430.211 Programming Methodology (프로그래밍 방법론)

Sorting Algorithm part 1

Lab2 #02

Fall 2025

Machine Intelligence & Data science LAB

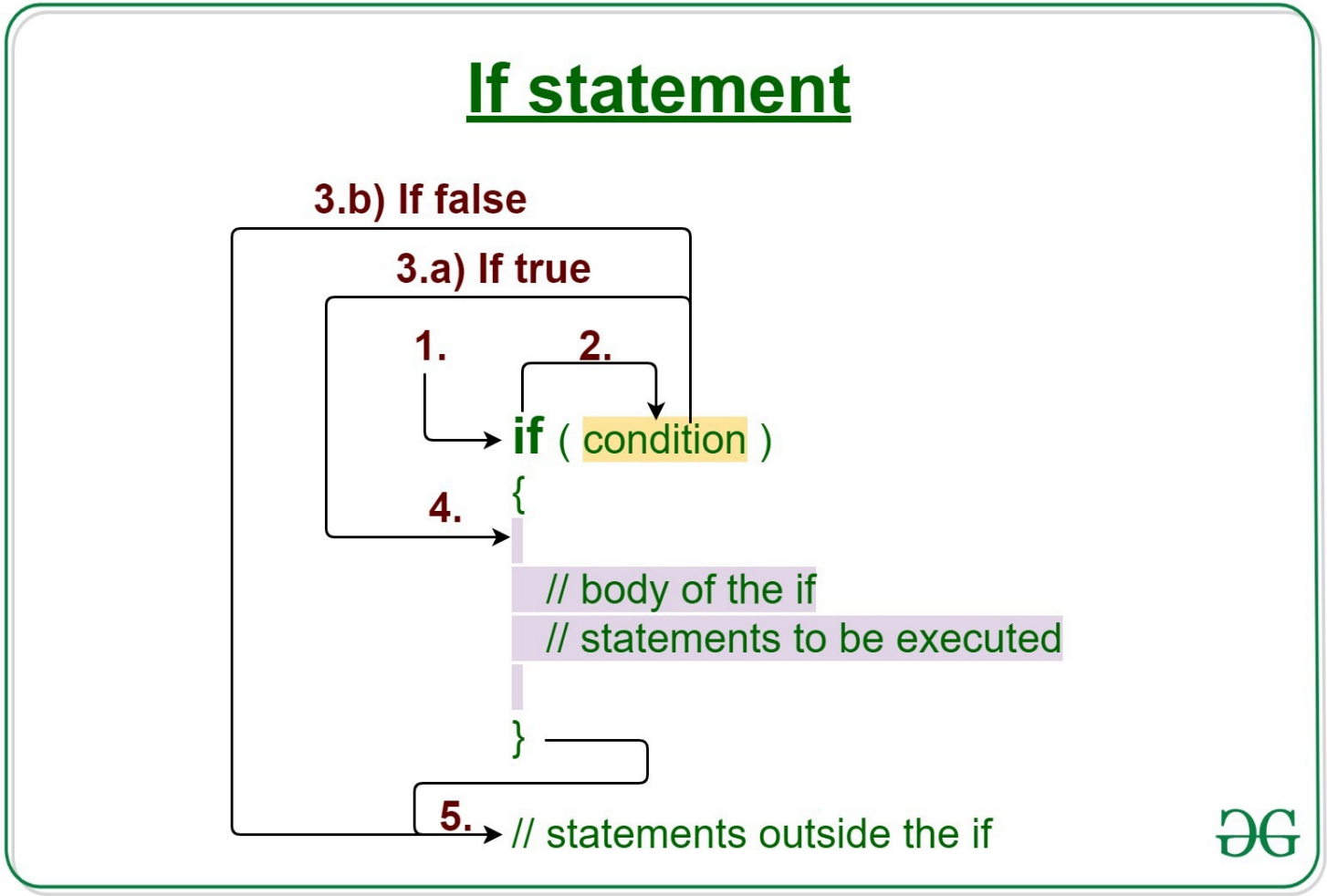
Attendance check

- Attendance will be taken after the class ends
  - With Quiz system in ELICE platform

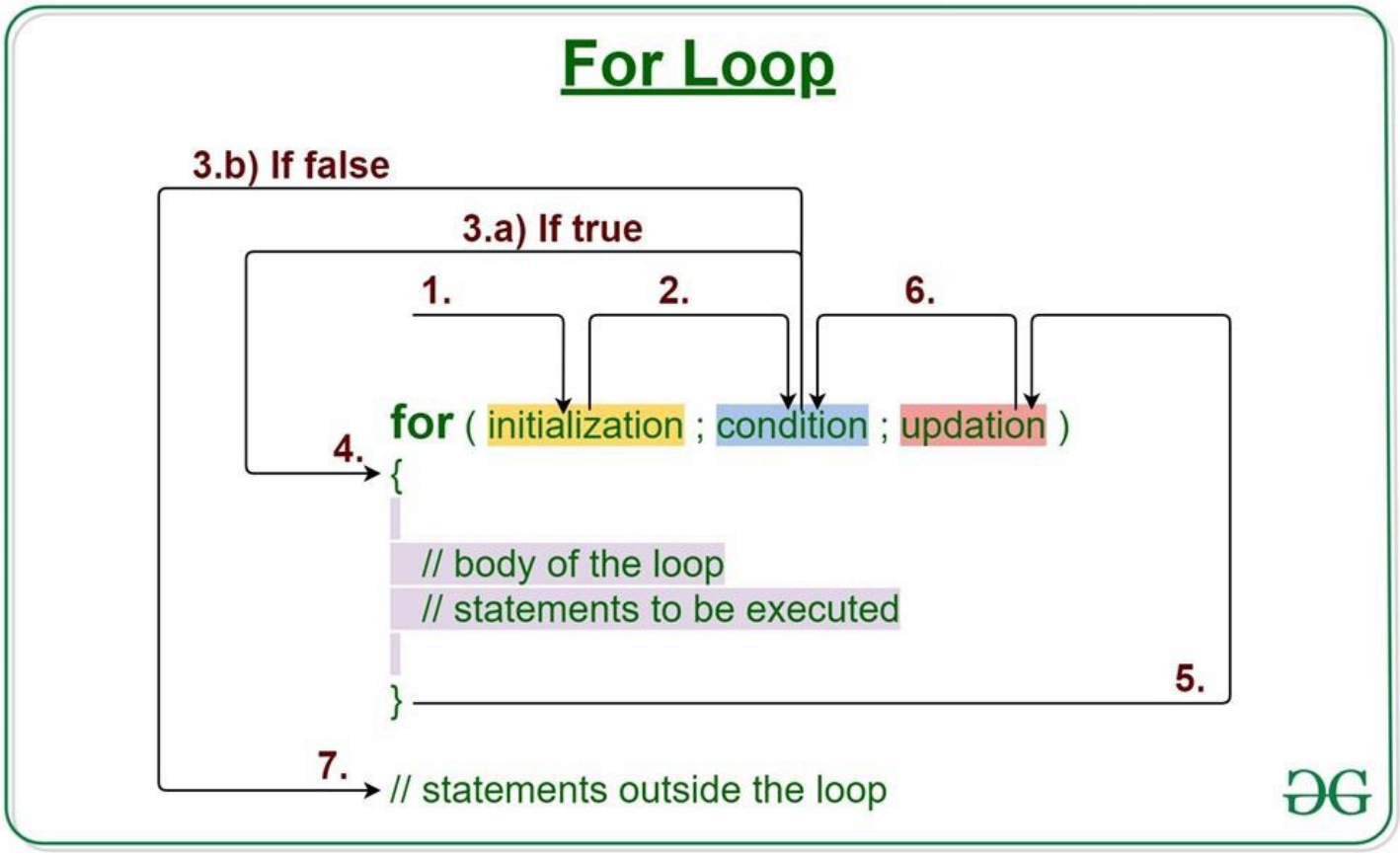
Outline

- **If / For loop / While loop**
  - **Since we are ahead of the Wednesday class schedule**
- Array
- What is sorting?
- Incremental Algorithm
  - Bubble sort
  - Insertion sort
  - Selection sort
  - Heap sort

If


If statement
3.b) If false
3.a) If true
1.
2.
if (condition)
{
4.
// body of the if
// statements to be executed
}
5.
// statements outside the if

For loop



- **Initialization** is executed (one time) before the execution of the code block.
- **Conditon** defines the condition for executing the code block.
- **Updation** is executed (every time) after the code block has been executed.

For loop

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {

    // for loop to print "Hi" 5 times
    for (int i = 5; i < 10; i++) {
        cout << "Hi" << endl;
    }

    return 0;
}
```

Output

```
Hi
Hi
Hi
Hi
Hi
```

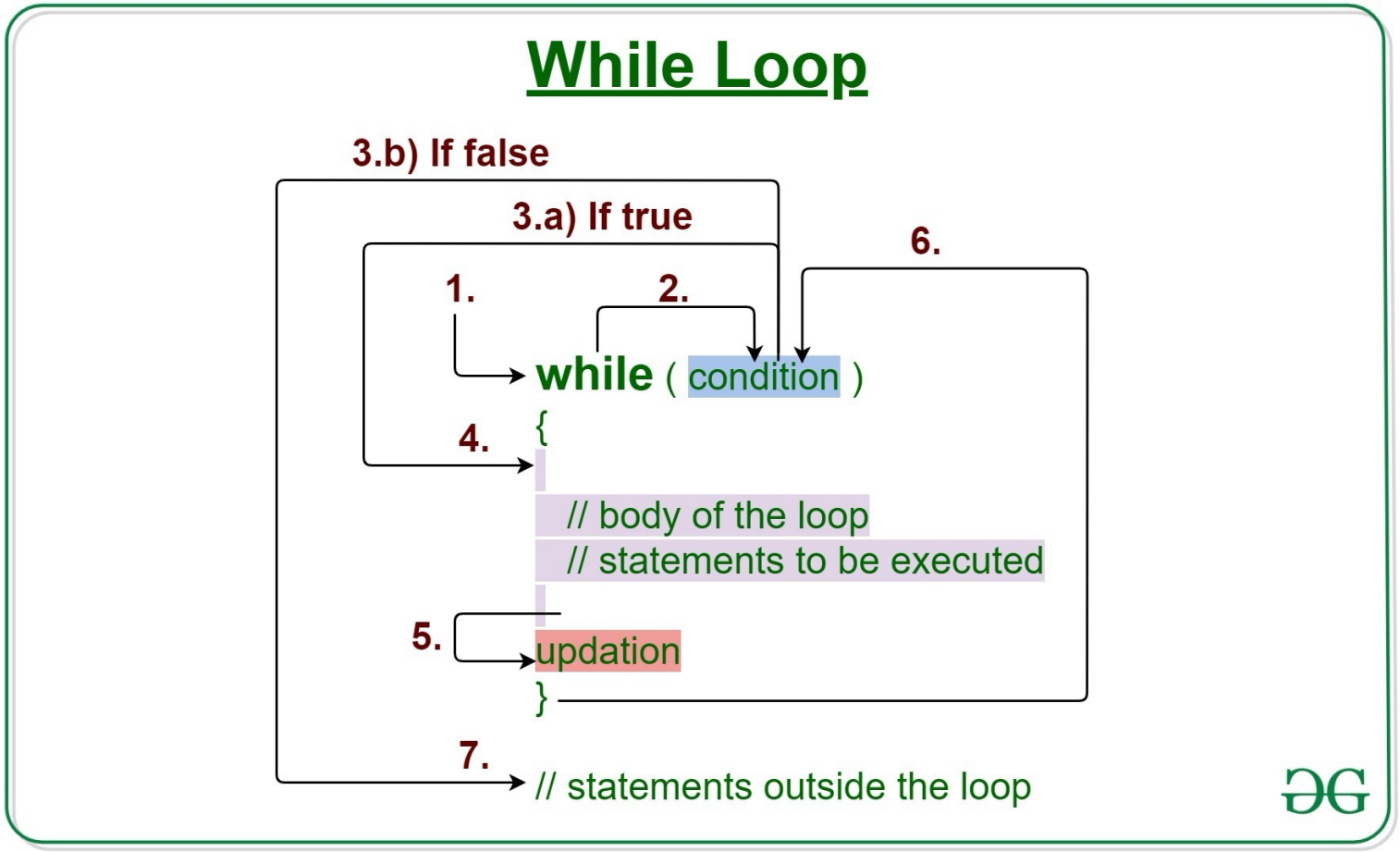
For loop

```cpp
#include <iostream>
using namespace std;

int main() {

    // Initial value of number
    int n = 5;

    // Initialization of loop variable
    int i;
    for (i = n; i >= 1; i--)
        cout << i << " ";
    return 0;
}
```

Output

```
5 4 3 2 1
```

While loop


While Loop
3.b) If false
3.a) If true
6.
1.
2.
while (condition)
4.
{
// body of the loop
// statements to be executed
5.
updation
}
7.
// statements outside the loop

While loop

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {

    // Declaration and Initialization of loop variable
    int i = 1;

    // while loop to print numbers from 1 to 5
    while (i <= 5) {
        cout << i << " ";

        // Updating loop varialbe
        i++;
    }

    return 0;
}
```

Output

```
1 2 3 4 5
```

Break statement

```
for (init; condition; update) {
    // code
    if (condition to break) {
        break;
    }
    // code
}
```

```
while (condition) {
    // code
    if (condition to break) {
        break;
    }
    // code
}
```

Machine Intelligence & Data science LAB

Outline

- If / For loop / While loop
- **Array**
- What is sorting?
- Incremental Algorithm
  - Bubble sort
  - Insertion sort
  - Selection sort
  - Heap sort

Array

- Array definition:
  - A collection of data of a same type
  - `int`,`float`,`double`,`char` are simple data types

- Array indexes always start with zero!
  - To computer scientists, "zero" is the "first" number

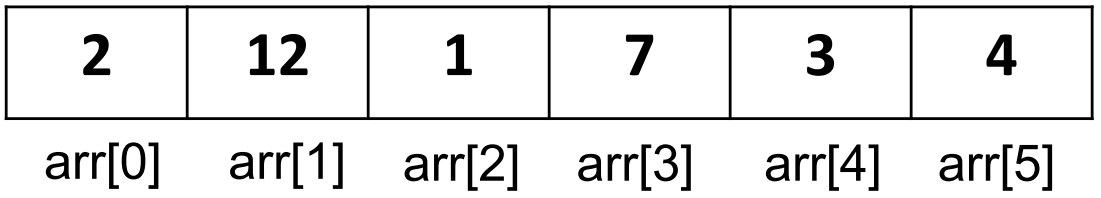
Array

- Initialization:

```
int arr[6] = {2, 12, 1, 7, 3, 4}; // declaration and initialization
```

  - Equivalent to the following:

```
int arr[6]; // declaration
arr[0] = 2; // assignment
arr[1] = 12;
arr[2] = 1;
arr[3] = 7;
arr[4] = 3;
arr[5] = 4;
```



  - Value in brackets called index or subscript
    - Numbered from **0 to (size – 1)**
  - Elements are stored sequentially in a contiguous block of memory.

Outline

- For loop and while loop
- Array
- **What is sorting?**
- Incremental Algorithm
  - Bubble sort
  - Insertion sort
  - Selection sort
  - Heap sort

What is Sorting?

- Arranging items in ascending or descending order

Unsorted array

Sorted array in ascending order.

Sorted array in descending order.

https://medium.com/@imasharasangee/exploring-the-power-of-sorting-algorithms-23667e22f935

What is Sorting?

- There are various sorting algorithms
  - Bubble sorting
  - Insertion sorting
  - Selection sorting
  - Heap sorting
  - Merge sorting
  - Quick sorting
  - Counting sorting..., and so on

Today lecture

What is Sorting?

- Which sorting algorithm is appropriate ?
  - Time complexity / Space complexity (Big O notation)
  - But, we will mainly focus on implementation in this course.

Formal definition of Big-Oh:

$f(N) = O(g(N))$, if there exists positive constants c , N_0 such that

$f(N) \leq c \cdot g(N)$ for all $N \geq N_0$.

If $c = 3, N_0 = 10$

```
Regular       Big-O
2             O(1)     --> It's just a constant number
2n + 10       O(n)     --> n has the largest effect
5n^2          O(n^2)   --> n^2 has the largest effect
```

f(N) g(N)

What is Sorting?

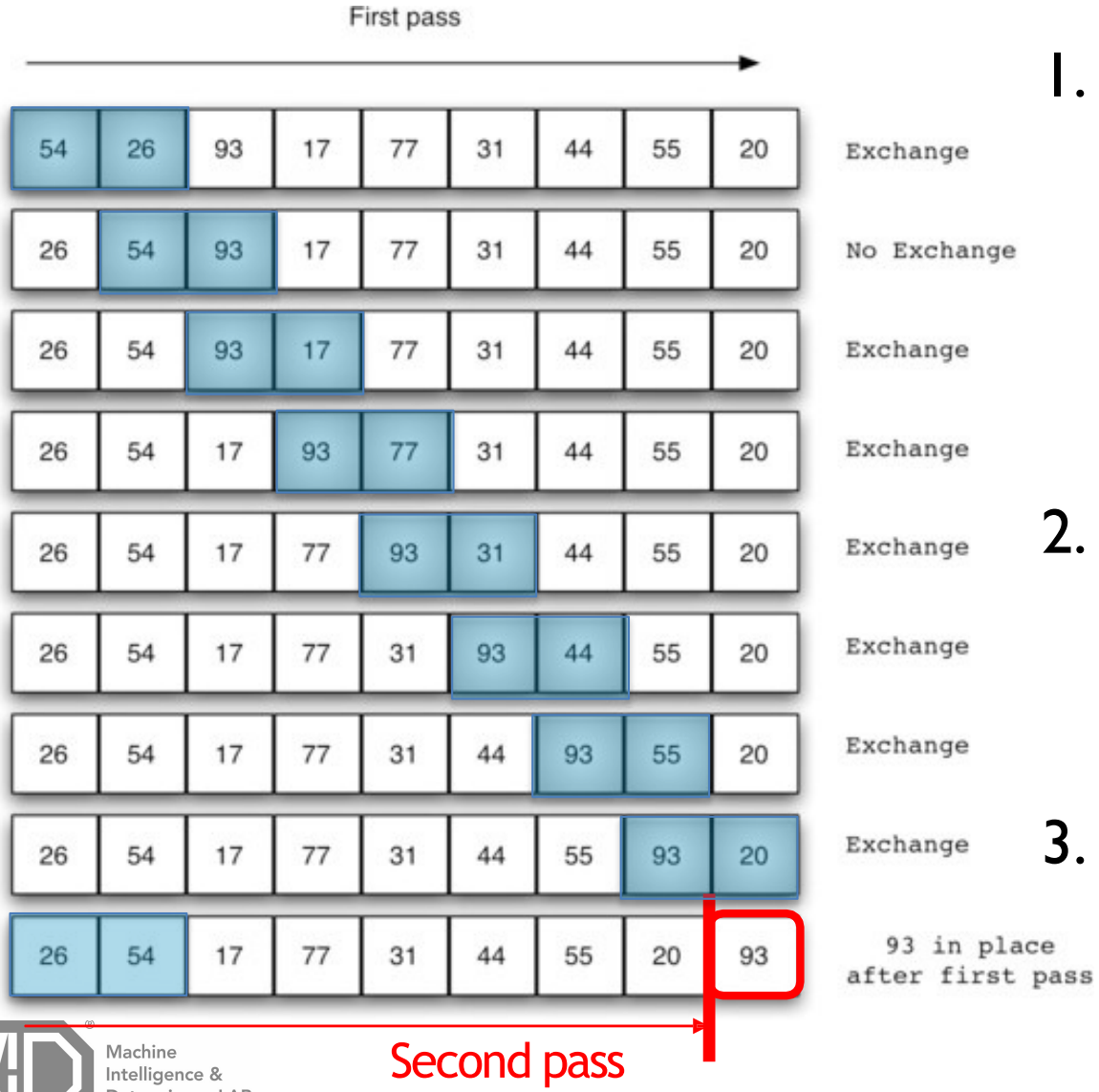
- Which sorting algorithm is appropriate ?
  - Time complexity / Space (memory) complexity (Big O notation)
  - Let # of items to sort = n
  - Count the number of operations in the program logic
    - Add / divide / multiply two items
    - Compare two items
    - …

(ex) if algorithm A requires a total of $2n^2 + 4n$ operations, then algorithm A has a time complexity of $O(n^2)$

Outline

- If / For loop / While loop
- Array
- What is sorting?
- **Incremental Algorithm**
  - **Bubble sort**
  - Insertion sort
  - Selection sort
  - Heap sort

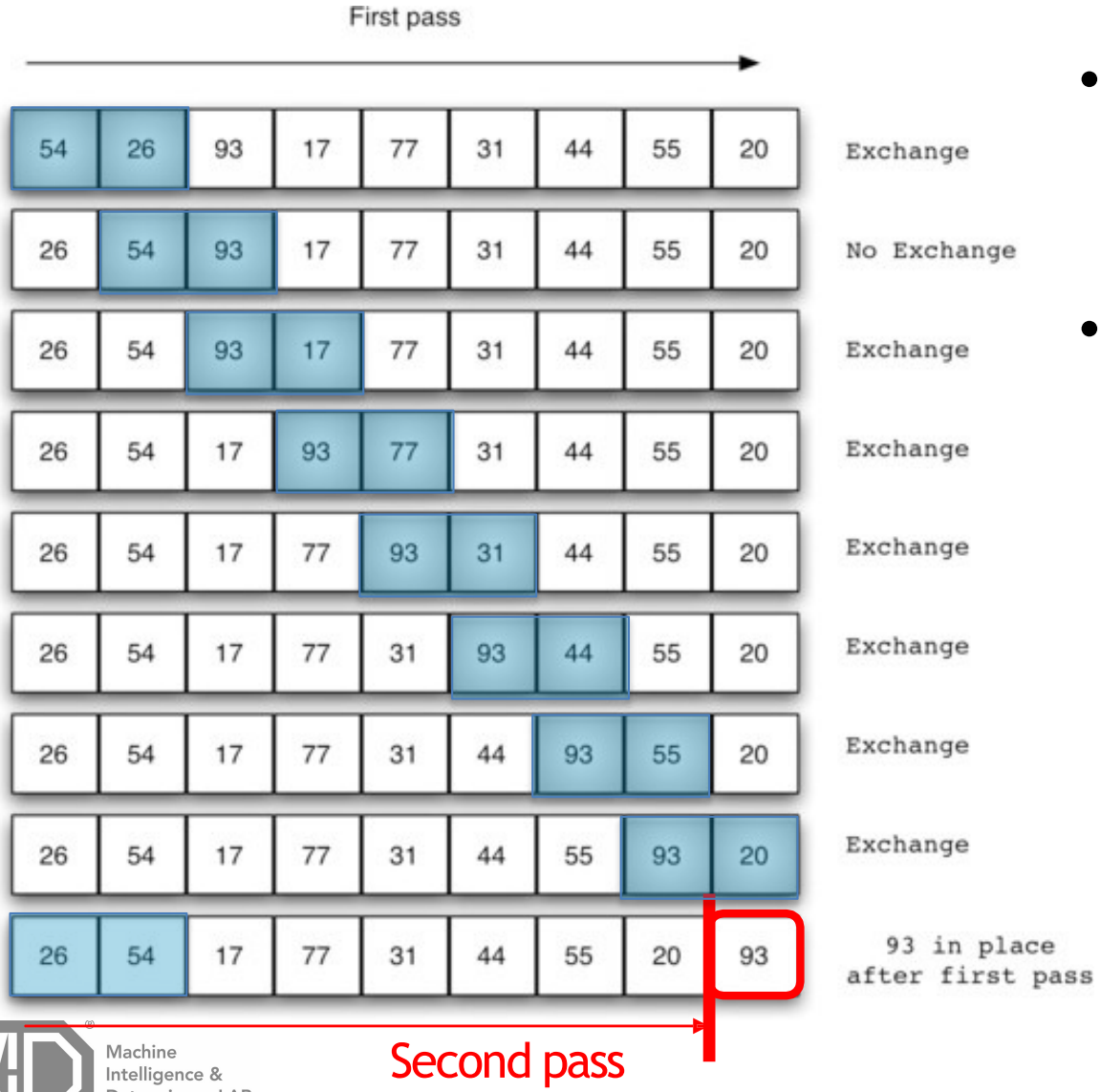
Bubble Sort



1. First pass
   - Compare two elements
   - Swap in the desirable order until the last element
   - The largest value has bubbled up to the rightmost position
2. Second Pass
   - Compare two elements
   - Swap in the right order until the second-to-last element
3. Continue…

https://interactivepython.org/runestone/static/pythonds/SortSearch/TheBubbleSort.html

Bubble Sort



- Pros :
  - Easy to implement
- Cons :
  - As the length of input increases, time increases exponentially.
  - Time complexity is $O(n^2)$

https://interactivepython.org/runestone/static/pythonds/SortSearch/TheBubbleSort.html

Bubble Sort

```
for (unsigned int i = 0; i < size - 1; i++)
    for (unsigned int j = 0; j < size - 1 - i; j++)
    {
        if (array[j] > array[j + 1])
        {
        //####################implement here################
            [redacted]
        //#################################################
        }
    }
```

- Hint
  - Index “i” : Used for passes
  - Index “j” : Used for comparing two indexes (Blue box)

Bubble Sort

```
for (unsigned int i = 0; i < size - 1; i++)
    for (unsigned int j = 0; j < size - 1 - i; j++)
    {
        if (array[j] > array[j + 1])
        {
        //####################implement here################
            int temp = array[j];
            array[j] = array[j+1];
            array[j+1] = temp;
        //################################################
        }
    }
```

- Hint
  - Index “i” : Used for passes
  - Index “j” : Used for comparing two indexes (Blue box)

Machine Intelligence & Data science LAB

Bubble Sort

```
for (unsigned int i = 0; i < size - 1; i++)
    for (unsigned int j = 0; j < size - 1 - i; j++)
    {
        if (array[j] > array[j + 1])
        {
        //####################implement here################
            int temp = array[j];
            array[j] = array[j+1];
            array[j+1] = temp;
        //##################################################
        }
    }
```

$\approx N^2/2$ iterations

3 operations

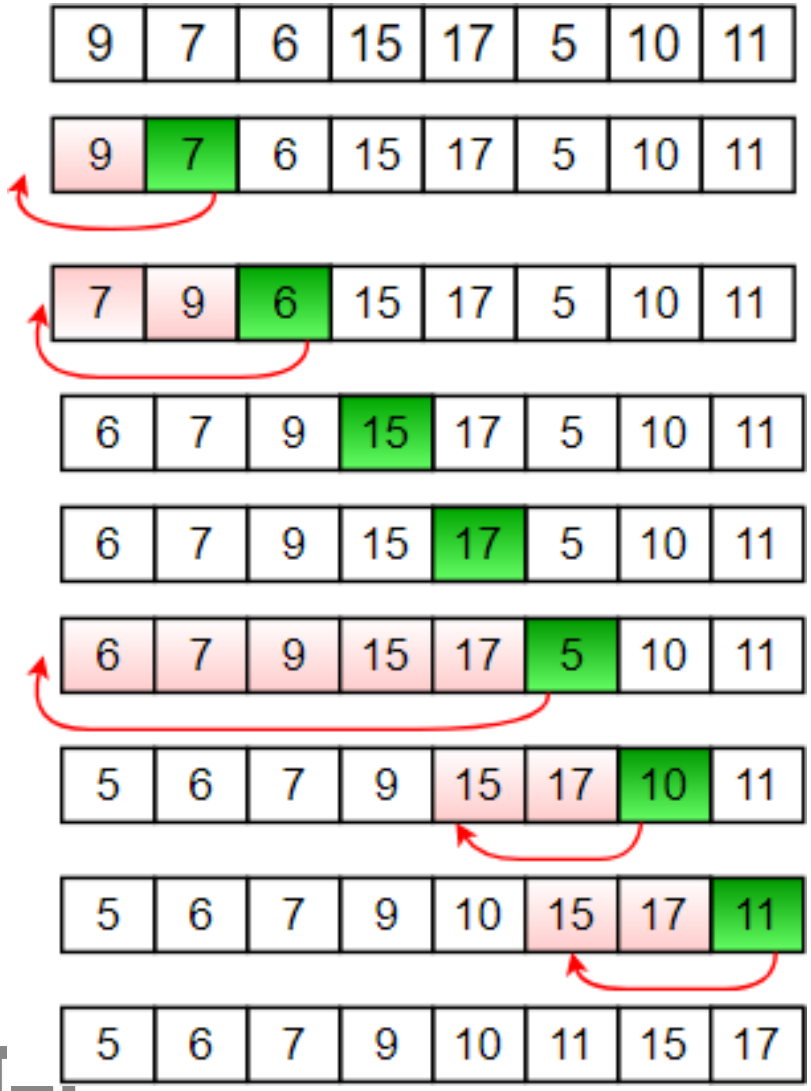
$= O(N^2)$ time complexity

- Hint
  - Index "i" : Used for passes
  - Index "j" : Used for comparing two indexes (Blue box)

Outline

- For loop and while loop
- Array
- What is sorting?
- **Incremental Algorithm**
  - Bubble sort
  - **Insertion sort**
  - Selection sort
  - Heap sort

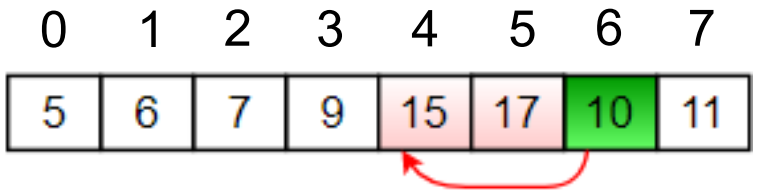
Insertion Sort



- Start from index=1
  - Compare with the indexes in red blocks
  - Insert the target in an **appropriate** location
- Start from index=2

  … continue

Insertion Sort



target = 10
Target Index = 6

0	1	2	3	4	5	6	7
5	6	7	9	15	17	10	11
5	6	7	9	15	17	17	11

Compare index = 5
(17 > target)

No Swap!

5	6	7	9	15	17	17	11
5	6	7	9	15	15	17	11

Compare index = 4
15 > target

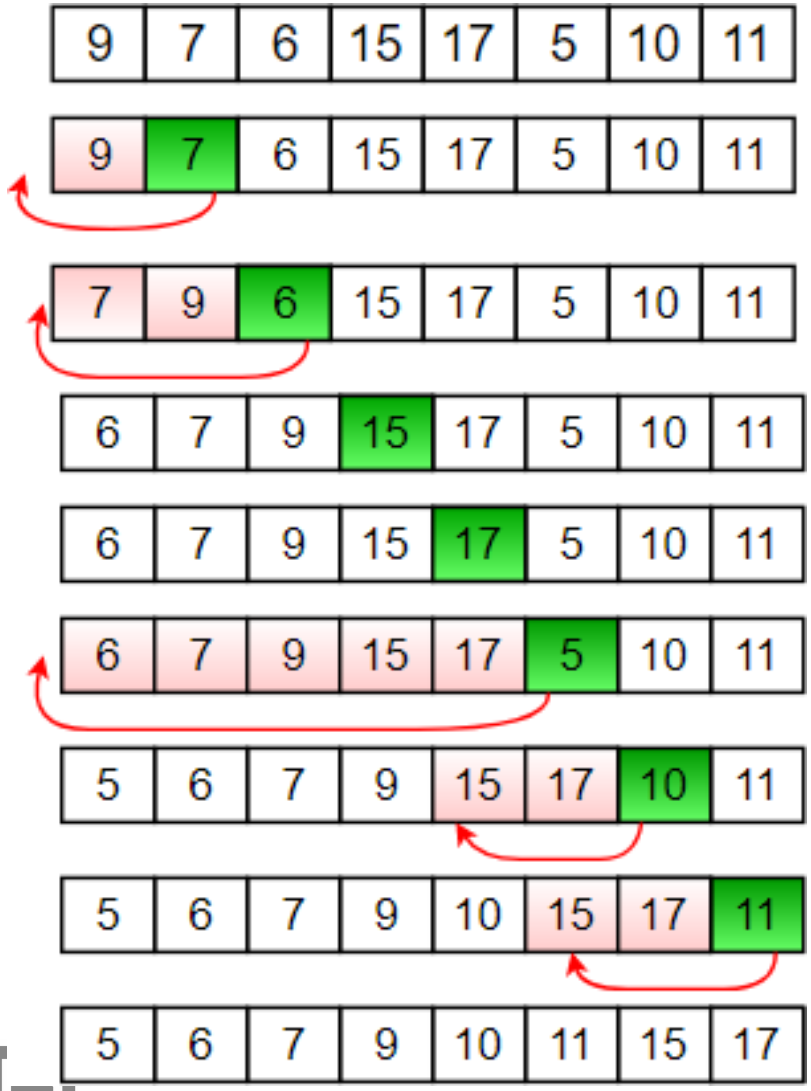
No Swap!

5	6	7	9	15	15	17	11
5	6	7	9	15	15	17	11
5	6	7	9	10	15	17	11

Compare index = 3
9 < target

array [Compare index + 1] = target

Insertion Sort



- Pros :
  - Could swap less times than bubble sort
  - Insertion Sort is very fast when the data is nearly sorted
- Cons :
  - Worst case have time complexity $O(n^2)$

Insertion Sort

```
for (unsigned int i = 1; i < size; i++) {
    int target = array[i];
    int j = i - 1;
    while (j >= 0) {
    //#####################implement here#################

    //################################################
    }
    array[j + 1] = target;
}
```

- Hint
  - Index “i” :Target (Green box)
  - Index “j” : Compare (Red box)

Machine Intelligence & Data science LAB

Insertion Sort

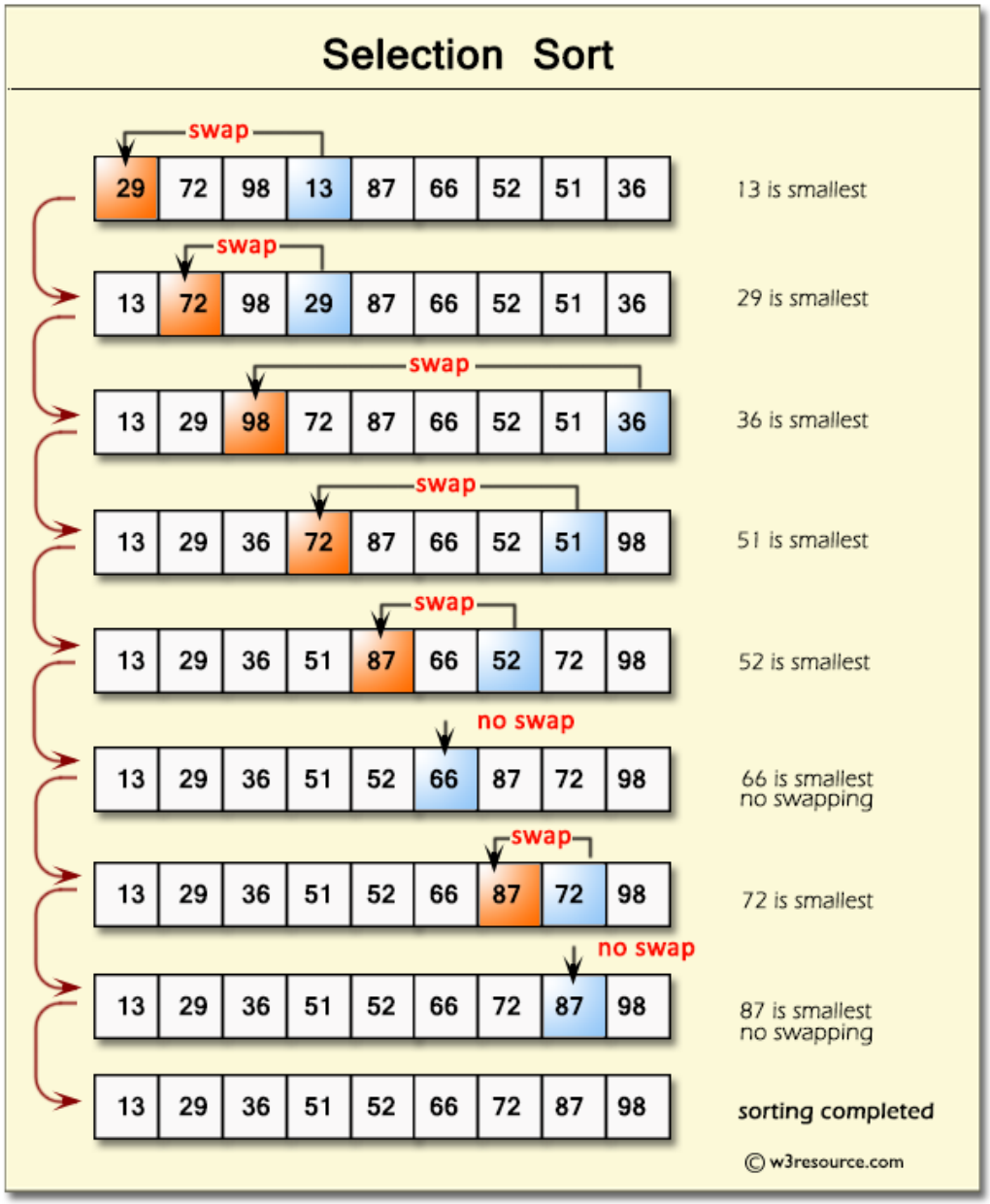
```
for (unsigned int i = 1; i < size; i++) {
    int target = array[i];
    int j = i - 1;
    while (j >= 0) {
    //####################implement here################
        if (array[j] <= target) break;
        else {
            array[j+1] = array[j];
            j--;  // decrease j by 1 (the same as "j = j-1;")
        }
    //#################################################
    }
    array[j + 1] = target;
}
```

- Hint
  - Index "i" : Target (Green box)
  - Index "j" : Compare (Red box)

Outline

- If / For loop / While loop
- Array
- What is sorting?
- **Incremental Algorithm**
  - Bubble sort
  - Insertion sort
  - **Selection sort**
  - Heap sort

Selection Sort



1. Start from the index 0

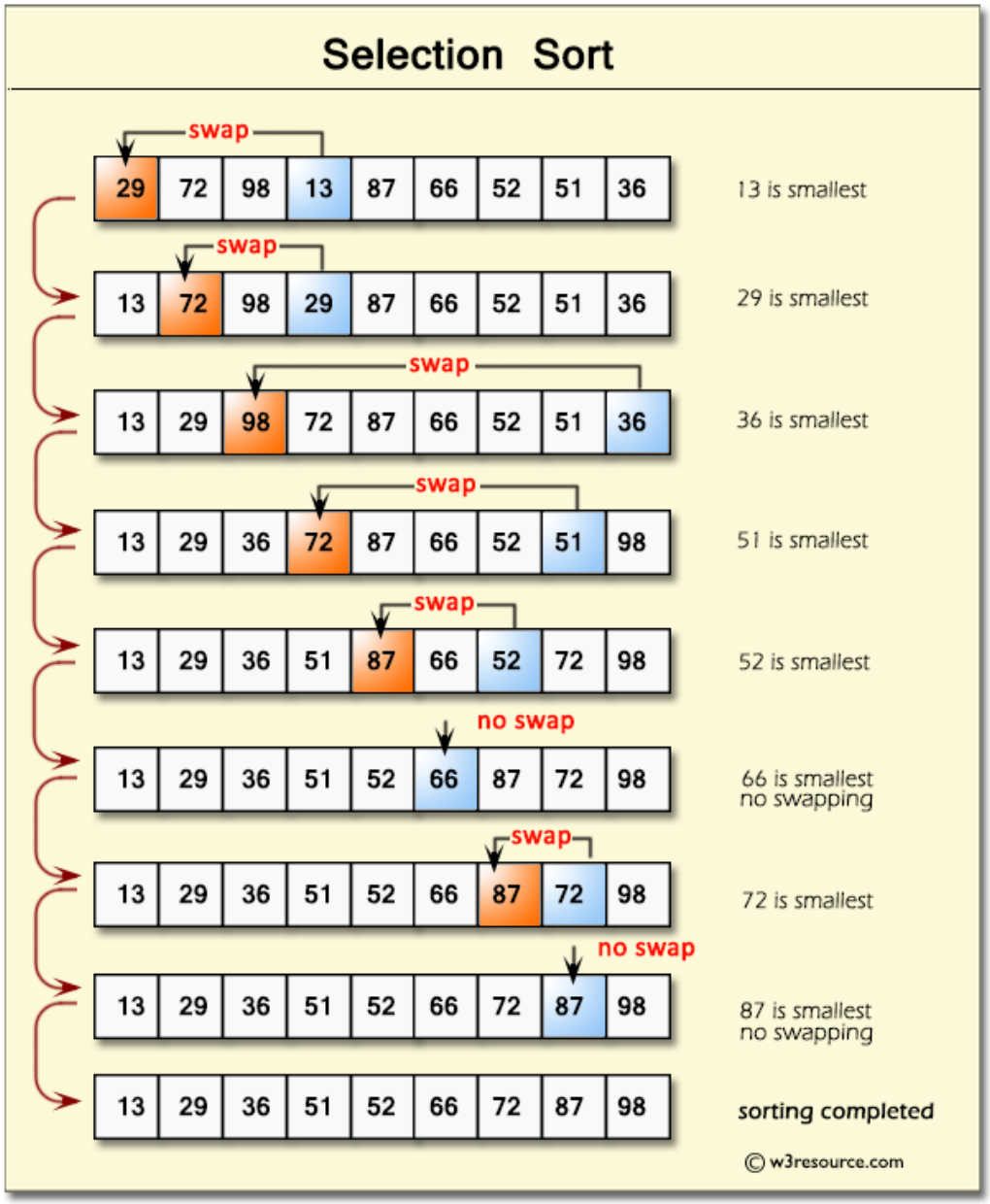
   Swap with the minimum value in the remaining list

2. Start from the index 1

   Swap with the minimum value in the remaining list

.. continue

Selection Sort



- Pros
  - A few swap
- Cons
  - A lot of compare
  - Time complexity $O(n^2)$

Selection Sort

```
for (unsigned int i = 0; i < size - 1; i++)
{
    int min = array[i];
    int min_idx = i;
    for (unsigned int j = i + 1; j < size; j++)
    {
    //####################implement here################
        [redacted]
    //################################################
    }
    int temp = array[i];
    array[i] = array[min_idx];
    array[min_idx] = temp;
}
```

- Hint
  - Index “i” : Red box in the “figure”
  - Index “j” : Blue box in the “figure”

Selection Sort

```
for (unsigned int i = 0; i < size - 1; i++)
{
    int min = array[i];
    int min_idx = i;
    for (unsigned int j = i + 1; j < size; j++)
    {
    //####################implement here################
        if (array[j] < min) {
            min = array[j];
            min_idx = j;
        }
    //#################################################
    }
    int temp = array[i];
    array[i] = array[min_idx];
    array[min_idx] = temp;
}
```

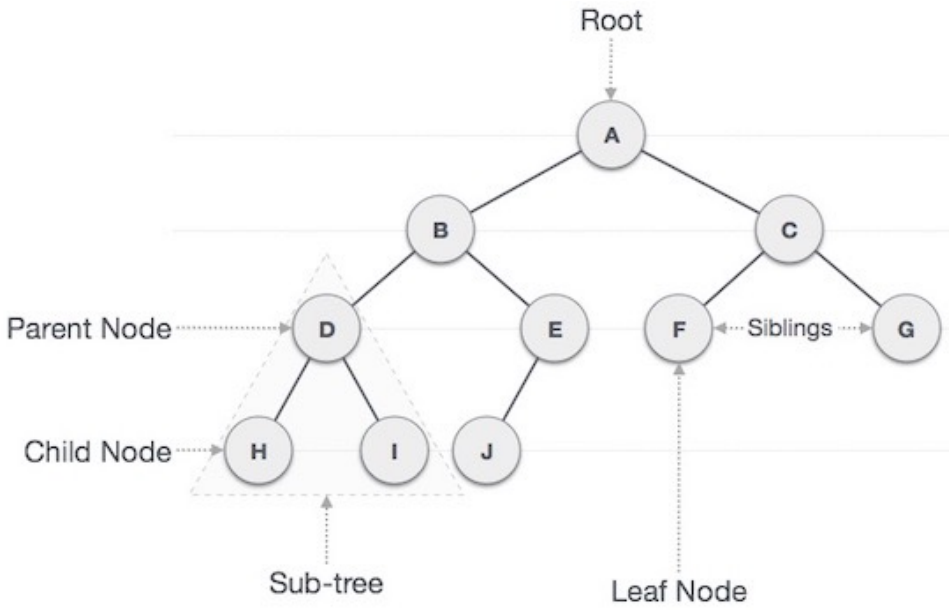
- Hint
  - Index "i" : Red box in the "figure"
  - Index "j" : Blue box in the "figure"

Outline

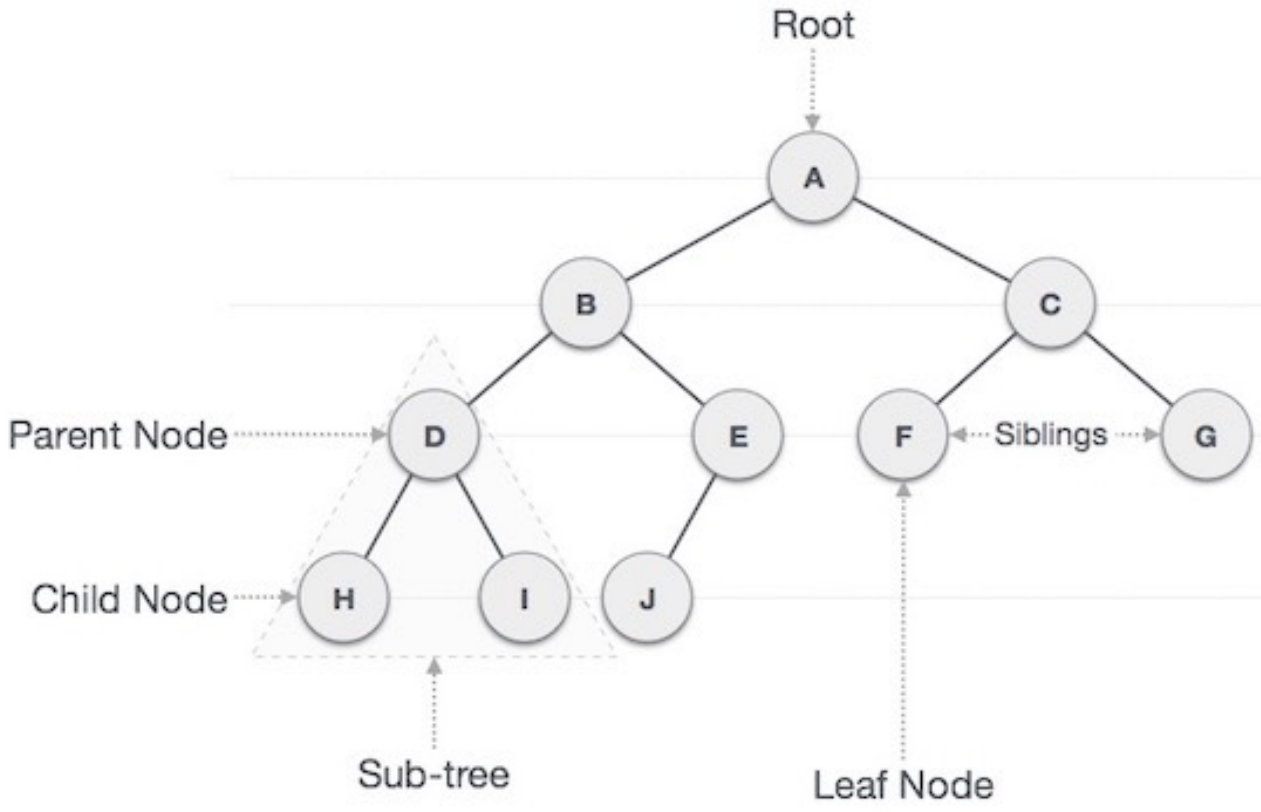
- If / For loop / While loop
- Array
- What is sorting?
- **Incremental Algorithm**
  - Bubble sort
  - Insertion sort
  - Selection sort
  - **Heap sort**

Heap

- Heap
  - Complete binary tree-based structure that satisfies Heap property
- Tree
  - Every node (except root node) has only one parent node.
  - No cycle.



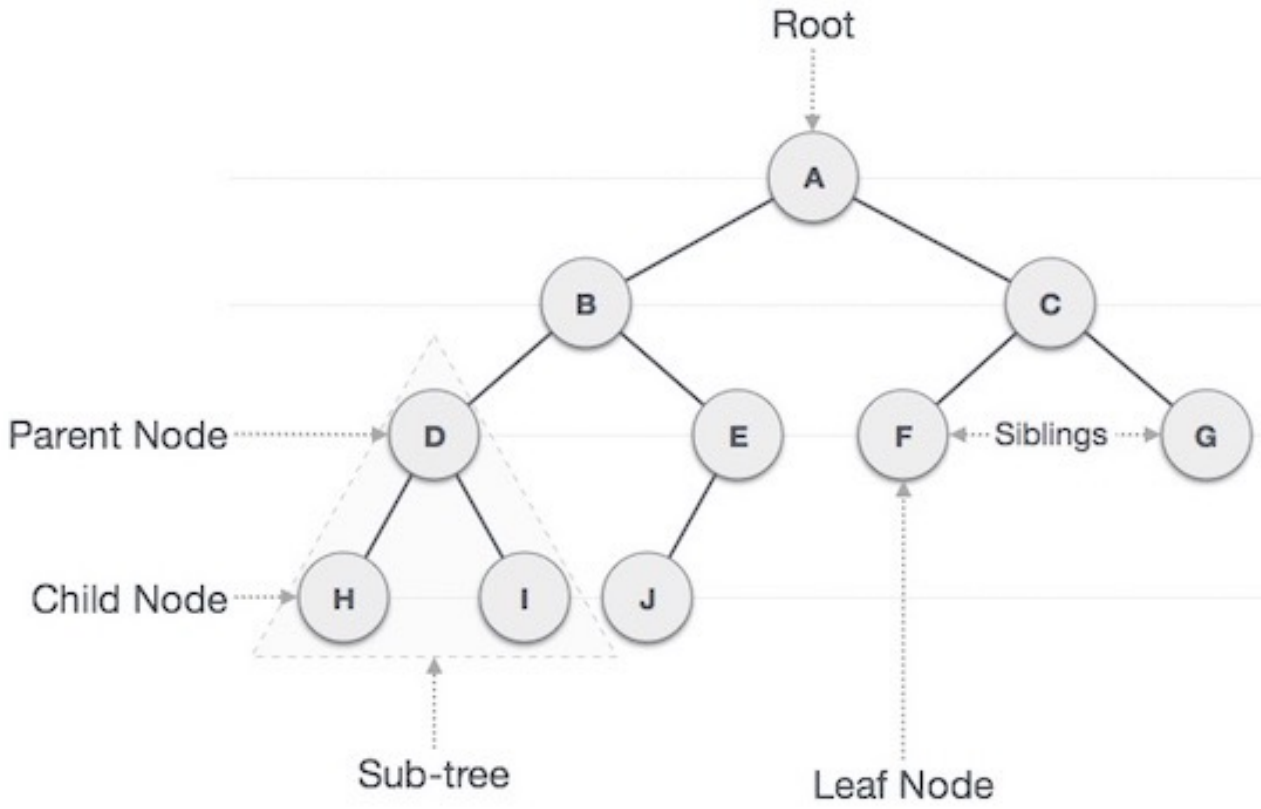
Heap



- **Node** : Elements composing a tree structure
- **Root node/Root** : Starting point of a tree. The highest node that has no parent node
- **Parent node** : A upper node connected towards the root node

https://suyeon96.tistory.com/29

Heap



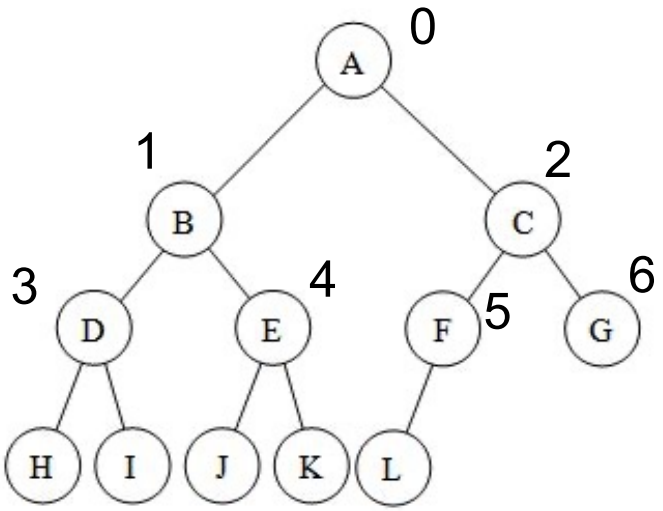
- **Child node** : A lower node connected towards the opposite of the root node
- **Siblings node** : The nodes with the same parent node
- **Leaf node** : The node that has no child node

https://suyeon96.tistory.com/29

Heap

- Complete binary tree
  - Binary tree: each parent node has at most two children
  - Complete binary tree:

    1) All levels except the last are completely filled
    2) Nodes in the last level are filled from left to right

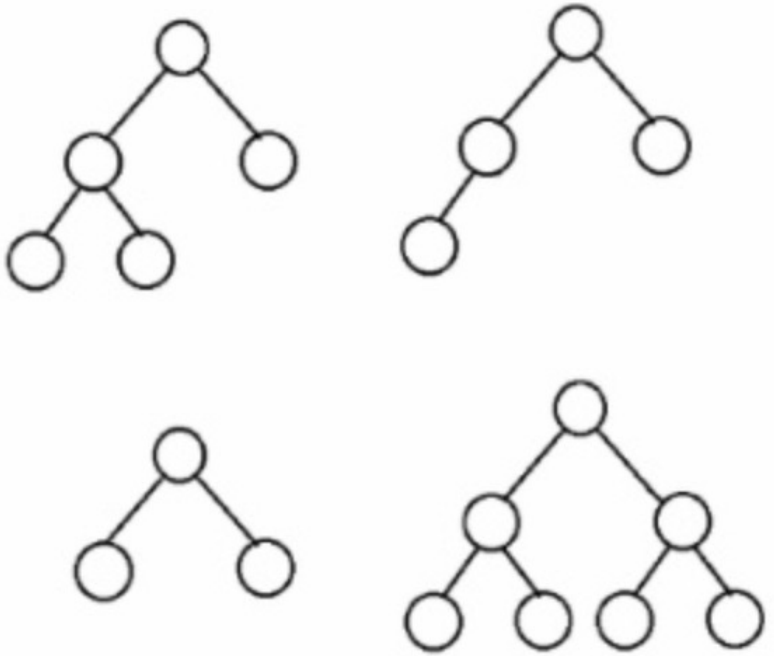


Left child index = 2 * index + 1
Right child index = 2 * index + 2

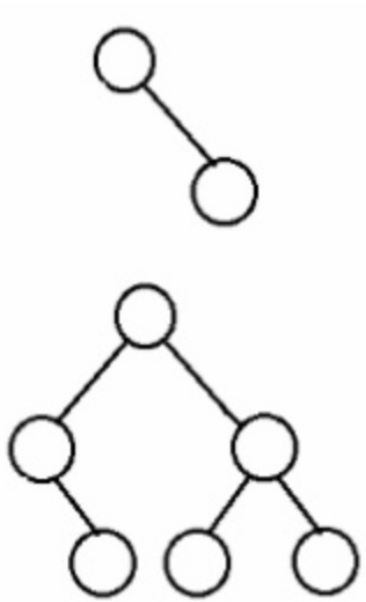
[A, B, C, D, E, F, G, H, I, J, K, L]

Heap

- Complete binary tree



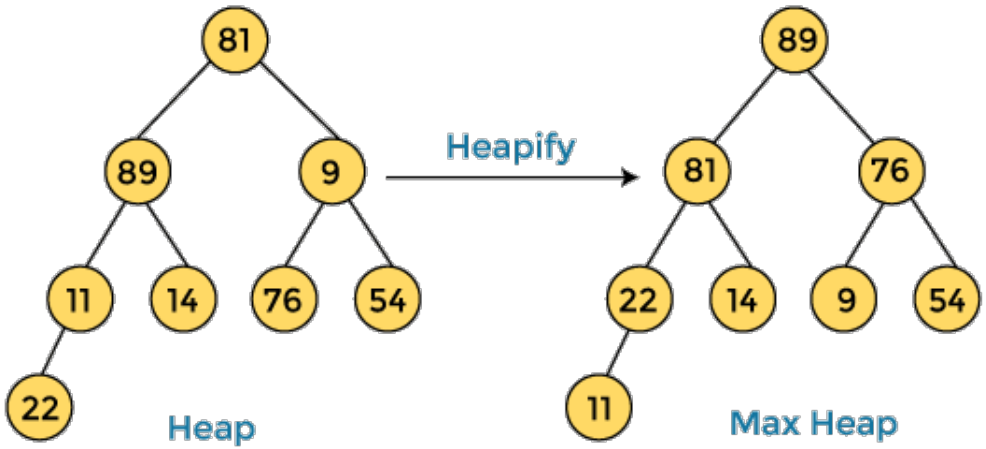
(Complete binary tree)



(**Not** Complete binary tree)

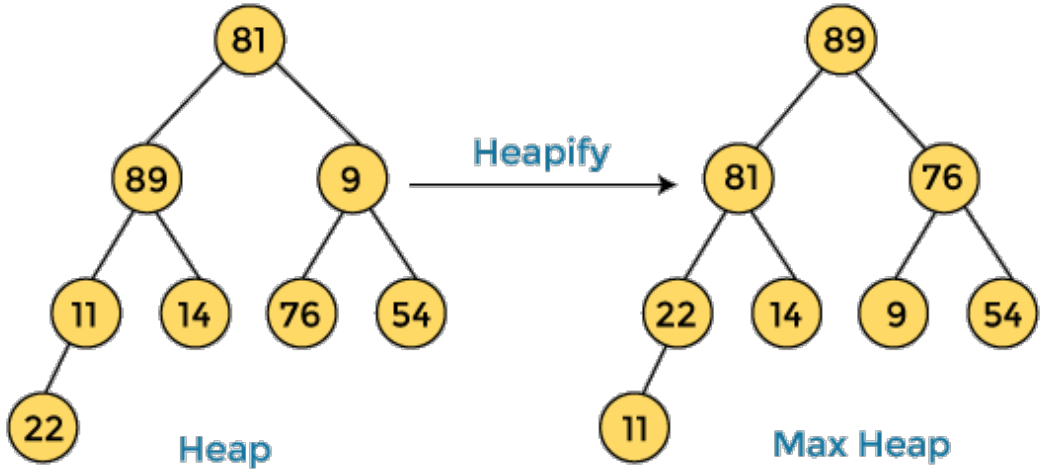
Heap

- Heap
  - Complete binary tree-based structure with Heap property
  - Heap property
    - ***Max heap property:***
      - Parent node's value ≥ children's values (root is largest)
    - *Min heap property:*
      - Parent node's value ≤ children's values (root is smallest)



Heap

- Max-Heapify



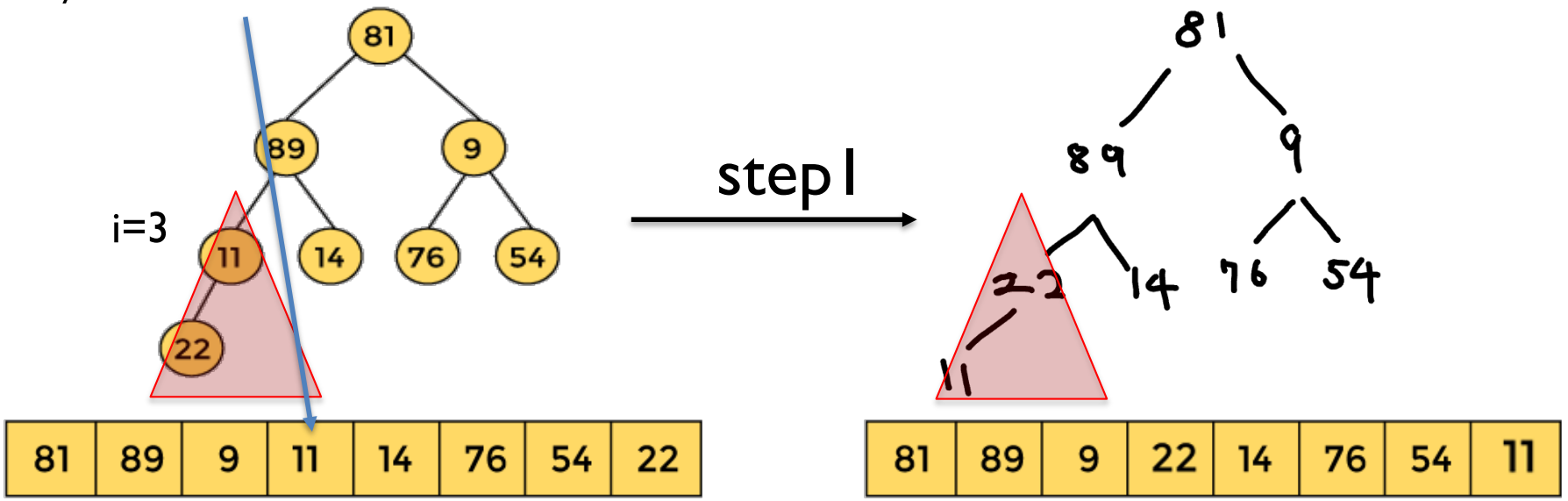
81	89	9	11	14	76	54	22

89	81	76	22	14	9	54	11

Heap
How to make a max heap

initial index = (tree size) / 2 - 1

$8/2 - 1 = 3$



Build-Max-Heap(A)

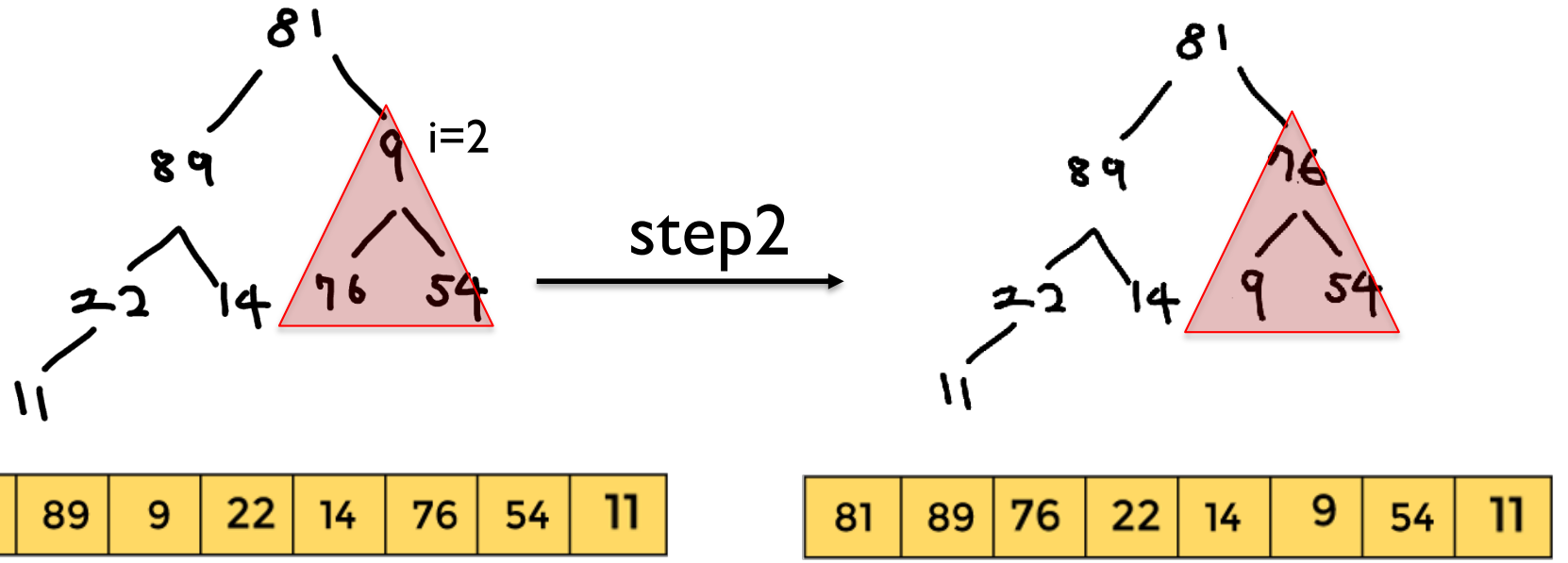
heap_size = length(A)

for i ← heap_size/2 – 1 ***down to*** 0

do **Max-Heapify**(A,i)

Heap
How to make a max heap

Swap the parent's value with the largest of the three numbers



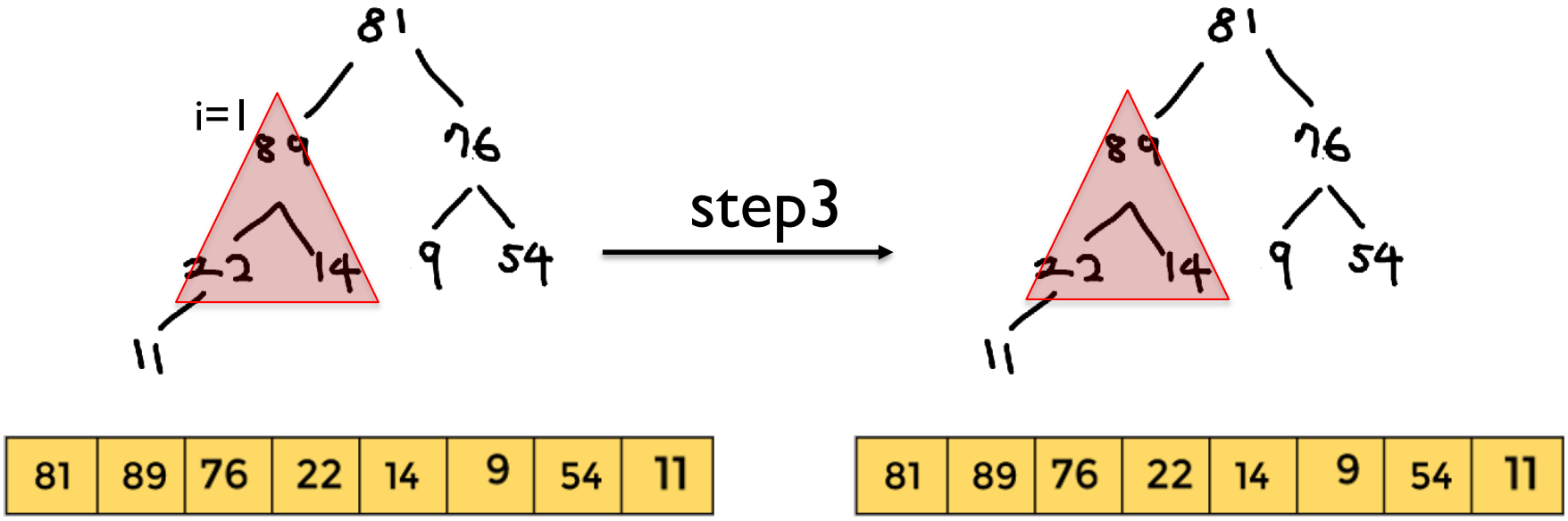
81	89	9	22	14	76	54	11

81	89	76	22	14	9	54	11

Build-Max-Heap(A)

heap_size = length(A)
for i ← heap_size/2 – 1 ***down to*** 0
do **Max-Heapify**(A,i)

Heap
How to make a max heap



81	89	76	22	14	9	54	11

81	89	76	22	14	9	54	11

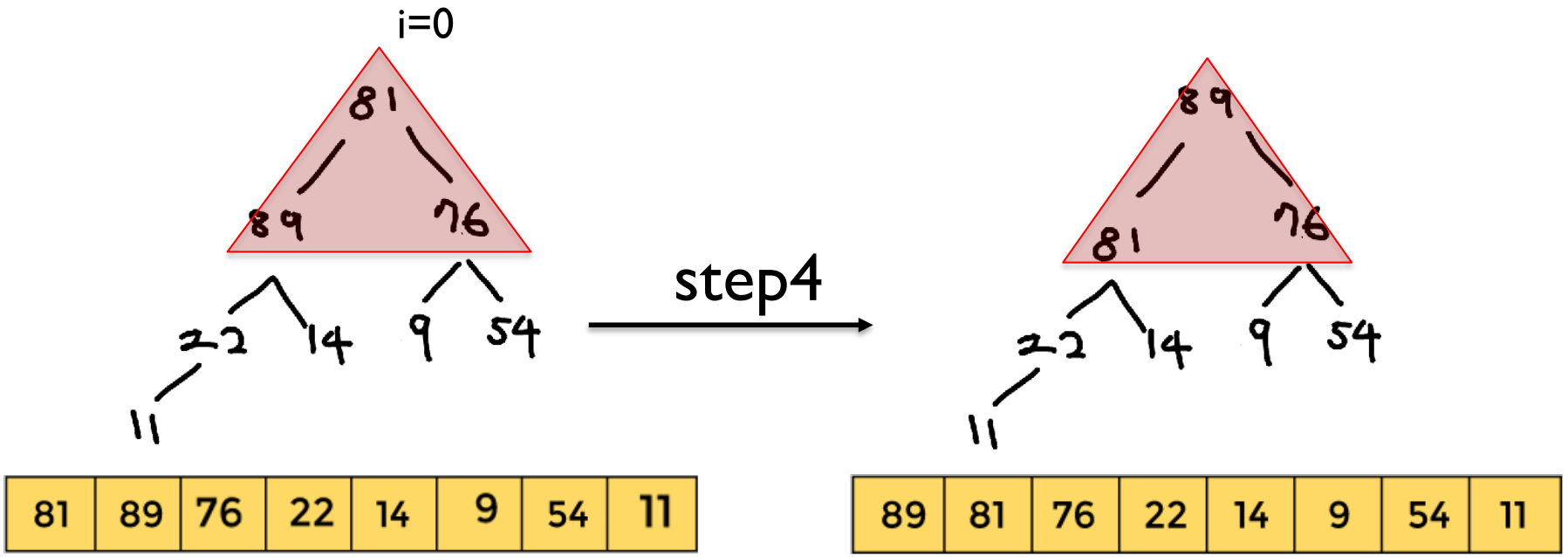
Build-Max-Heap(A)

heap_size = length(A)

for i ← heap_size/2 – 1 ***down to*** 0

do **Max-Heapify**(A,i)

Heap
How to make a max heap



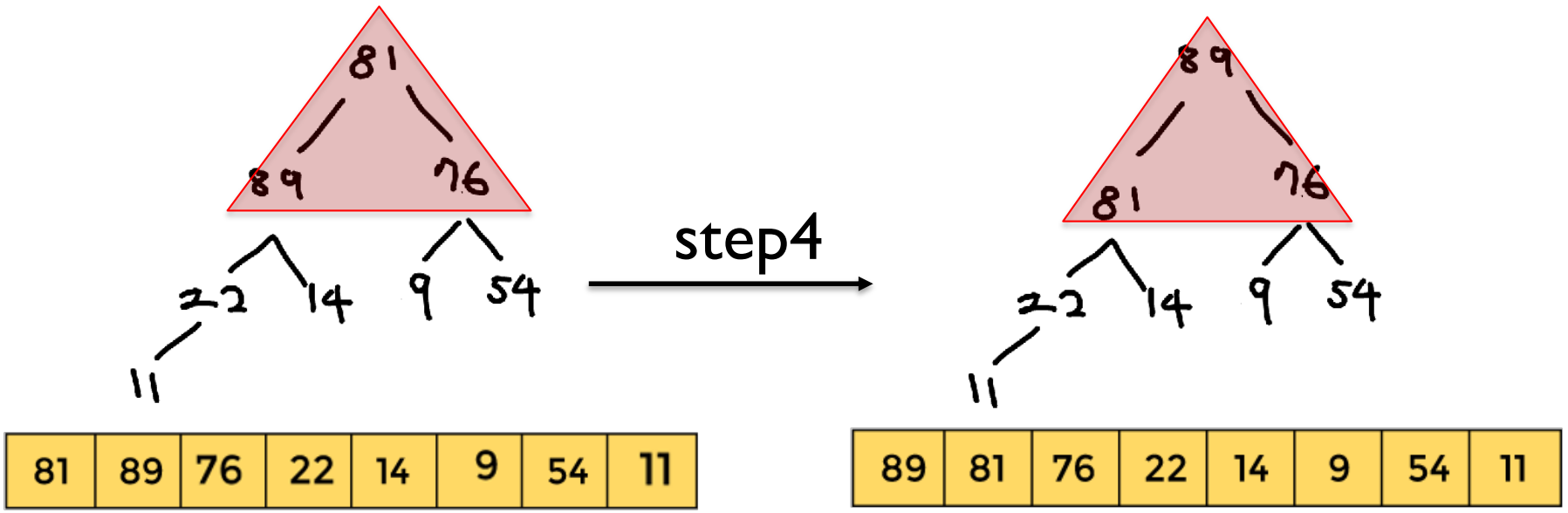
Build-Max-Heap(A)

heap_size = length(A)

for i ← heap_size/2 – 1 ***down to*** 0

do **Max-Heapify**(A,i)

Heap
How to make a max heap



Build-Max-Heap(A)

heap_size = length(A)

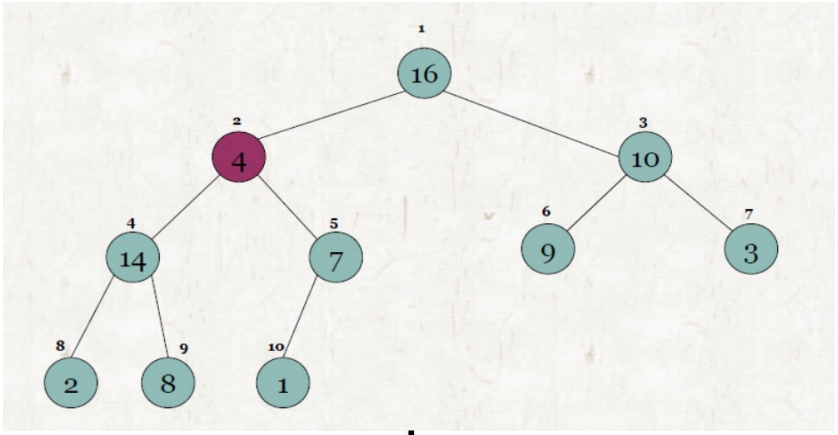
for i ← heap_size/2 – 1 ***down to*** 0

do **Max-Heapify**(A,i)

```
MAX-HEAPIFY(A, i)
1  l ← LEFT(i)
2  r ← RIGHT(i)
3  if l < heap-size[A] and A[l] > A[i]
4     then largest ← l
5     else largest ← i
6  if r < heap-size[A] and A[r] > A[largest]
7     then largest ← r
8  if largest ≠ i
9     then exchange A[i] ↔ A[largest]
10          MAX-HEAPIFY(A, largest)
```

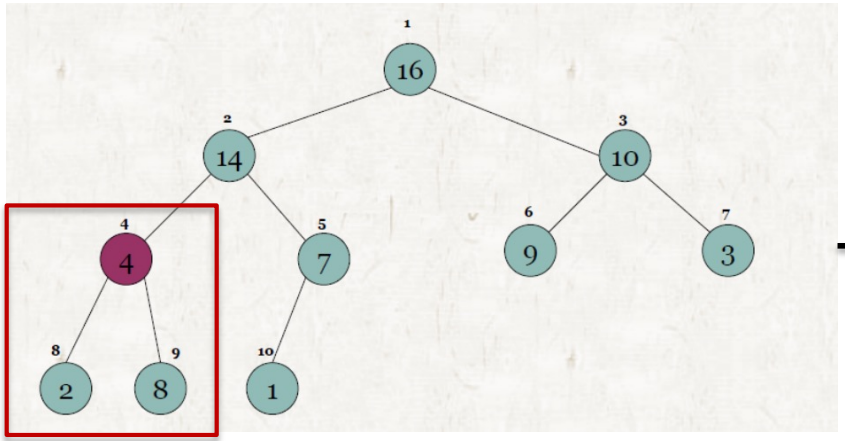
Recursive call
Why?

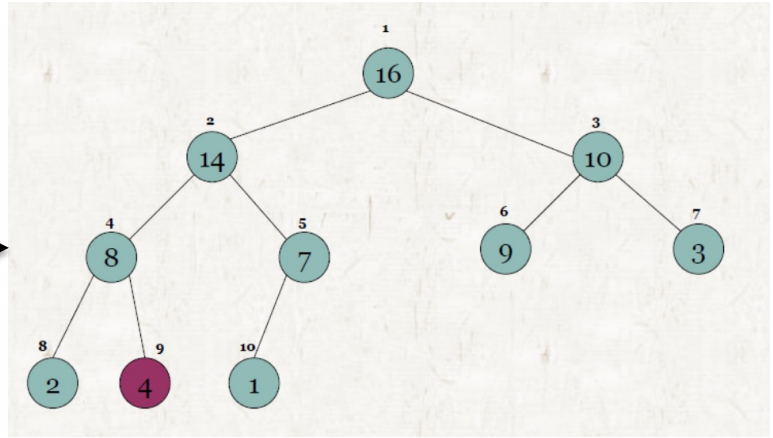
Heap
How to make a max heap



Assume the scenario where we are heapifying at index 2

- During Max-Heapify, 4 moves down the tree, and the Max-Heap property of the lower subtree is violated
- Therefore, we need to heapify that subtree again



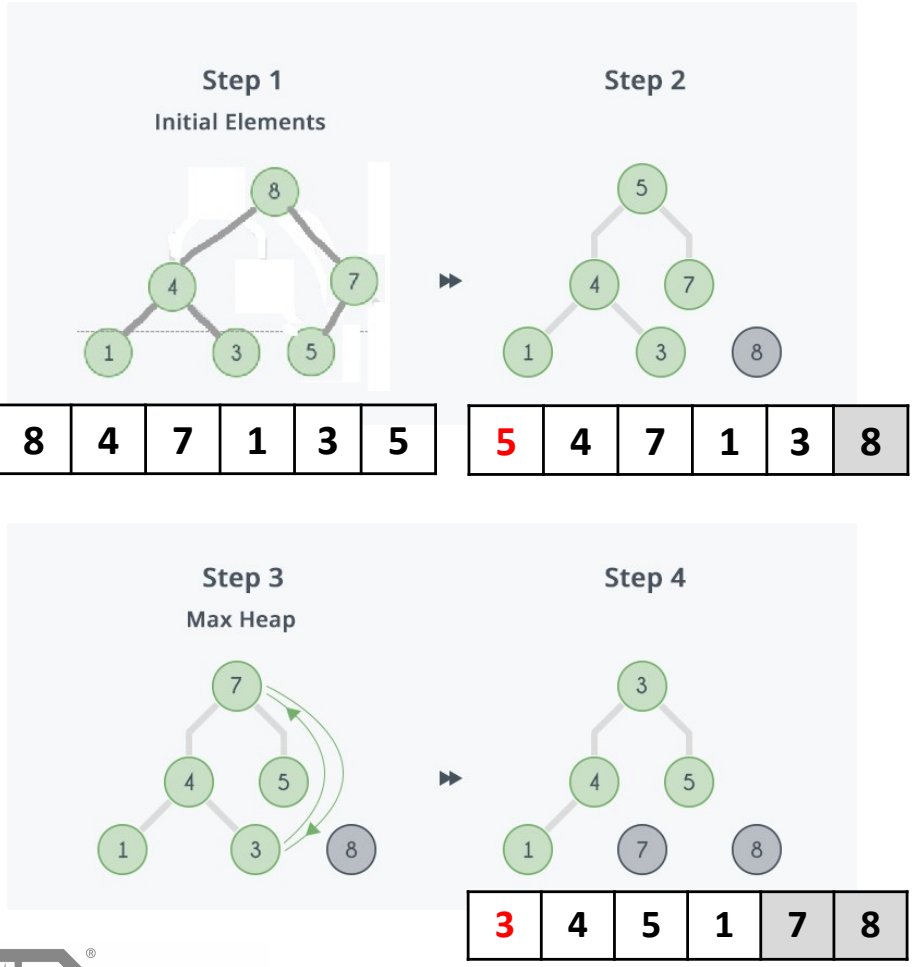


Machine Intelligence & Data science LAB

*We will cover recursive functions in more detail in a later lecture

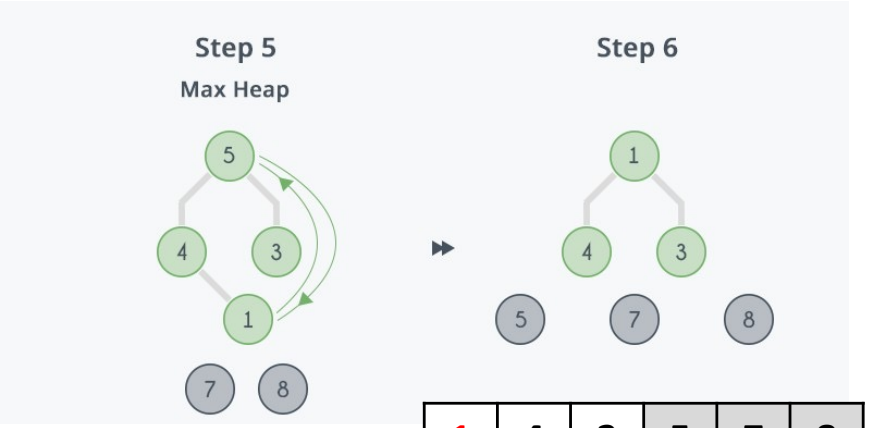
Heap Sort

- From Max Heap

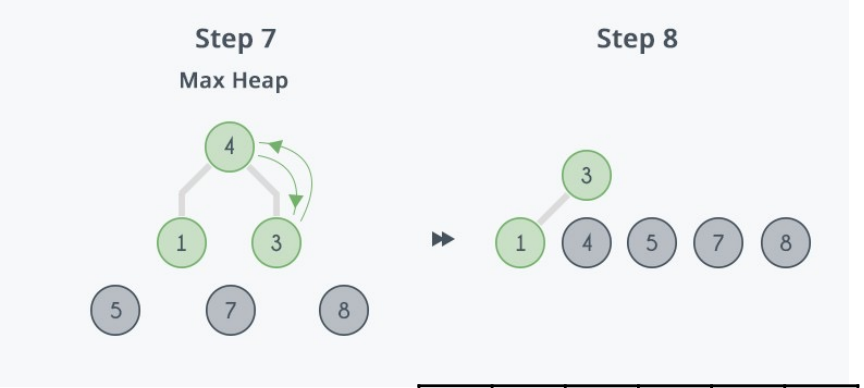


1. Build a max heap from **Arr**
2. The root element, Arr[0] is the maximum element
   → Swap with the last element of **Arr**
3. Max-Heapify from root node
   because all node satisfies the max heap property, except for the root node
4. Repeat 2-3 step

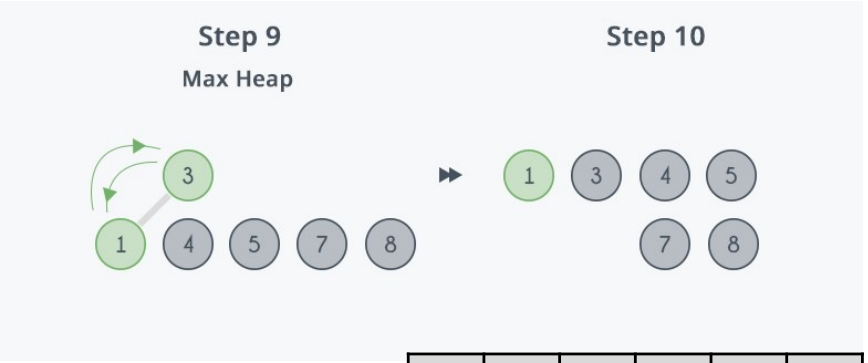
Heap Sort



1	4	3	5	7	8



3	1	4	5	7	8



1	3	4	5	7	8

Machine Intelligence & Data science LAB

Heap Sort

- Pros
  - Nice time complexity
  - The most efficient among the sorting algorithms with $O(n \log n)$, ideally
- Cons
  - According to the state of the input, slower than some algorithms (quick sort, merge sort) in practice.

Build-Max-Heap(A)

heap_size = length(A)

O(n) *for* i ← heap_size/2 – 1 ***down to*** 0

do **Max-Heapify**(A,i)

```
MAX-HEAPIFY(A, i)
1  l ← LEFT(i)
2  r ← RIGHT(i)
3  if l < heap-size[A] and A[l] > A[i]
4     then largest ← l
5     else largest ← i
6  if r < heap-size[A] and A[r] > A[largest]
7     then largest ← r
8  if largest ≠ i
9     then exchange A[i] ↔ A[largest]
10       MAX-HEAPIFY(A, largest)
```

O(log n)
(∵ recursive call and tree depth is log n)

Heap Sort

```
void max_heapify(int* array, int index, int heap_size) {
    int L = 2 * index + 1;
    int R = 2 * index + 2;
    int largest;

    //####################implement here################
    largest = index;
    if (L < heap_size && array[L] > array[index])
        largest = L;
    if (R < heap_size && array[R] > array[largest])
        largest = R;
    if (largest != index) {
        [redacted]
    }
    //#####################################################
}
```

Array argument

```
MAX-HEAPIFY(A, i)
1  l ← LEFT(i)
2  r ← RIGHT(i)
3  if l < heap-size[A] and A[l] > A[i]
4     then largest ← l
5     else largest ← i
6  if r < heap-size[A] and A[r] > A[largest]
7     then largest ← r
8  if largest ≠ i
9     then exchange A[i] ↔ A[largest]
10          MAX-HEAPIFY(A, largest)
```

Heap Sort

Build-Max-Heap(A)

heap_size = length(A)

for i ← heap_size/2 – 1 ***down to*** 0

do **Max-Heapify**(A,i)

```
int heap_size = size;
for (int i = size/2 - 1; i >= 0; i--)
    max_heapify(array, i, heap_size);

for (int i = size - 1; i > 0; i--) {
    //#####################implement here#################
    [redacted]
    //#################################################
}
```

Build-max-heap

Sorting

Heap Sort

```
void max_heapify(int* array, int index, int heap_size) {
    int L = 2 * index + 1;
    int R = 2 * index + 2;
    int largest;

    //####################implement here################
    largest = index;
    if (L < heap_size && array[L] > array[index])
        largest = L;
    if (R < heap_size && array[R] > array[largest])
        largest = R;
    if (largest != index) {
        int temp = array[index];
        array[index] = array[largest];
        array[largest] = temp;
        max_heapify(array, largest, heap_size);
    }
    //#################################################
}
```

Array argument

```
MAX-HEAPIFY(A, i)
1  l ← LEFT(i)
2  r ← RIGHT(i)
3  if l < heap-size[A] and A[l] > A[i]
4     then largest ← l
5     else largest ← i
6  if r < heap-size[A] and A[r] > A[largest]
7     then largest ← r
8  if largest ≠ i
9     then exchange A[i] ↔ A[largest]
10          MAX-HEAPIFY(A, largest)
```

Machine Intelligence & Data science LAB

Heap Sort

- Time complexity $O(n \log n)$

```
int heap_size = size;
for (int i = size/2 - 1; i >= 0; i--)
    max_heapify(array, i, heap_size); // O(logn)

for (int i = size - 1; i > 0; i--) {
    //#####################implement here#################
    int temp = array[0];
    array[0] = array[i];
    array[i] = temp;
    max_heapify(array, 0, --heap_size); // O(logn)
    //#####################################################
}
```

O(nlogn)

O(nlogn)

O(nlogn)

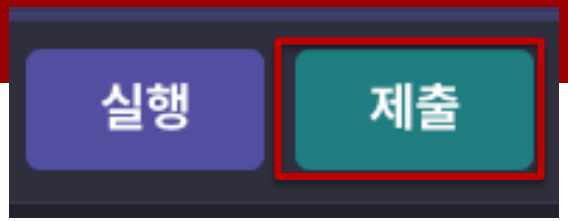
Assignment

- Problem
  - Implement Bubble sort / Insertion sort / Selection sort / Heap sort **(Descending order)**
- Example
  - **Input:** array size, sorting type, and array with random order.
  - **Output:** sorted array with descending order.
  - Ex) Size 5, bubble sort, input array= [12, 8, 11, 2 ,19]

    Output array= [19,12,11,8,2]

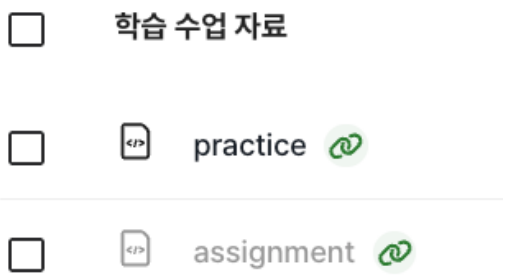
```
Give me the size : 5
Give me the type of algorithm (0: Bubble, 1: Insertion, 2: Selection, 3: Heap): 0
12 8 38 3 1
38 12 8 3 1
Reset
```

Assignment



- Due date : 9/15(Mon) 14:30 PM
- Push “제출” button on Elice
- There is no hidden case
  - A result like the one on the right means you got 100 points.

- Once the due date has passed, the assignment will be hidden and can no longer be submitted



Lab 2 week 1 assignment

1. Environment setting assignment
2. Code of Ethics

Due date : 9/15 (Mon) 14:30 PM

As some students have joined during the "수강신청변경기간" (course add/drop) period,

the deadline will be extended by one week

Attendance Check

- Quiz in Elice