Lecture # 8

Quick Sort

Quick Sort

- It is one of the fastest sorting algorithms known and is the method of choice in most sorting libraries.
- Quicksort is based on the divide and conquer strategy. Here is the algorithm:

Quick Sort

```
QUICKSORT( array A, int p, int r)
1  if (r > p)
2     then
3          i ← a random index from [p..r]
4          swap A[i] with A[p]
5          q ← PARTITION(A, p, r)
6          QUICKSORT(A, p, q - 1)
7          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

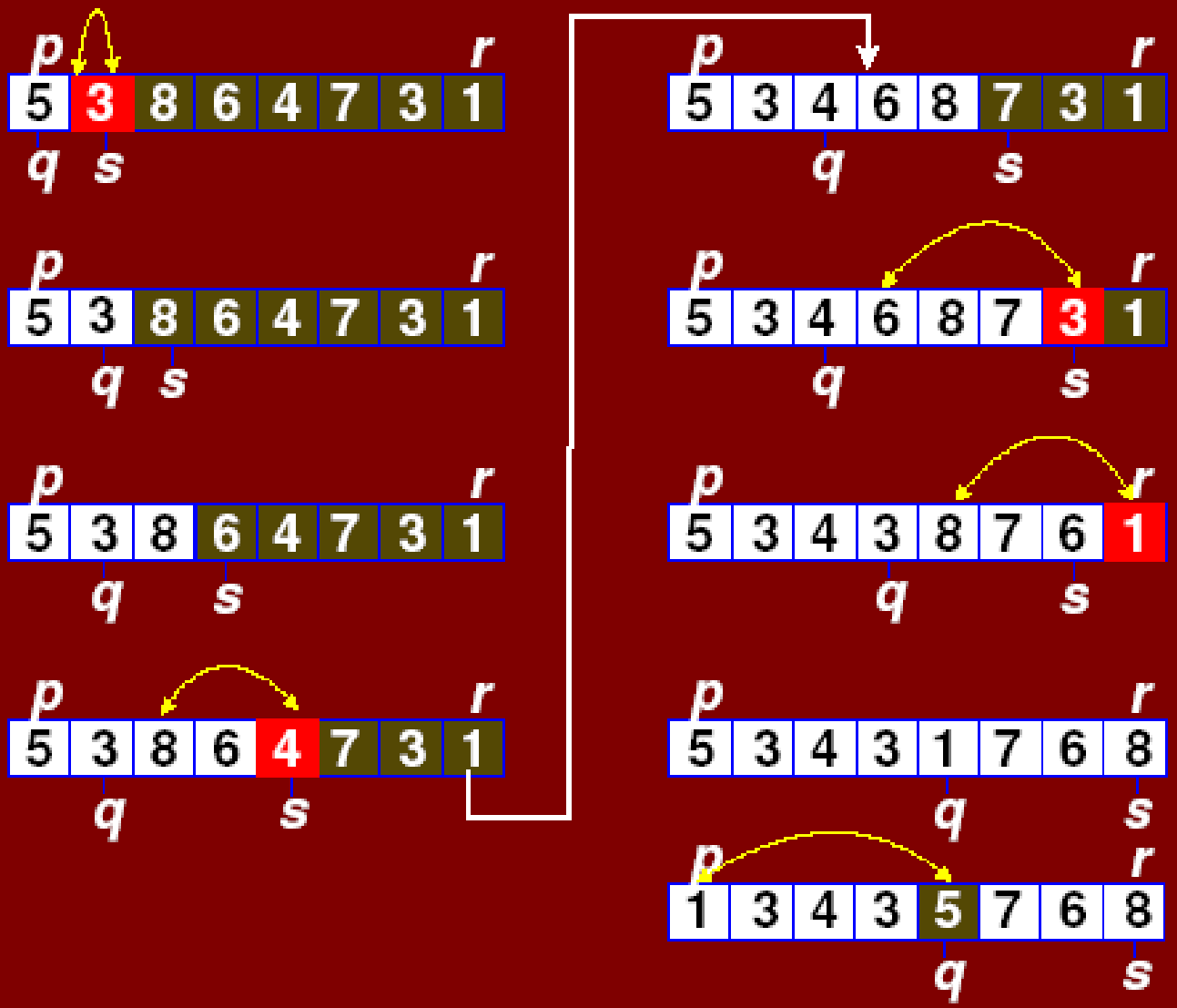
Partition Algorithm

- The algorithm works by maintaining the following *invariant condition*.
- A[p] = x is the pivot value.
- A[p..q - 1] contains elements that are less than x,
- A[q + 1..s - 1] contains elements that are greater than or equal to x.
- A[s..r] contains elements whose values are currently unknown.

Partition Algorithm

```
PARTITION( array A, int p, int r)
1       x ← A[p]
2       q ← p
3       for s ← p + 1 to r
4               do if (A[s] < x)
5                       then  q ← q + 1
6                             swap A[q] with A[s]
7       swap A[p] with A[q]
8       return q
```

```
PARTITION(array A,int p,int r)
1   x ← A[p]
2   q ← p
3   for s ← p + 1 to r
4     do if (A[s] < x)
5        then  q ← q + 1
6              swap A[q] with A[s]
7   swap A[p] with A[q]
8   return q
```



Quick Sort Example

- 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

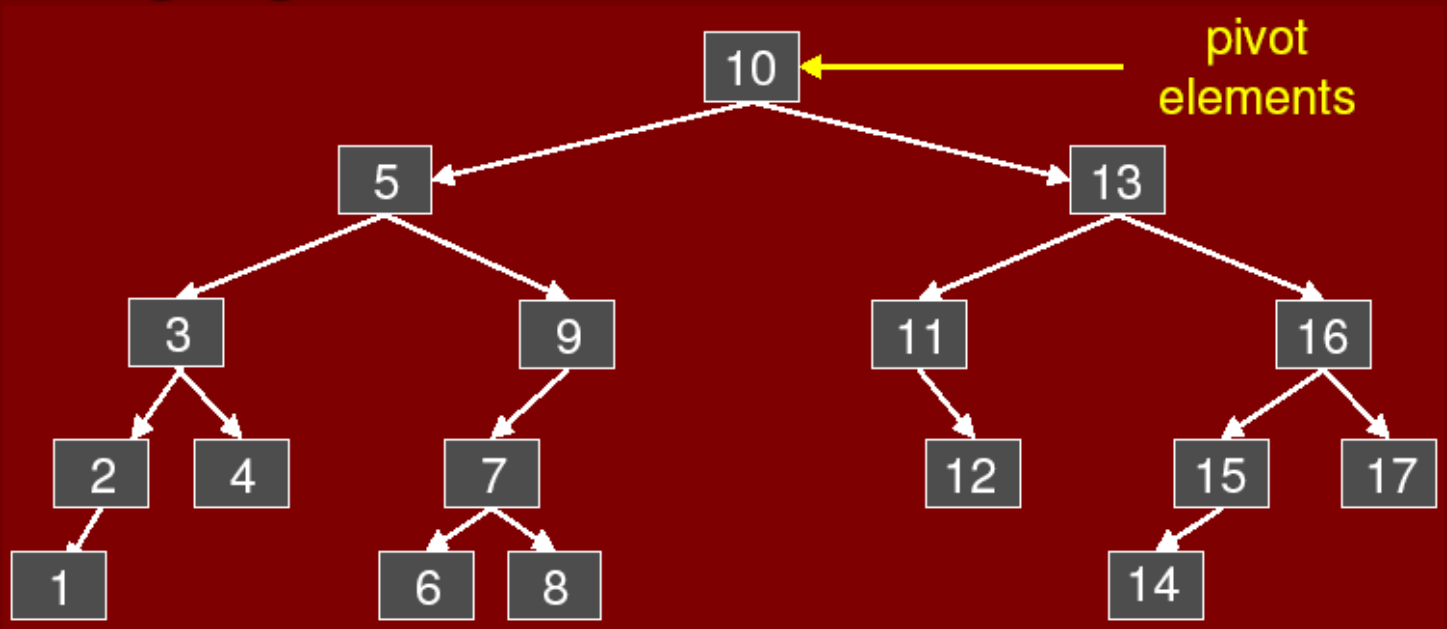
7	6	12	3	11	8	7	1	15	13	17	5	16	14	9	4	10
7	6	4	3	9	8	2	1	5	10	17	15	16	14	11	12	13
1	2	4	3	5	8	6	7	9	10	12	11	13	14	15	17	16

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 (but not surprising) 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.

- It takes $\Theta(n)$ time to do the partitioning and other overhead, and
- we make two recursive calls. The first is to the subarray A[1 : q - 1] which has q - 1 elements, and the other is to the subarray A[q + 1 : n] which has n - q elements. we get the recurrence:

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

- This depends on the value of q. To get the worst case, we maximize over all possible values of q. Putting is together, we get the recurrence

$$T(n) = \begin{cases} 1 & \text{if } n \leq 1 \\ \max\limits_{1 \leq q \leq n} (T(q-1) + T(n-q) + n) & \text{otherwise} \end{cases}$$

- Recurrences that have max's and min's embedded in them are very messy to solve. The key is determining which value of q gives the maximum. (A rule of thumb of algorithm analysis is that the worst cases tends to happen either at the extremes or in the middle. So we would plug in the value q = 1, q = n, and q = n/2 and work each out.)

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

$$
\begin{aligned}
T(n) &\le T(0) + T(n-1) + n \\
&= 1 + T(n-1) + n \\
&= T(n-1) + (n+1) \\
&= T(n-2) + n + (n+1) \\
&= T(n-3) + (n-1) + n + (n+1) \\
&= T(n-4) + (n-2) + (n-1) + n + (n+1)
\end{aligned}
$$

$$
= T(n-k) + \sum_{i=-1}^{k-2} (n-i)
$$

- For the basis T(1) = 1 we set k = n - 1 and get

$$T(n) \leq T(1) + \sum_{i=-1}^{n-3} (n - i)$$

= 1 + (3 + 4 + 5 + · · · + (n - 1) + n + (n + 1))

≤ 1 + **2** + 3 + 4 + 5 + · · · + (n - 1) + n + (n + 1))

$$\leq \sum_{i=1}^{n+1} i = \frac{(n+1)(n+2)}{2} \in \Theta(n^2)$$

$$1 + \mathbf{2} + 3 + 4 + 5 + \cdots + (n-1) + n + (n+1)) = \sum_{k=1}^{n+1} k$$

Average-case Analysis of Quick Sort

- We will now show that in the average case, quicksort runs in $\Theta(n \log n)$ time.

- Average-case in the case of quicksort, only depends upon the random choices of pivots that the algorithm makes.

Average-case Analysis of Quick Sort

- The algorithm has **n** random choices for the pivot element, and each choice has an equal probability of **1/n** of occurring. So we can modify the above recurrence to compute an average rather than a max, giving:

$$T(n) = \begin{cases} 1 & \text{if } n \leq 1 \\ \frac{1}{n}\sum_{q=1}^{n}(T(q-1) + T(n-q) + n) & \text{otherwise} \end{cases}$$

- The time T(n) is the weighted sum of the times taken for various choices of **q**. I.e.,
- $T(n) = [\ 1/n\ (\ T(0) + T(n - 1) + n\ ) +$

$1/n\ (\ T(1) + T(n - 2) + n\ ) +$

$1/n\ (\ T(2) + T(n - 3\ ) + n\ ) + \ldots\ldots$

$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$

$\ldots\ldots + 1/n\ (T(n - 1) + T(0) + n)\ ]$

- We have not seen such a recurrence before. To solve it, expansion is possible but it is rather tricky.
- We will attempt a constructive induction to solve it.
- We know that we want $\Theta(n \log n)$. Let us assume that $T(n) \leq cn \log n$ for $n \geq 2$ where c is constant.
- For the base case **n = 2** we have

$$
\begin{aligned}
T(n) &= \frac{1}{2} \sum_{q=1}^{2} (T(q-1) + T(2-q) + 2) \\
&= \frac{1}{2} \left[(T(0) + T(1) + 2) + (T(1) + T(0) + 2) \right] \\
&= \frac{8}{2} = 4.
\end{aligned}
$$

- We want this to be at most **c.2 log 2**, i.e.,

  $T(2) \leq c2 \log 2$

  Or

  $4 \leq c2 \log 2$

  therefore

  $c \geq 4/(2 \log 2) \approx 2.88.$

- For the induction step, we assume that $n \geq 3$ and The induction hypothesis is that for any $n' < n$, we have $T(n') \leq c\, n' \log n'$. We want to prove that it is true for T(n).
- By expanding T(n) and moving the factor of n outside the sum we have

$$T(n) = \frac{1}{n}\sum_{q=1}^{n}(T(q-1) + T(n-q) + n)$$

$$= \frac{1}{n}\sum_{q=1}^{n}(T(q-1) + T(n-q)) + n$$

$$= \frac{1}{n}\sum_{q=1}^{n} T(q-1) + \frac{1}{n}\sum_{q=1}^{n} T(n-q) + n$$

$$T(n) = \frac{1}{n}\sum_{q=1}^{n} T(q-1) + \frac{1}{n}\sum_{q=1}^{n} T(n-q) + n$$

- Observe that the two sums add up the same values. One counts up and other counts down i.e. generating: T(0) + T(1) + T(2) +……. T(n-1)

- Thus we can replace them with: **2.** $\sum_{q=0}^{n-1} T(q)$
- We will extract T(0) and T(1) and treat them specially.

$$T(n) = \frac{2}{n}\left(\sum_{q=0}^{n-1} T(q)\right) + n$$

$$= \frac{2}{n}\left(T(0) + T(1) + \sum_{q=2}^{n-1} T(q)\right) + n$$

- We will apply the induction hypothesis for q < n we have

$$T(n) = \frac{2}{n}\left(T(0) + T(1) + \sum_{q=2}^{n-1} T(q)\right) + n$$

$$\leq \frac{2}{n}\left(1 + 1 + \sum_{q=2}^{n-1} (c\,q \log q)\right) + n$$

$$= \frac{2c}{n}\left(\sum_{q=2}^{n-1} (\ q \ln q)\right) + n + \frac{4}{n}$$

- We have never seen this sum before:

$$S(n) = \sum_{q=2}^{n-1} (c\,q \ln q)$$

- Recall from calculus that for any monotonically increasing function **f(x)**:

$$\sum_{i=a}^{b-1} f(i) \leq \int_{a}^{b} f(x)\,dx$$

- The function **f(x) = x ln x** is monotonically increasing, and so

$$S(n) = \sum_{q=2}^{n-1} (cq \ln q) \leq \int_{2}^{n} x \ln x \, dx$$

Monotonic functions are functions that tend to move in only one direction as x increases. A **monotonic increasing** function always increases as x increases, i.e. f(a)>f(b) for all a>b.

- We can integrate this by parts:

$$\int_2^n x \ln x \, dx = \frac{x^2}{2} \ln x - \frac{x^2}{4} \Big|_{x=2}^{n}$$

$$\begin{aligned}
\int_2^n x \ln x \, dx &= \frac{x^2}{2} \ln x - \frac{x^2}{4} \Big|_{x=2}^{n} \\
&= \left(\frac{n^2}{2} \ln n - \frac{n^2}{4} \right) - (2 \ln 2 - 1) \\
&\leq \frac{n^2}{2} \ln n - \frac{n^2}{4}
\end{aligned}$$

- We thus have

$$S(n) = \sum_{q=2}^{n-1} (cq \ln q) \leq \frac{n^2}{2} \ln n - \frac{n^2}{4}$$

- Plug this back into the expression for T(n) to get

$$T(n) = 2c/n\,(n^2/2 \ln n - n^2/4) + n + 4/n$$

$$T(n) = 2c/n\,(n^2/2 \ln n - n^2/4) + n + 4/n$$

$$= cn \ln n - cn/2 + n + 4/n$$

$$= cn \ln n + n(1 - c/2) + 4/n$$

$$T(n) = cn \ln n + n(1 - c/2) + 4/n$$

- To finish the proof, we want all of this to be at most $cn \ln n$. For this to happen, we will need to select **c** such that:

$$n(1 - c/2) + 4/n \leq 0$$

- If we select $c = 3$, and use the fact that $n \geq 3$ we get

- $n(1 - c/2) + 4/n = n(1-3/2) + 4/n$

$= n(-1/2) + 4/n$

$= 4/n - n/2$

Putting n =3 as well as was base for next inductive step:

$= 4/n - n/2 \leq 4/3 - 3/2$

$= -1/6 \leq 0$

- From the basis case we had $c \geq 2.88$. Choosing $c = 3$ satisfies all the constraints. Thus

$T(n) = 3n\ln n \in \mathbf{\Theta(n \log n)}$