Lecture # 8

Quick Sort (Recursive Algorithm)

Quick Sort

- Quicksort is based on the divide and conquer strategy. Here is the algorithm:

Quick Sort

QUICKSORT(array A, int p, int r)

if (r > p)

then

    i □ a random index from [p..r]

    swap A[i] with A[p]

    q □ PARTITION(A, p, r)

    QUICKSORT(A, p, q - 1)

    QUICKSORT(A, q + 1, r)

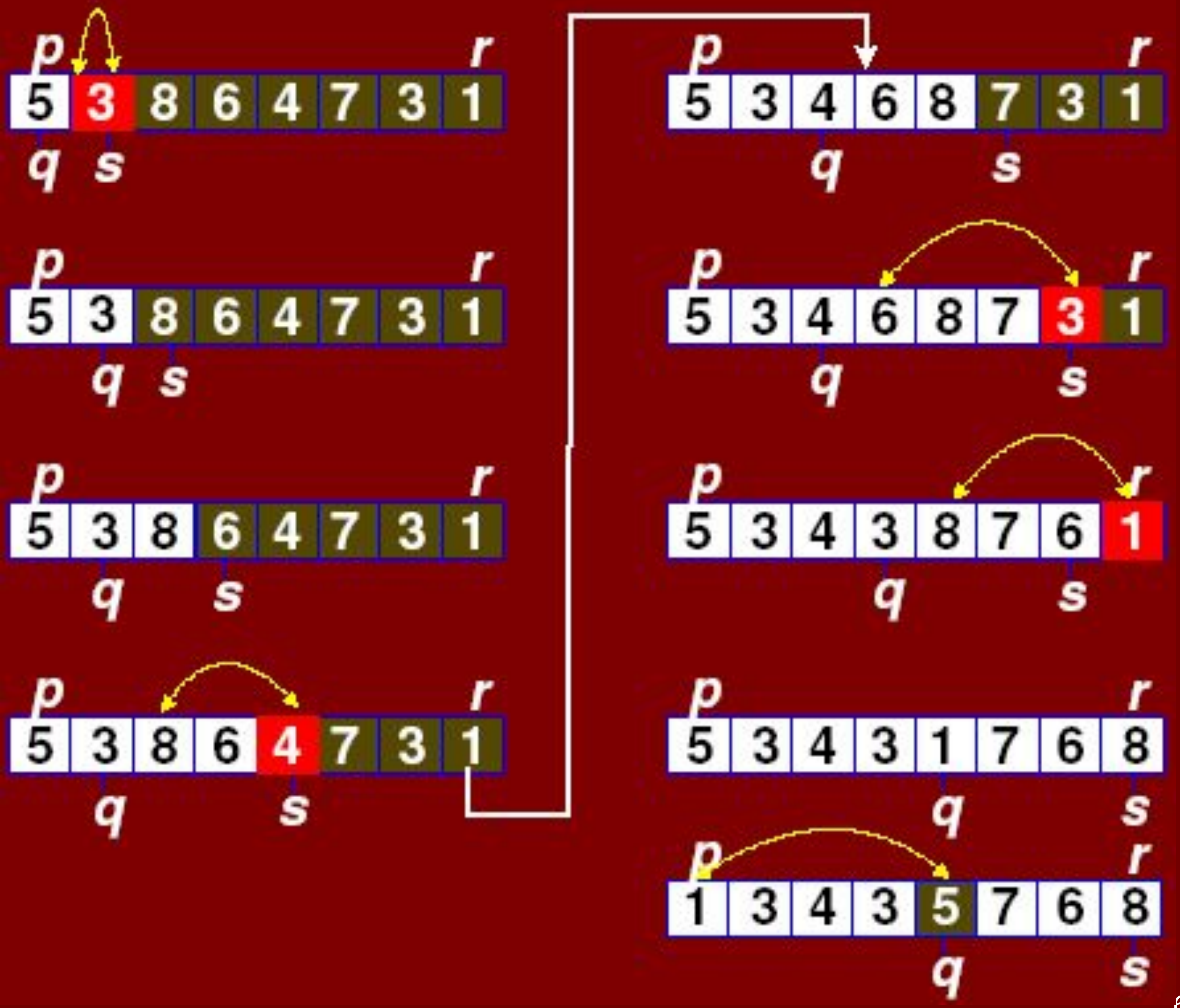
Partition Algorithm

- The partition algorithm partitions the array A[p..r] into three sub arrays about a pivot element x. A[p..q - 1] whose elements are less than or equal to x, A[q] = x,
- A[q + 1..r] whose elements are greater than x.
- We will choose the first element of the array as the pivot, i.e., x = A[p].
- If a different rule is used for selecting the pivot, we can swap the chosen element with the first element. We will choose the pivot randomly

Partition Algorithm

```
PARTITION( array A, int p, int r)
x □ A[p]
q □ p
for s □ p + 1 to r
    do if (A[s] < x)
        then  q □ q + 1
              swap A[q] with A[s]
swap A[p] with A[q]
return q
```

```
PARTITION(array A,int p,int r)
x ☐ A[p]
q ☐ p
for s ☐ p + 1 to r
  do if (A[s] < x)
      then  q ☐ q + 1
       swap A[q] with A[s]
swap A[p] with A[q]
return q
```



Quick Sort Example-2

- The following Figures trace out the quick sort algorithm.
- The first partition is done using the last element, 10, of the array. The left portion are then partitioned about 5 while the right portion is partitioned about 13.
- Notice that 10 is now at its final position in the eventual sorted order.
- The process repeats as the algorithm recursively partitions the array eventually sorting it.

7	6	12	3	11	8	7	1	15	13	17	5	16	14	9	4	10

↓

7	6	12	3	11	8	7	1	15	13	17	5	16	14	9	4	10
7	6	4	3	9	8	2	1	5	10	17	15	16	14	11	12	13

↓

7	6	12	3	11	8	7	1	15	13	17	5	16	14	9	4	10
7	6	4	3	9	8	2	1	5	10	17	15	16	14	11	12	13

↓

7	6	12	3	11	8	7	1	15	13	17	5	16	14	9	4	10
7	6	4	3	9	8	2	1	5	10	17	15	16	14	11	12	13
1	2	4	3	5	8	6	7	9	10	12	11	13	14	15	17	16

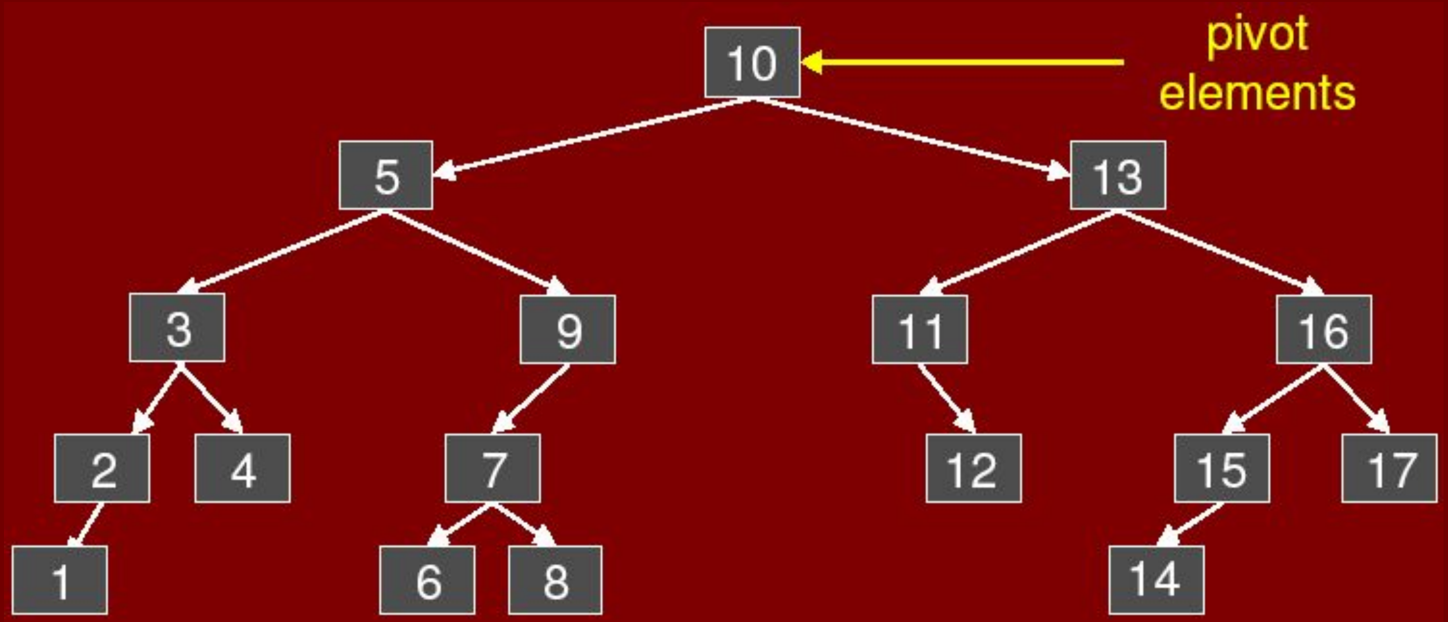
7	6	12	3	11	8	7	1	15	13	17	5	16	14	9	4	10
7	6	4	3	9	8	2	1	5	10	17	15	16	14	11	12	13
1	2	4	3	5	8	6	7	9	10	12	11	13	14	15	17	16

7	6	12	3	11	8	7	1	15	13	17	5	16	14	9	4	10
7	6	4	3	9	8	2	1	5	10	17	15	16	14	11	12	13
1	2	4	3	5	8	6	7	9	10	12	11	13	14	15	17	16
1	2	3	4	5	8	6	7	9	10	11	12	13	14	15	16	17

7	6	12	3	11	8	7	1	15	13	17	5	16	14	9	4	10
7	6	4	3	9	8	2	1	5	10	17	15	16	14	11	12	13
1	2	4	3	5	8	6	7	9	10	12	11	13	14	15	17	16
1	2	3	4	5	8	6	7	9	10	11	12	13	14	15	16	17
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

7	6	12	3	11	8	7	1	15	13	17	5	16	14	9	4	10
7	6	4	3	9	8	2	1	5	10	17	15	16	14	11	12	13
1	2	4	3	5	8	6	7	9	10	12	11	13	14	15	17	16
1	2	3	4	5	8	6	7	9	10	11	12	13	14	15	16	17
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

- It is interesting to note that the pivots form a binary search tree as illustrated in following Figure



Analysis of Quick sort

- The running time of quicksort depends heavily on the **selection of the pivot**. If the rank (index value) of the pivot is very large or very small, then the partition (BST) will be unbalanced.

- Since the pivot is chosen randomly in our algorithm, the expected running time is $O(n \log n)$.

- The worst-case time, however, is $O(n^2)$. Luckily, this happens rarely.

Worst Case Analysis of Quick Sort

- Let's begin by considering the worst-case performance, because it is easier than the average case. Since this is a recursive program, it is natural to use a recurrence to describe its running time.
- But unlike Merge Sort, where we had control over the sizes of the recursive calls, here we do not. It depends on how the pivot is chosen.
- Suppose that we are sorting an array of size n, A[1 : n], and further suppose that the pivot that we select is of rank q, for some q in the range 1 to n.

Worst-Case Time complexity

- The worst-case behavior for quicksort occurs when the partitioning routine produces one subproblem with n - 1 elements and one with 0 elements.
- Assume that this unbalanced partitioning arises in each recursive call. The partitioning costs Θ(n) time.
- A rule of thumb of algorithm analysis is that the worst cases tends to happen at the extremes.

$$T(n) = T(q - 1) + T(n - q) + n$$

$$\begin{aligned} T(n) &= T(n-1) + T(0) + \Theta(n) \\ &= T(n-1) + \Theta(n) . \end{aligned}$$

- In this case, the worst case happens at either of the extremes. If we expand the recurrence for q = 1, we get:

$$
\begin{aligned}
T(n) &= T(n-1) + n \\
&= T(n-2) + (n-1) + n \\
&= T(n-3) + (n-2) + (n-1) + n
\end{aligned}
$$

K times

$= T(n-k) + (n-k-1) + \ldots + (n-2) + (n-1) + n$

Assume $n - k = 0$ then $n = k$

$= 1 + 2 + 3 + \ldots\ldots + (n-2) + (n-1) + n$

This is arithmetic series…. Solve it..

Best-case Analysis of Quick Sort

- We will now show that in the Best/average case, quicksort runs in $\Theta(n \log n)$ time.
- Best-case in the case of quicksort, only depends upon the random choices of pivots that the algorithm makes.
- PARTITION produces two subproblems, each of size no more than n/2, In this case, quicksort runs much faster. The recurrence for the running time is then

$$T(n) = 2T(n/2) + \Theta(n) ,$$

- $T(n) = 2\,T(n/2) + n$
- Solve the above Equation and you will get

$T(n) = \mathbf{\Theta(n \log n)}$