</talentlabs>

Lecture 5
Sorting Algorithms - Part 1

</talentlabs>

Agenga

- Sorting Algorithms
- Bubble Sort
- Insertion Sort

</talentlabs>

Sorting Algorithms

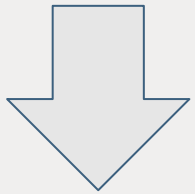
</talentlabs>

</talentlabs>

Sorting Algorithms

- Sorting refers to sorting making a randomly ordered array into an array with increasing or decreasing order.

27	34	10	70	48



Sorting in an increasing order

10	27	34	48	70

</talentlabs>

Array Sorting Algorithms

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	θ(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	θ(n+k)	O(n+k)	O(k)
Cubesort	Ω(n)	θ(n log(n))	O(n log(n))	O(n)

Source: https://lamfo-unb.github.io/2019/04/21/Sorting-algorithms/

</talentlabs>

Sorting Algorithm 1 - Bubble Sort

</talentlabs>

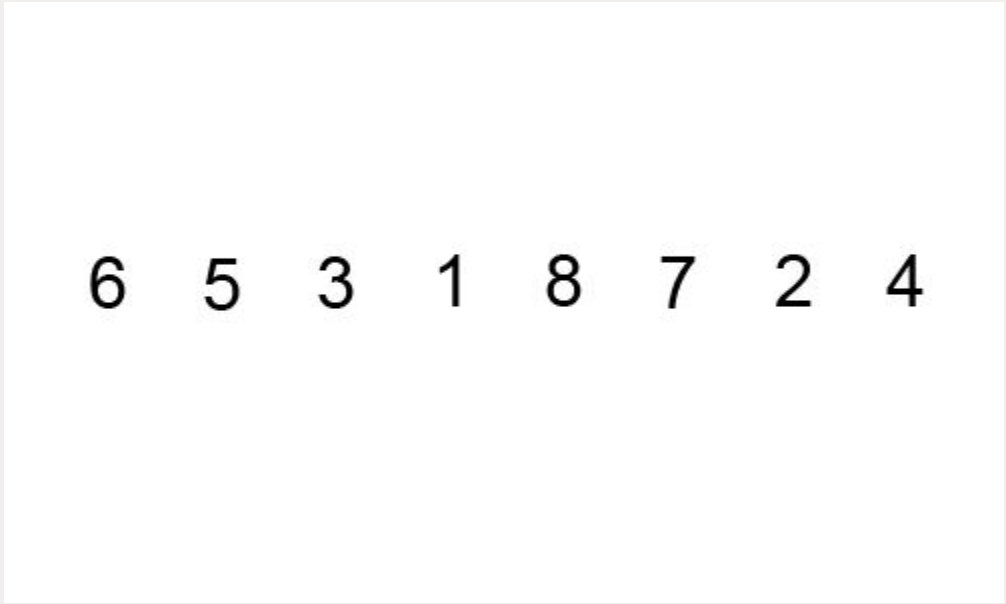
</talentlabs>

Bubble Sort

- Bubble sort works by examining each set of adjacent elements in the array, from left to right, switching their positions if they are out of order.
- The algorithm repeats this process until it can traverse the entire array and cannot find two elements that need to be swapped.

</talentlabs>

Bubble Sort



Key Study Notes:

1. After first iteration, the top 1 element is fixed (as it is already the largest element). After second iteration, the top 2 elements are fixed (as they are already the largest 2 elements)
2. This algorithm is solving one element at a time, from the largest to the smallest

</talentlabs>

Bubble Sort

Consider using bubble sort to sort this array in increasing order.

14	33	28	40	10

Bubble Sort

1st: Bubble sort starts with the first two elements, comparing them to check which one is greater.

14	33	28	40	10

Since 33 > 14, it is in the correct position

Bubble Sort

2nd: We compare 33 with 28.(Compare 2nd and 3rd element)

14	33	28	40	10

Since 28 < 33, it is in incorrect position.

So we need to swap the position of 28 and 33.

The new array will be:

14	28	33	40	10

Bubble Sort

3rd: We compare 33 with 40.(Compare 3rd and 4th element)

14	28	33	40	10

Since 40 > 33, it is in the correct position

Bubble Sort

4th: We compare 40 with 10.(Compare 4th and 5th element)

14	28	33	40	10

Since 10 < 40, it is in incorrect position.

So we need to swap the position of 10 and 40.

The new array will be:

14	28	33	10	40

Bubble Sort

5th: Now we have reached the end of the array. Our array look like this:

14	28	33	10	40

But the array is still not sorted.

We need to repeat step 1 to 4 again. Until our array is sorted.

(Note that we don’t need to worry about “40” anymore as we are sure that it is the largest one)

Bubble Sort

We start with:

14	33	28	40	10

After 1st iteration:

14	28	33	10	40

After 2nd iteration:

14	28	10	33	40

After 3rd iteration:

14	10	28	33	40

After 4th iteration:

10	14	28	33	40

Bubble Sort Algorithm

```
1  const arr = [14, 33, 28, 40, 10]
2
3  for (let i = 0; i < arr.length-1; i++){
4      for (let j = 0; j < arr.length-i-1; j++){
5          if (arr[j] > arr[j+1]){
6              j_value = arr[j]
7              j_plus_1_value = arr[j+1]
8              arr[j] = j_plus_1_value
9              arr[j+1] = j_value
10         }
11     }
12 }
13
14 console.log(arr)
```

How many iterations of swapping we need to do?
In the first cycle, we solved first element
In the second cycle, we solved second element
=> For list of 5 elements, we need 4 cycles to solve 5 elements (as solving 4 elements is same as solving 5 elements)
=> For a list of n elements, we need n-1 iterations

For each iteration, how many comparisons we need to do?
For array length of 5,
In the first iteration (i=0), we need to compare 4 times (1-2, 2-3, 3-4, 4-5)
In the second iteration (i=1), we need to compare 3 times only (1-2, 2-3, 3-4) as the last element is already fixed.
=> For each iteration, we only need to check for “number of unfixed elements - 1” times
=> Number of unfixed elements = array.length - i

</talentlabs>

Sorting Algorithm 2 - Insertion Sort

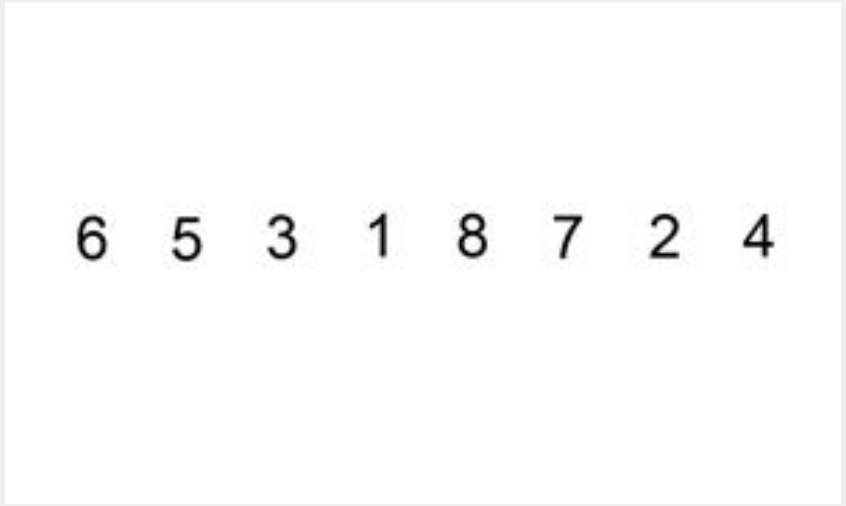
</talentlabs>

</talentlabs>

Insertion Sort

- The idea of insertion sort is to insert the elements to the right position one by one, from the left to right
  - For the first iteration, the focus is to make sure the first 2 items are sorted
  - For the second iteration, the focus is to make sure the first 3 items are sorted

Insertion Sort



By Swfung8 - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=14961606

Key Study Notes:

1. In the first iteration, we focus on the first 2 elements. We will try to “insert” the second element to the correct position
2. In the second iteration, we focus on “inserting” the third element to the correct position
3. For each iteration, we shift the elements that’s larger than our “current element” to the right to make space for the “current element.

</talentlabs>

Insertion Sort

Consider using insertion sort to sort this array in increasing order.

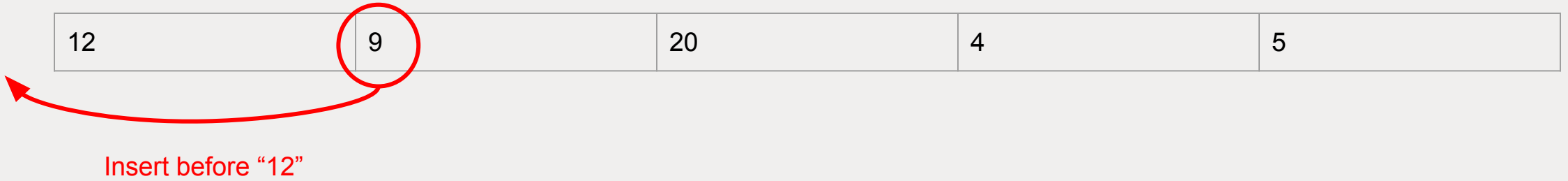
12	9	20	4	5

Loop form i = 1 (second element of the array) to 4 (last element of the array)

Insertion Sort

1st: i = 1 (second element of the array). Since 9 is smaller than 12, move 12 and insert 9 before 12

Before Insertion



After Insertion

9	12	20	4	5

Insertion Sort

2nd: i = 2. 20 will remain at its position as all elements in Array[0..i-1] are smaller than 20.

Both smaller than 20, so no need move the element “20”

No need to worry about elements after “20”

Insertion Sort

3rd: i = 3 (third element of the array).It will start comparing the previous element to the first element in the array. All the element larger than array[3] will move 1 position ahead.

Before Insertion

9	12	20	4	5

Insert before “9”

No need to worry about elements after “5”

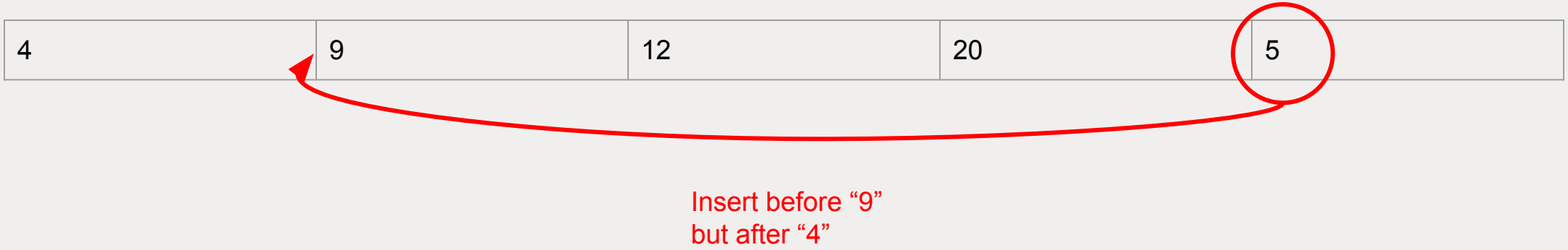
After Insertion

4	9	12	20	5

Insertion Sort

4th: i = 4 (third element of the array).It will start comparing the previous element to the first element in the array. Only the element larger than array[4] will move 1 position ahead.

Before Insertion



After Insertion

4	5	9	12	20

Insertion Sort Algorithm

```
1   const arr = [12, 9, 20, 4, 5]
2
3   for (let i = 1; i < arr.length; i++) {
4       let current = arr[i]
5       let j = i - 1
6       while (j >= 0 && current < arr[j]) {
7           arr[j+1] = arr[j]
8           j--
9       }
10      arr[j+1] = current
11  }
12
13
14  console.log(arr)
```

How many iterations do we need?
We only need to start from the second element (as there will only be insertion for 2+ elements)
So we will start with i = 1 and end with i = length - 1

For each iteration, how many "comparisons and shift" we need to do?
We will need to check for every elements before the "current element", so we will start with **j - 1.**

The meaning here is - for every elements that is smaller than "current element" and before the "current element", we shift it to the right by 1.

</talentlabs>