Quicksort

Shusen Wang

Algorithm Description

Step 1: Picking the Pivot

8	**1**	**4**	**9**	**0**	**6**	**5**	**2**	**7**	**3**

- Select one element from the array as the pivot.
- The pivot should be close to the median.

Step 1: Picking the Pivot

8	1	4	9	0	$\boxed{6}$	5	2	7	3

`left` `center` `right`

`Pick it as pivot!`

- Select one element from the array as the pivot.
- The pivot should be close to the median. (Why?)
- Heuristic:

$$\text{pivot} = \text{median}(\text{left}, \text{center}, \text{right})$$

Step 1: Picking the Pivot

8	1	4	9	0	6	5	2	7	3

pivot

swap

- Select one element from the array as the pivot.
- The pivot should be close to the median. (Why?)
- Heuristic:

$$\text{pivot} = \text{median}(\text{left}, \text{center}, \text{right})$$

- Put the pivot at the end.

Step 1: Picking the Pivot

8	1	4	9	0	3	5	2	7	6

pivot

- Select one element from the array as the pivot.
- The pivot should be close to the median. (Why?)
- Heuristic:

$$\text{pivot} = \text{median}(\text{left}, \text{center}, \text{right})$$

- Put the pivot at the end.

Step 2: Partition

8	1	4	9	0	3	5	2	7	6

- Partition the first $n - 1$ elements to 2 groups.
- Group 1: $\{x \mid x \leq 6\}$.
- Group 2: $\{x \mid x \geq 6\}$.

Step 2: Partition

8	1	4	9	0	3	5	2	7	6

Greater than 6?

Yes!

Step 2: Partition

8	1	4	9	0	3	5	2	7	6

Greater than 6?

Yes!

Less than 6?

No!

Step 2: Partition

8	1	4	9	0	3	5	2	7	6

Greater than 6?

Yes!

Less than 6?

Yes!

Step 2: Partition

8	1	4	9	0	3	5	2	7	6

swap

Step 2: Partition

2	1	4	9	0	3	5	8	7	6

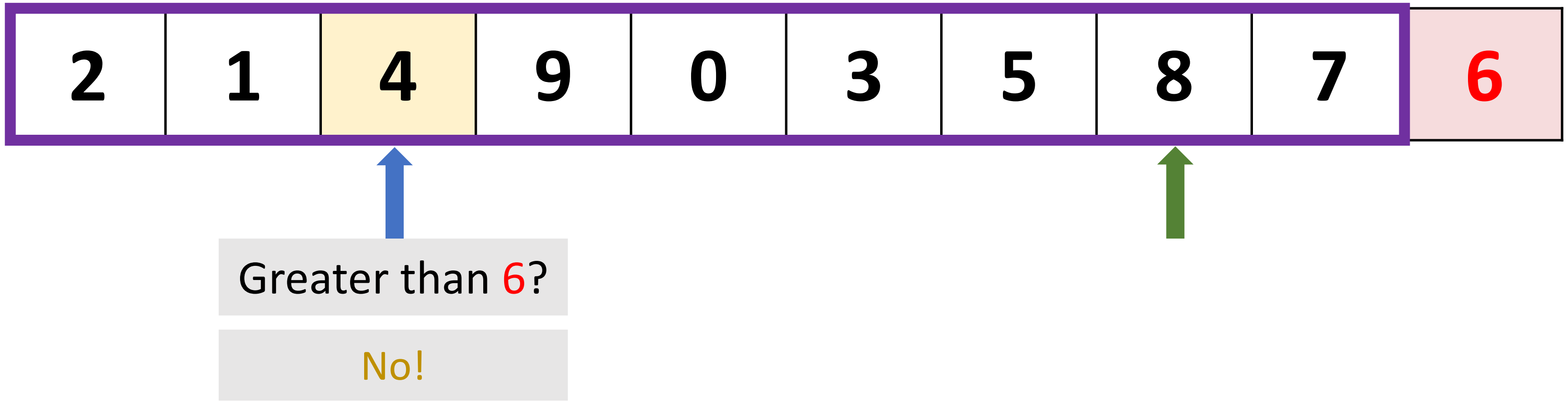
Step 2: Partition

2	1	4	9	0	3	5	8	7	6

Greater than 6?

No!

Step 2: Partition



Step 2: Partition

2	1	4	9	0	3	5	8	7	6

Greater than 6?

Yes!

Step 2: Partition

2	1	4	9	0	3	5	8	7	6

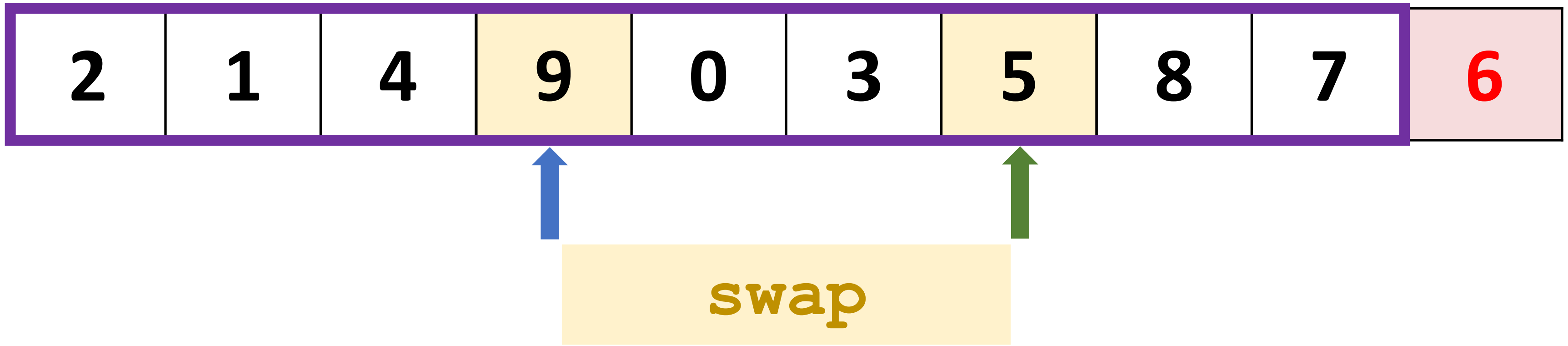
Greater than 6?

Yes!

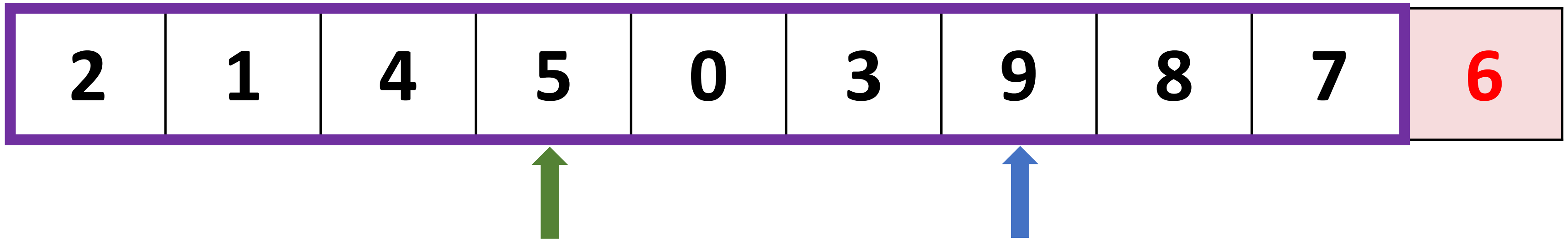
Less than 6?

Yes!

Step 2: Partition



Step 2: Partition



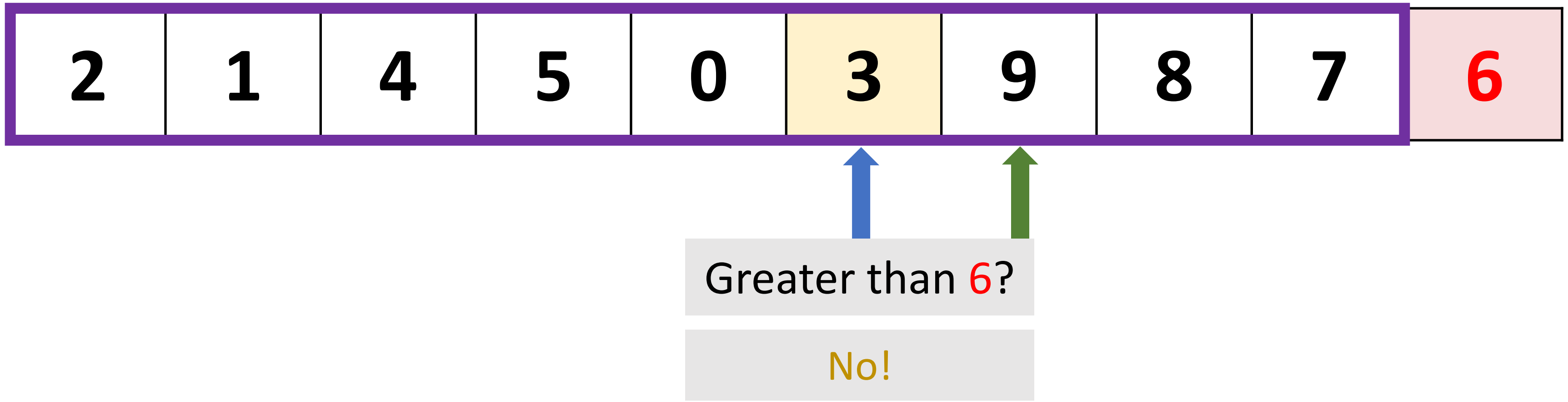
Step 2: Partition

2	1	4	5	0	3	9	8	7	6

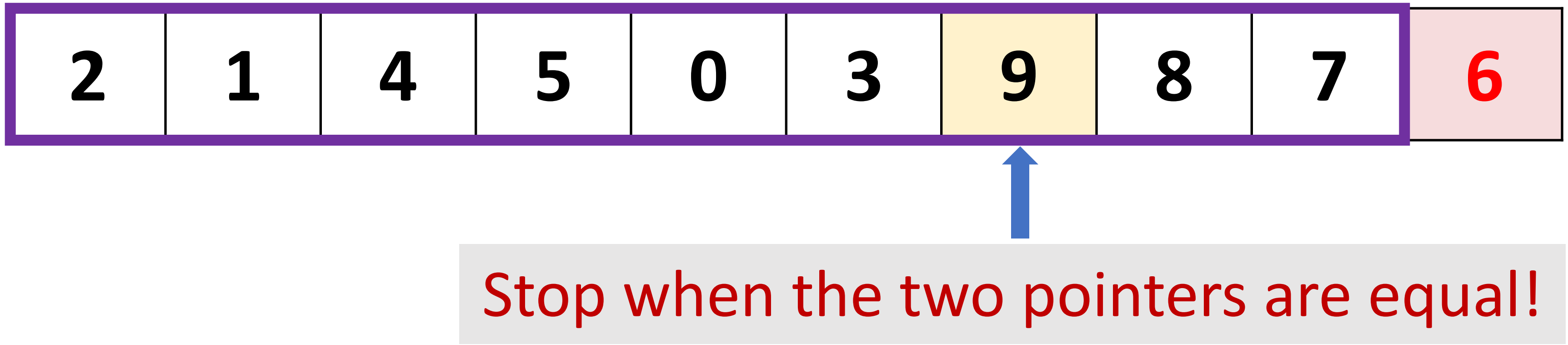
Greater than 6?

No!

Step 2: Partition

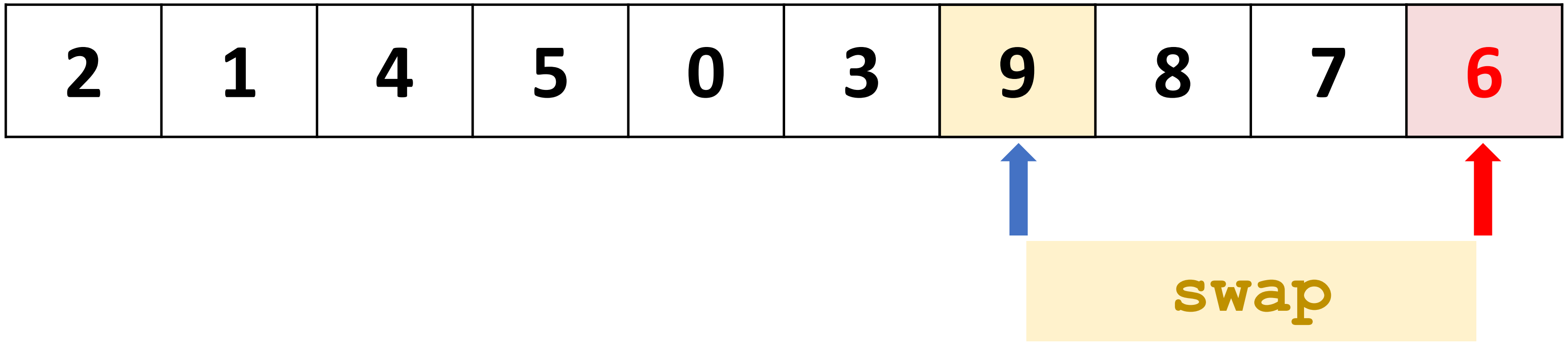


Step 2: Partition



- Partition the first $n - 1$ elements to 2 groups.
- Group 1: $\{x \mid x < 6\}$.
- Group 2: $\{x \mid x \geq 6\}$.

Step 2: Partition



- Partition the first $n-1$ elements to 2 groups.
- Group 1: $\{x \mid x < 6\}$.
- Group 2: $\{x \mid x \geq 6\}$.

Step 2: Partition

2	1	4	5	0	3	6	8	7	9

pivot

- Partition the first $n - 1$ elements to 2 groups.
- Group 1: $\{x \mid x < 6\}$.
- Group 2: $\{x \mid x \geq 6\}$.

Step 2: Partition

2	1	4	5	0	3	6	8	7	9

Group 1: $\{x \mid x \leq 6\}$.

Group 2: $\{x \mid x \geq 6\}$.

Step 3: Recursion

2	1	4	5	0	3	6	8	7	9

Group 1: $\{x \mid x \leq 6\}$.

Group 2: $\{x \mid x \geq 6\}$.

- **Recursion:**
  - Quicksort(Group 1).
  - Quicksort(Group 2).

Step 3: Recursion

2	1	4	5	0	3	6	8	7	9

Group 1: $\{x \mid x \leq 6\}$.

Group 2: $\{x \mid x \geq 6\}$.

- **Recursion:**
  - Quicksort(Group 1).
  - Quicksort(Group 2).
- **Cut-Off:**
  - If array is small (e.g., $n < 10$), Quicksort is not quick.
  - In this case, use insertion sort instead.

```
void quickSort(int arr[], int left, int right) {
    if (left+10 > right) { // for short array
        insertsort(arr, left, right);
    }
    else { // for long array
        int p = selectpivot(arr, left, right);// pivot position
        swap(arr, p, right-1); //put pivot in the end
        int i = partition(arr, left, right);
        swap(arr, i, right-1); // restore pivot
        // recursively sort the two sub-arrays
        quickSort(arr, left, i - 1);
        quickSort(arr, i + 1, right);
    }
}
```

```
void quickSort(int arr[], int left, int right) {
    if (left+10 > right) { // for short array
        insertsort(arr, left, right);
    }
    else { // for long array
        int p = selectpivot(arr, left, right);// pivot position
        swap(arr, p, right-1); //put pivot in the end
        int i = partition(arr, left, right);
        swap(arr, i, right-1); // restore pivot
        // recursively sort the two sub-arrays
        quickSort(arr, left, i - 1);
        quickSort(arr, i + 1, right);
    }
}
```

```
int partition(int arr[], int left, int right) {
    int ptr1 = left;
    int ptr2 = right-1;
    int pivot = arr[right-1];
    while (true) {
        while (arr[++ptr1] < pivot) {}
        while (ptr2 > 0 && arr[--ptr2] > pivot) {}
        if (ptr1 < ptr2)
            swap(arr, ptr1, prt2);
        else
            break;
    }
    return ptr1;
}
```

Time Complexity

Best-Case Time Complexity

Suppose pivot is the median (best-case).

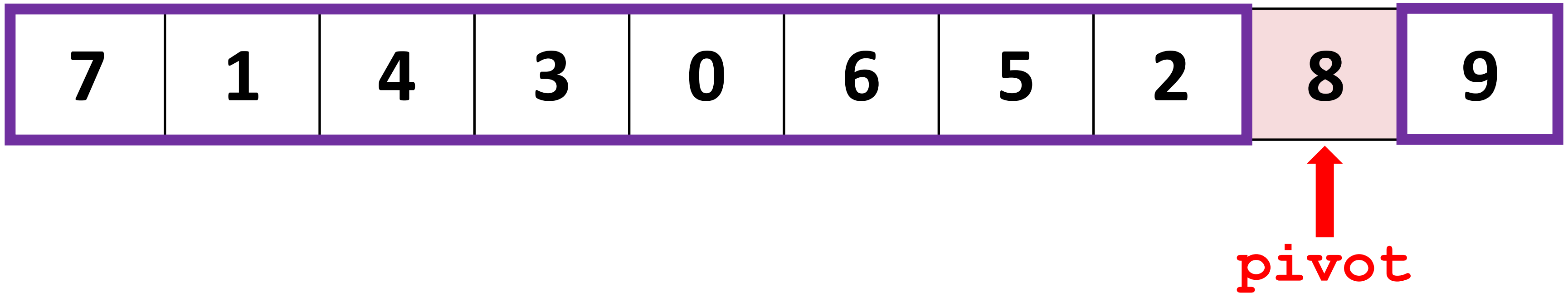
- $T(n)$: Time complexity of sorting size-n array.
- Sizes of Group 1 and Group 2 are both $\frac{n}{2}$.
- Time complexity:

$$T(n) \;=\; 2\,T(n/2) \;+\; c\,n.$$

- ➔ $T(n) = O(n \log n)$.

Worst-Case Time Complexity

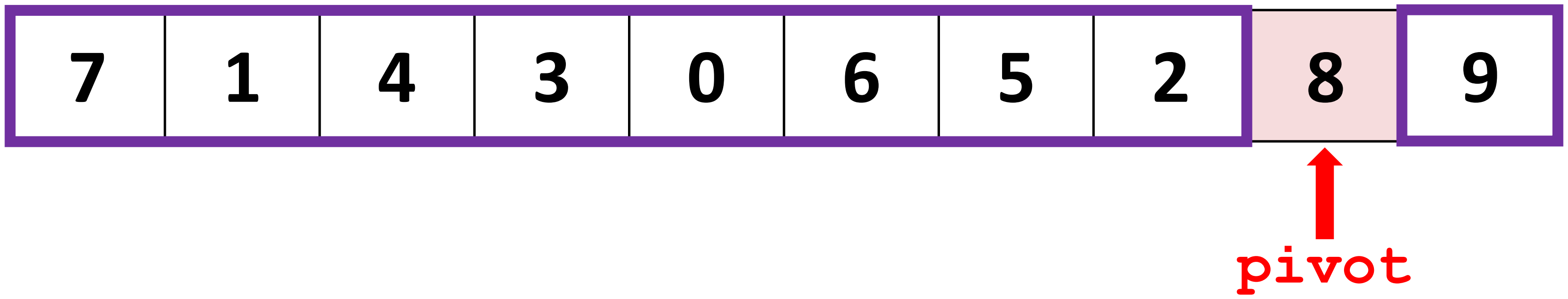
Pivot can be the second largest (worst-case).



- Group 1 has $n-2$ elements.
- Group 2 has only one element.

Worst-Case Time Complexity

Pivot can be the second largest (worst-case).



- Time complexity:

$$T(n) \;=\; T(n-2) + T(1) + cn.$$

- ➔ $T(n) = O(n^2)$.

Time Complexity: Summary

- **Best case:**
  - Sizes of the two sub-groups are both $n/2$.
  - Time complexity: $O(n \log n)$.

- **Worst case:**
  - Size of Group 1: $n - 2$
  - Size of Group 2: 1.
  - Time complexity: $O(n^2)$.

This is why we hope pivot is close to the median!

Time Complexity: Summary

- **Average case:**
  - Assume the data is randomly shuffled.
  - All the elements are equally likely to be the pivot.
  - Expected time complexity: $O(n \log n)$.
  - Algorithm is non-random; data is random.

Time Complexity: Summary

- **Average case:**
  - Assume the data is randomly shuffled.
  - All the elements are equally likely to be the pivot.
  - Expected time complexity: $O(n \log n)$.
  - Algorithm is non-random; data is random.

- **Random pivot:**
  - The position of the pivot is random.
  - Expected time complexity: $O(n \log n)$.
  - Algorithm is random; data is non-random.

Thank You!

Best-Case Time Complexity

Suppose pivot is the median (best-case).

- $\dfrac{T(n)}{n} = \dfrac{T(n/2)}{n\,/\,2} + c$.
- $\dfrac{T(n/2)}{n/2} = \dfrac{T(n/4)}{n/4} + c$.
- $\dfrac{T(n/4)}{n/4} = \dfrac{T(n/8)}{n/8} + c$.

$$\vdots$$

- $\dfrac{T(2)}{2} = \dfrac{T(1)}{1} + c$.

$$\begin{aligned}
\frac{T(n)}{n} &= \frac{T(n/2)}{n\,/\,2} + c \\
&= \frac{T(n/4)}{n/4} + 2c \\
&= \frac{T(n/8)}{n/8} + 3c \\
&= \cdots \\
&= \frac{T(1)}{1} + c\log_2 n
\end{aligned}$$

Worst-Case Time Complexity

Pivot can be the second largest (worst-case).

- $T(n) = T(n-2) + cn.$
- $T(n-2) = T(n-4) + c(n-2).$
- $T(n-4) = T(n-6) + c(n-4).$

$$\vdots$$

- $T(4) = T(2) + 4c.$
- $T(2) = 2c.$
- Thus $T(n) = c \cdot [2c + 4c + \cdots + (n-4) + (n-2) + n]$

$$= O(n^2)$$