 1$

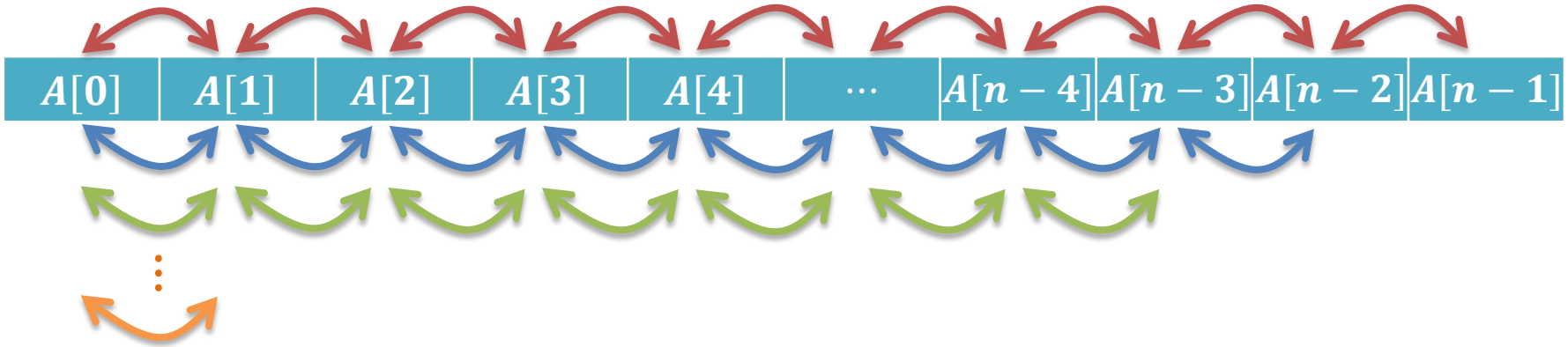
	Result:
9	7
19	9
7	19
22	22
27	27
36	33
33	36
45	41
54	45
41	54
63	59
59	63
81	72
72	81
90	90

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
```

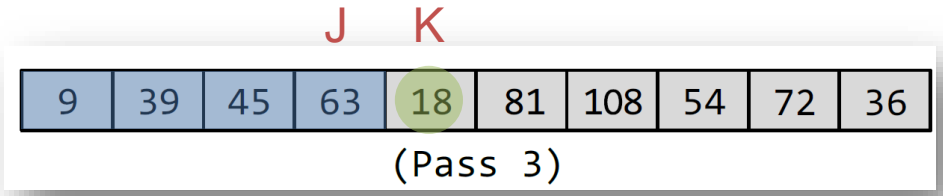
Comparisons.

- Bubble Sort



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

- Insertion Sort



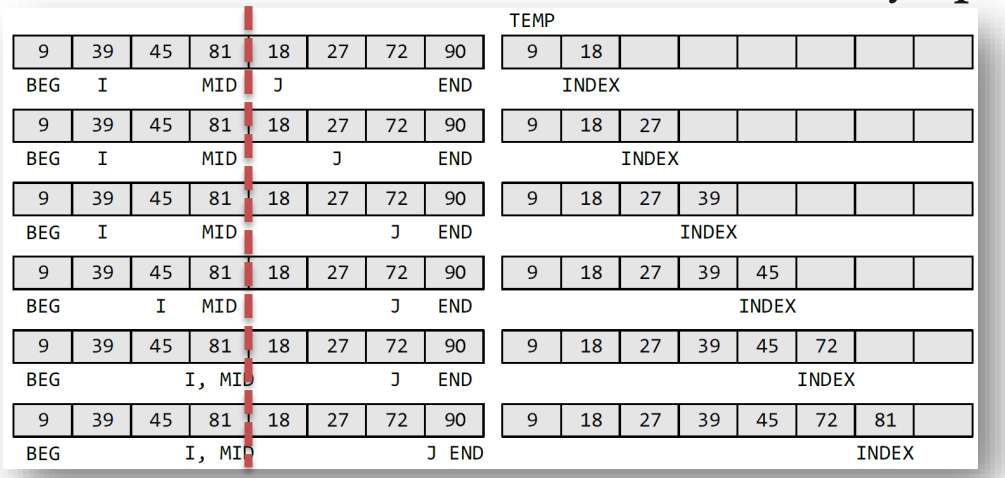
  - Best Case: $\mathbf{O}(n)$
  - Worst Case: $\mathbf{O}(n^2)$

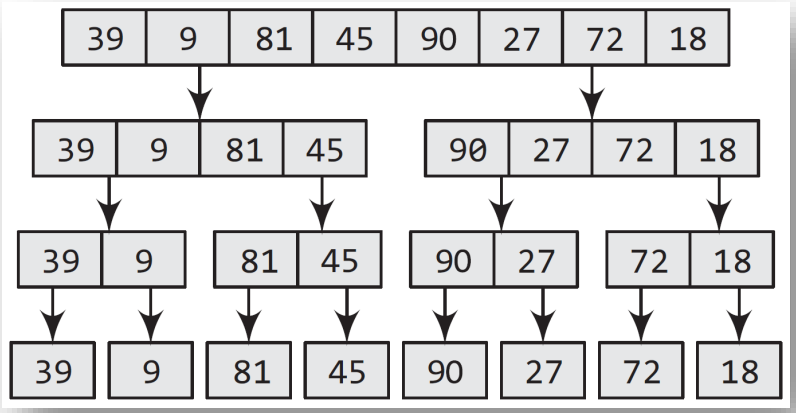
Comparisons..

- 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





Comparisons…

- Quick Sort

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

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

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									

Comparisons….

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

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

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

Questions?

kychen@mail.ntust.edu.tw