# CSC508 Data Structures

## Topic 10 : Sorting Algorithms

Compiled & edited by: Zahid Zainal

## Recap

- Binary Search Tree (BST)
- BST Operation
- AVL Tree

## Topic Structure

- Selection Sort
- Insertion Sort
- Merge Sort
- Heap Sort

## Learning Outcomes

- At the end of this lesson, students should be able to:
  - Describe and implement selection sort algorithm
  - Describe and implement insertion sort algorithm
  - Describe and implement merge sort algorithm
  - Describe and implement heap sort algorithm

## Sorting

- A procedure that rearranges the elements of a given list (e.g. array) in either ascending or descending orders.
- Sorted list may improve process like searching.
- Several approaches for sorting algorithm, e.g. brute-force and divide & conquer.
- Sorting algorithm : Bubble sort, Selection sort, Insertion sort, Merge sort, Heap sort, Quick sort

## Selection Sort

- The array is divided into two parts, a sorted and an unsorted part.
- Values from the unsorted part are picked and placed at the correct position in the sorted part
- Brute-force approach : $O(n^{2})$

## Selection Sort Algorithms

```
for i = 0 to length step 1
    min = i;
        for j = i+1 to length-1 step 1)
            if (arr[j] < arr[min])
                min = j
            endif
        endfor
    temp = arr[min]
    arr[i] = arr[min]
endfor
```

## Selection Sort Implementation – ascending

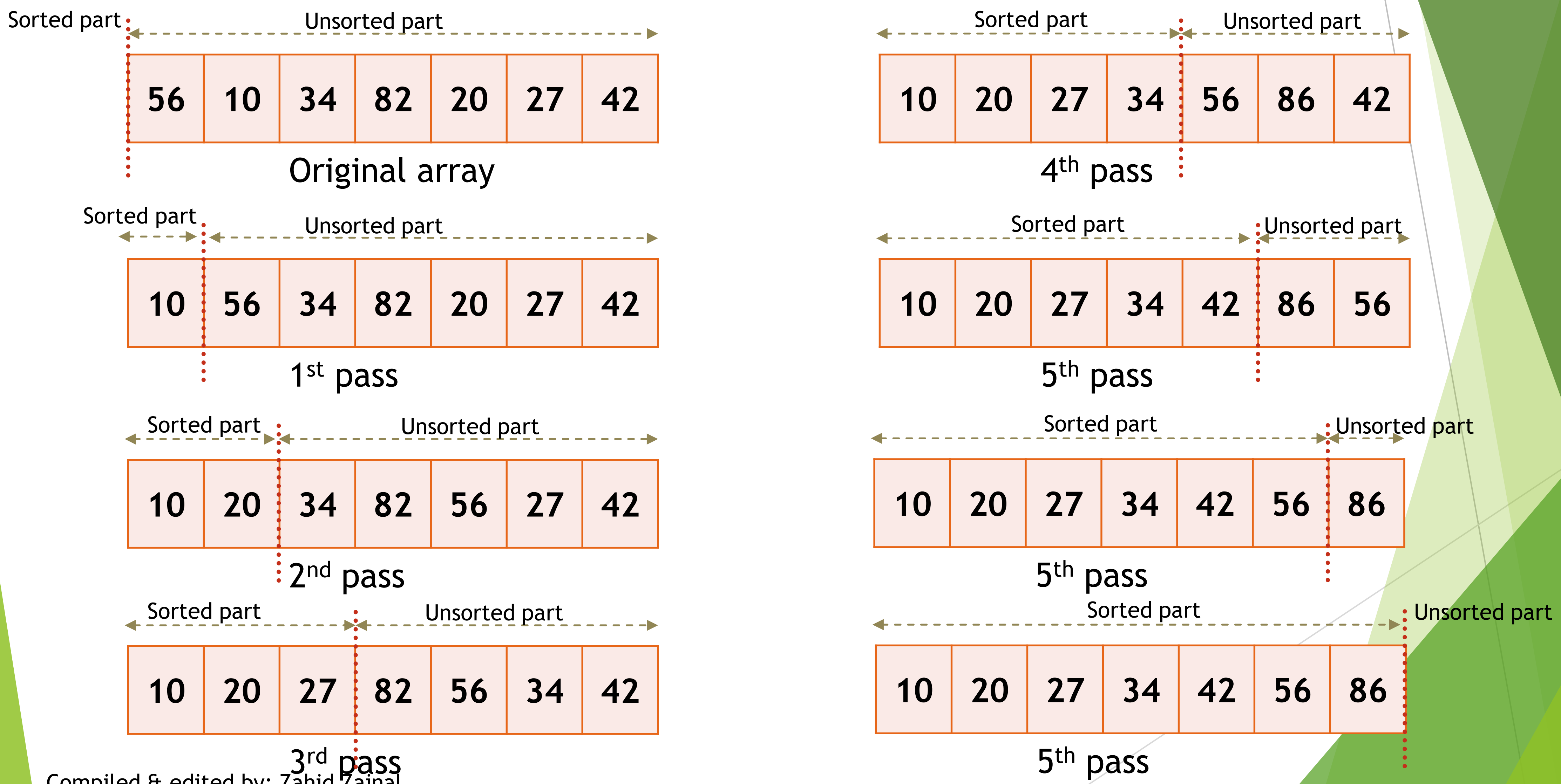
```
void sort(int arr[]){
   int n = arr.length;

   for (int i = 0; i < n-1; i++){
      int min_idx = i;
      for (int j = i+1; j < n; j++)
         if (arr[j] < arr[min_idx])
            min_idx = j;

      int temp = arr[min_idx];
      arr[min_idx] = arr[i];
      arr[i] = temp;
   }
}
```

Update the sorted part boundary

Search for the minimum value in unsorted part

Swap the minimum value into first element of unsorted part

# Selection Sort Simulation



| | | | | | | |
|---|---|---|---|---|---|---|
| 56 | 10 | 34 | 82 | 20 | 27 | 42 |

Original array

| | | | | | | |
|---|---|---|---|---|---|---|
| 10 | 56 | 34 | 82 | 20 | 27 | 42 |

1st pass

| | | | | | | |
|---|---|---|---|---|---|---|
| 10 | 20 | 34 | 82 | 56 | 27 | 42 |

2nd pass

| | | | | | | |
|---|---|---|---|---|---|---|
| 10 | 20 | 27 | 82 | 56 | 34 | 42 |

3rd pass

| | | | | | | |
|---|---|---|---|---|---|---|
| 10 | 20 | 27 | 34 | 56 | 86 | 42 |

4th pass

| | | | | | | |
|---|---|---|---|---|---|---|
| 10 | 20 | 27 | 34 | 42 | 86 | 56 |

5th pass

| | | | | | | |
|---|---|---|---|---|---|---|
| 10 | 20 | 27 | 34 | 42 | 56 | 86 |

5th pass

| | | | | | | |
|---|---|---|---|---|---|---|
| 10 | 20 | 27 | 34 | 42 | 56 | 86 |

5th pass

## Insertion Sort

- Sorts the list by moving each element to its proper place.
- General process
  - Divide the list into two segment, sorted and unsorted.
  - Take the first element in the unsorted segment and compared it with the elements in the sorted segment.
- Brute-force algorithm. Complexity O($n^2$).
  - May approach O(n) for ‘almost’ sorted list.

## Insertion Sort Algorithm

```
for i = 1 to length step 1 do
    key = arr[i]
    j = i - 1
    while (j >= 0 && arr[j] > key) do
        arr[j + 1] = arr[j]
        j = j - 1
    endwhile
    arr[j + 1] = key
endfor
```

UNIVERSITI TEKNOLOGI MARA

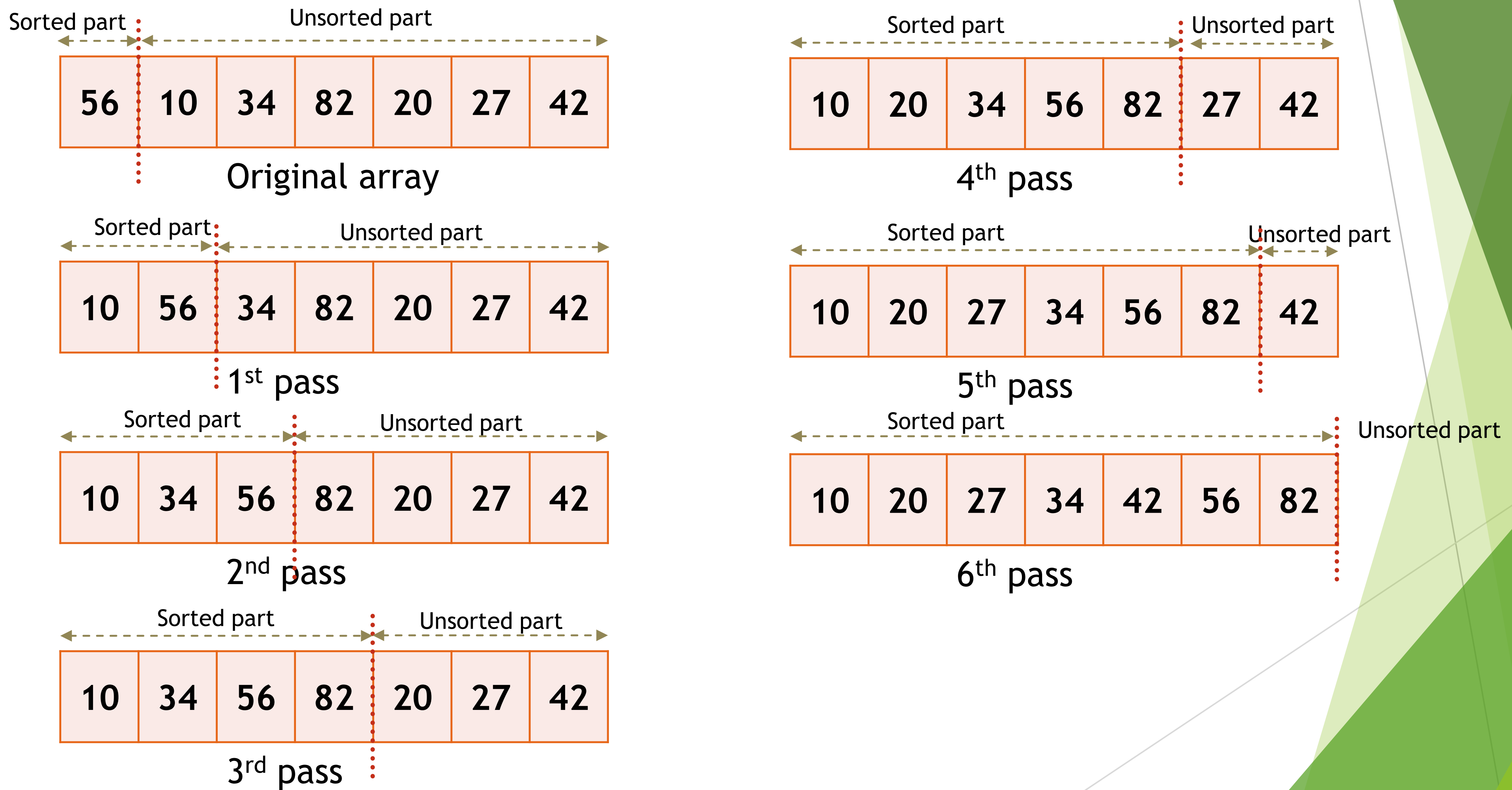
# Insertion Sort Implementation - ascending

```
void sort(int arr[]){
   int n = arr.length;
   for (int i = 1; i < n; ++i){
      int key = arr[i];
      int j = i - 1;

      while (j >= 0 && arr[j] > key) {
         arr[j + 1] = arr[j];
         j = j - 1;
      }
      arr[j + 1] = key;
   }
}
```

Move elements of arr[0..i-1], that are greater than key, to one position ahead of their current position

Compiled & edited by: Zahid Zainal

# Insertion Sort Simulation

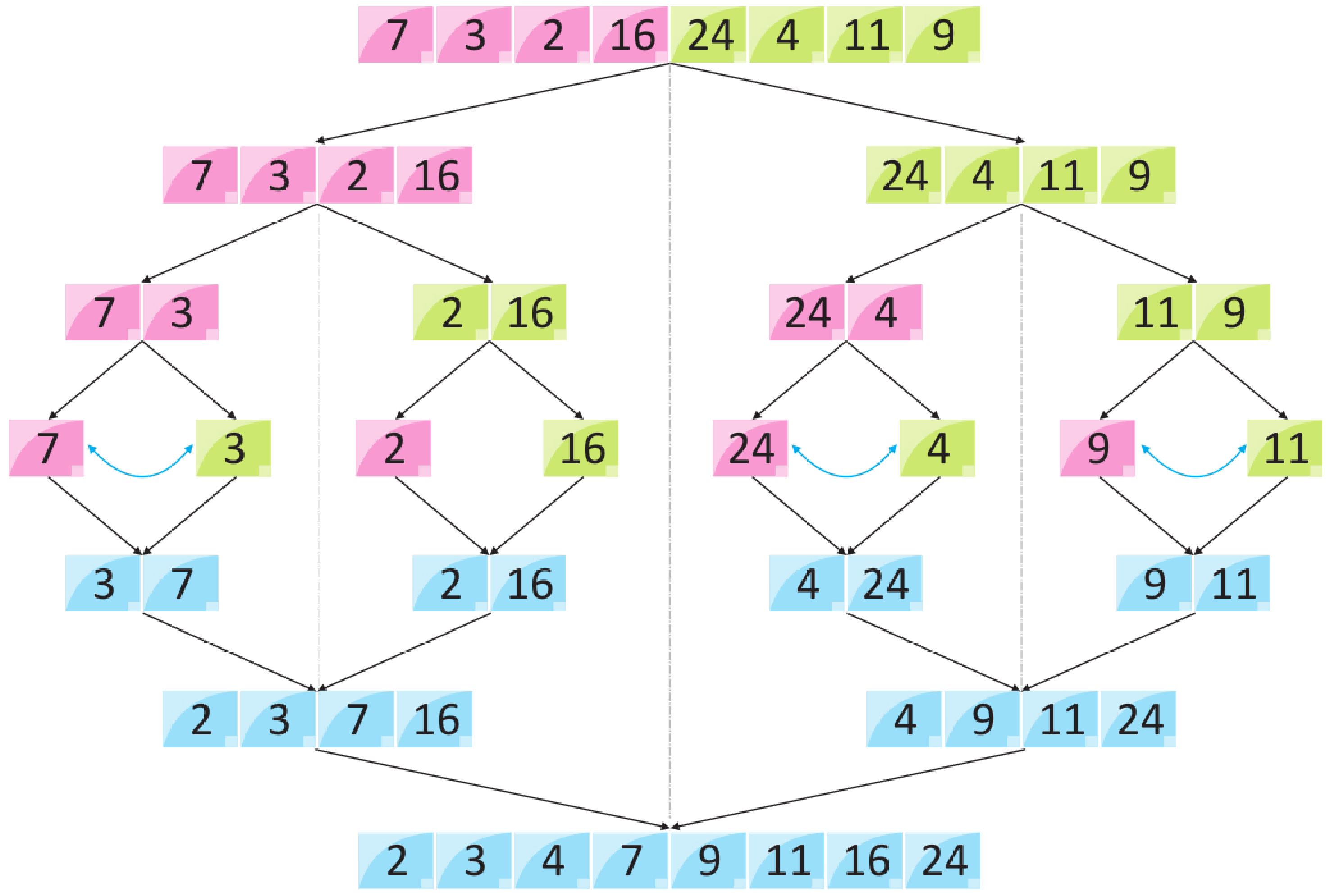


Compiled & edited by: Zahid Zainal

## Merge Sort

- Based on divide and conquer approach
- Recursively divide a list into two sub lists, sorts the sub lists, and then combines the sorted sub lists into one sorted list
- Time complexity : O(n log n)

# Merge Sort Example



https://linuxhint.com/merge-sort-python/

Compiled & edited by: Zahid Zainal

## Merge Sort Algorithm

```
step 1: start
step 2: declare array and left, right, mid variable
step 3: perform merge function.
    if left > right
        return
    mid= (left+right)/2
    mergesort(array, left, mid)
    mergesort(array, mid+1, right)
    merge(array, left, mid, right)
step 4: Stop
```

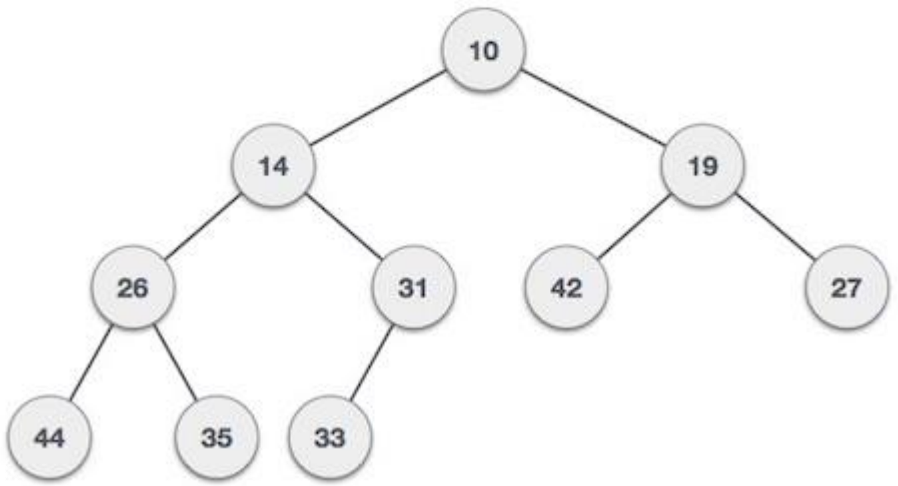
## Heap Sort

- Based on Binary Heap data structure.
- Similar to selection sort where we first find the maximum element and place the maximum element at the end.
- Time complexity : O(n log n)

## Binary Heap

- A heap is a list in which each element contains a key, such that the key in the element at position *k* in the list is at least as large as the key in the element at position 2*k* + 1 (if it exists), and 2*k* + 2 (if it exists).
  - If the parent node is stored at index i, the left child can be calculated by 2**i* + 1 and right child by 2**i* + 2 (assuming the indexing starts at 0).
- A heap is a Complete Binary Tree where items are stored in a special order such that:
  - value in a parent node is greater (max heap) than the values in its two children nodes.
  - value in a parent node is smaller (min heap) than the values in its two children nodes.

## Sample Min Heap



| 10 | 14 | 19 | 26 | 31 | 42 | 27 | 44 | 35 | 33 |
|---|---|---|---|---|---|---|---|---|---|
| [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] |

| | | 2i + 1 | 2i + 2 |
|---|---|---|---|
| Node value | Node 's Index | Left Child Index | Right Child Index |
| 10 | 0 | 2(0) + 1 = 1 | 2(0) + 2 = 2 |
| 14 | 1 | 2(1) + 1 = 3 | 2(1) + 2 = 4 |
| 19 | 2 | 2(2) + 1 = 5 | 2(2) + 2 = 6 |
| 26 | 3 | 2(3) + 1 = 7 | 2(3) + 2 = 8 |
| … | … | … | … |

# Heap Sort Algorithm

Sort Ascending

Step 1 - Build a max heap from the input data.

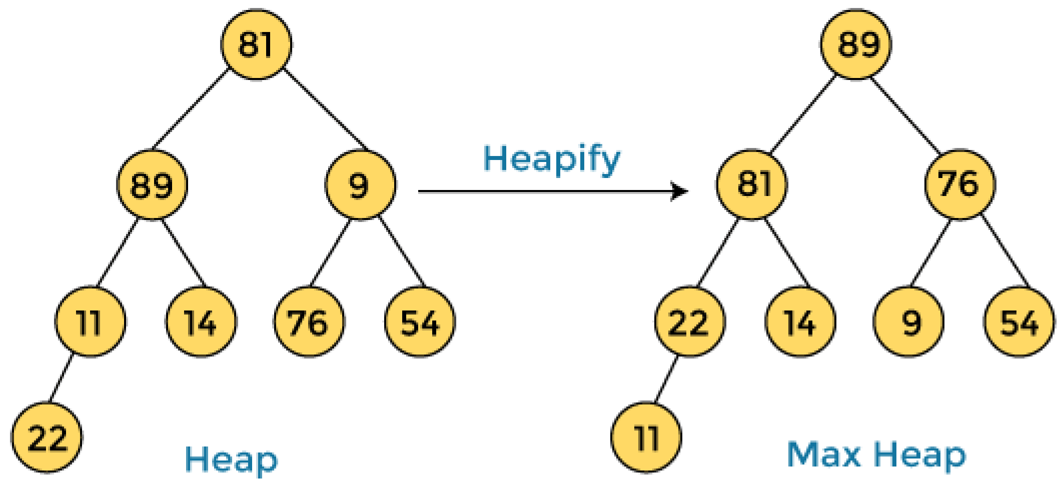
Step 2 - At this point, the maximum element is stored at the root of the heap. Replace it with the last item of the heap followed by reducing the size of the heap by 1. Finally, heapify the root of the tree.

Step 3 - Repeat step 2 while the size of the heap is greater than 1.

## Heap Sort Simulation - ascending

| 81 | 89 | 9 | 11 | 14 | 76 | 54 | 22 |
|---|---|---|---|---|---|---|---|

Original list



Heapify the original list. The largest element is the root

| 89 | 81 | 76 | 22 | 14 | 9 | 54 | 11 |
|---|---|---|---|---|---|---|---|

List after heapify.

| 11 | 81 | 76 | 22 | 14 | 9 | 54 | 89 |
|---|---|---|---|---|---|---|---|

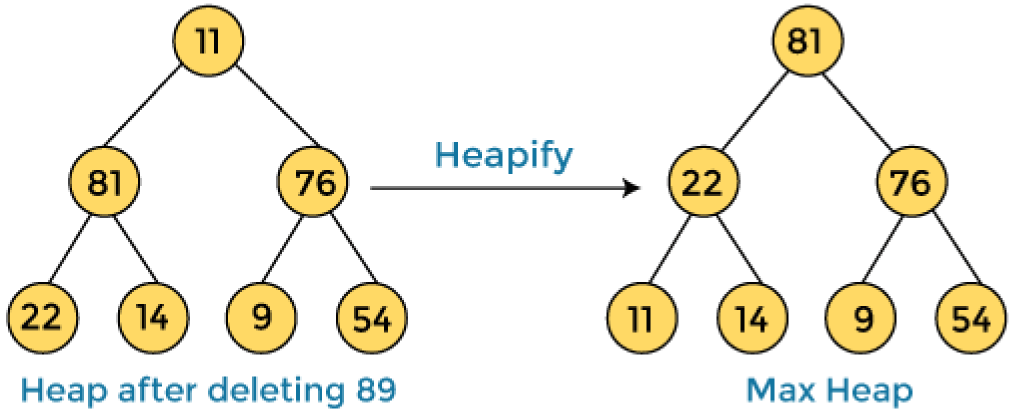
Swap the root with the last element.

https://www.javatpoint.com/heap-sort

Compiled & edited by: Zahid Zainal

# Heap Sort Simulation - ascending

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 11 | 81 | 76 | 22 | 14 | 9 | 54 | 89 |



Heapify the updated list. The largest element is the root

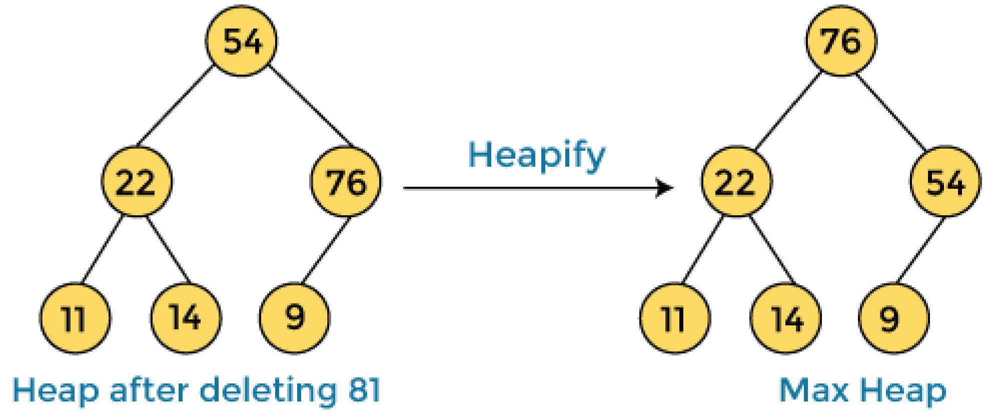
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 81 | 22 | 76 | 11 | 14 | 9 | 54 | 89 |

List after heapify.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 54 | 22 | 76 | 11 | 14 | 9 | 81 | 89 |

Swap the root with the last element.

|  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|
| 54 | 22 | 76 | 11 | 14 | 9 | 81 | 89 |



Heapify the updated list. The largest element is the root

|  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|
| 76 | 22 | 54 | 11 | 14 | 9 | 81 | 89 |

List after heapify.

|  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|
| 9 | 22 | 54 | 11 | 14 | 76 | 81 | 89 |

Swap the root with the last element.

The process continues until all elements are sorted.

## Summary

- A procedure that rearranges the elements of a given list
- Several approaches for sorting algorithm, e.g. brute-force and divide & conquer.
- Complexity : Insertion sort $O(n^2)$, Selection sort $O(n^2)$, Merge sort $O(n \log n)$, Heap sort $(n \log n)$

## Next Topic…

- Searching.

## References

- Carrano, F. & Savitch, W. 2005. *Data Structures and Abstractions with Java, 2nd ed. Prentice-Hall.*
- Malik D.S, & Nair P.S., Data Structures Using Java, Thomson Course Technology, 2003.
- Rada Mihalcea, CSCE 3110 Data Structures and Algorithm Analysis notes, U of North Texas.