Lecture 10 - Quick Sort
Characteristics of Quick Sort

- sort almost in "place", i.e., does not require an additional array
- very practical, average sort performance $O(N \times \log_2(N))$, with small constant factors.
- worst case running time is $O(N^2)$
Some background about Quick Sort

- Invented by Tony Hoare in 1960
Quick Sort - the Principle

- To understand quick-sort, let’s look at a high-level description of the algorithm.
- A divide-and-conquer algorithm.
  - **Divide:** partition array into 2 subarrays such that elements in the lower part ≤ elements in the higher part.
  - **Conquer:** recursively sort the 2 subarrays.
  - **Combine:** trivial since sorting is done in place.
Partitioning Procedure (linear)

```plaintext
function PARTITION(A, p, r)
    x ← A[r]
    i ← p - 1
    for j = p to r-1 do
        if A[j] ≤ x then
            i ← i + 1
            swap A[i] and A[j]
        end if
    end for
    swap A[i+1] and A[r]
    return i+1
end function
```
Partition Procedure Example

- Boundary variable: $p = 0; r = 7$
- $i = -1; j = 0$

```
9  6  5  0  8  2  4  7
```
Partition Procedure Example [1]

1st Iteration:
- Initial values: $i = -1; j = 0; x = 7$
- Final values: $i = -1; j = 1; x = 7$
- No Swap needed

```
9  6  5  0  8  2  4  7
```

2nd Iteration:
- Initial values: $i = -1; j = 1; x = 7$
- Final values: $i = 0; j = 2; x = 7$
- Swap A[0] and A[1]

```
6  9  5  0  8  2  4  7
```
Partition Procedure Example [2]

3rd Iteration:
- Initial values: \( i = 0; \ j = 2; \ x = 7 \)
- Final values: \( i = 1; \ j = 3; \ x = 7 \)

\[
\begin{array}{cccccccc}
6 & 5 & 9 & 0 & 8 & 2 & 4 & 7 \\
\end{array}
\]

4th Iteration:
- Initial values: \( i = 1; \ j = 3; \ x = 7 \)
- Final values: \( i = 2; \ j = 4; \ x = 7 \)

\[
\begin{array}{cccccccc}
6 & 5 & 0 & 9 & 8 & 2 & 4 & 7 \\
\end{array}
\]
Partition Procedure Example [3]

- **5th Iteration:**
  - Initial values: $i = 2; j = 4; x = 7$
  - Final values: $i = 2; j = 5; x = 7$
  - No Swap needed

  \[
  \begin{array}{ccccccc}
  6 & 5 & 0 & 9 & 8 & 2 & 4 & 7 \\
  \end{array}
  \]

- **6th Iteration:**
  - Initial values: $i = 2; j = 5; x = 7$
  - Final values: $i = 3; j = 6; x = 7$

  \[
  \begin{array}{ccccccc}
  6 & 5 & 0 & 2 & 8 & 9 & 4 & 7 \\
  \end{array}
  \]
Partition Procedure Example [4]

7th Iteration:
- Initial values: \( i = 3; \ j = 6; \ x = 7 \)
- Final values: \( i = 4; \ j = 7; \ x = 7 \)

```
6 5 0 2 4 9 8 7
```

Last Execution:
- Initial values: \( i = 4; \ j = 7; \ x = 7 \)
- Final values: \( i = 4; \ j = 7; \ x = 7 \)
- Return 5 (\( i = 5 \))

```
6 5 0 2 4 7 8 9
```
Quick Sort

function QUICKSORT(A, p, r)
    if p < r then
        q ← Partition(A,p,r)
        QuickSort(A, p, q-1)
        QuickSort(A, q+1, r)
    end if
end function
Quick Sort Example
Quick Sort Running Time

- Worst Case: $O(n^2)$
- Best Case: $O(n \times \log n)$
- Average Case: $O(n \times \log n)$

How?
Quick Sort More Examples (Analyze)

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |

- Running time: $O(n^2)$
  - Make notes
Quick Sort More Examples (Analyze)

| 7 | 6 | 5 | 4 | 3 | 2 | 1 |

- Running time: \(O(n^2)\)
  - Make notes
Quick Sort More Examples (Analyze)

| 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |

- Running time: $O(n^2)$
- Make notes
Quick Sort Pivot Splitting

- 1: 9 split - $O(n \times \log n)$
- 1: 99 split - $O(n \times \log n)$
- 1: 999 split - $O(n \times \log n)$
- 0: n-1 split - $O(n^2)$
  - Make notes
One question to think of is can we do a better job in selecting the pivot element?

- We will discuss this in our next class.
Questions