Chapter 7
Quicksort

Chi-Yeh Chen
陳奇業
成功大學資訊工程學系

Overview

- Quicksort
  - Worst-case running time: $\Theta(n^2)$
  - Expected running time: $\Theta(n \lg n)$
  - Constants hidden in $\Theta(n \lg n)$ are small
  - Sorts in place

Description of Quicksort

- Quicksort is based on the three-step process of divide-and-conquer.
  - To sort the subarray $A[p \ldots r]$:
    - Divide: Partition $A[p \ldots r]$, into two (possibly empty) subarrays $A[p \ldots q-1]$ and $A[q+1 \ldots r]$, such that each element in the first subarray $A[p \ldots q-1]$ is $\leq A[q]$ and $A[q]$ is less than each element in the second subarray $A[q+1 \ldots r]$.
    - Conquer: Sort the two subarrays by recursive calls to QUICKSORT.
    - Combine: No work is needed to combine the subarrays, because they are sorted in place.
  - Perform the divide step by a procedure PARTITION, which returns the index q that marks the position separating the subarrays.

- Initial call is QUICKSORT(A, 1, n)

```
QUICKSORT(A, p, r)
if p < r then
    q ← PARTITION(A, p, r)
    QUICKSORT(A, p, q − 1)
    QUICKSORT(A, q + 1, r)
```

- Partitioning
  - Partition subarray $A[p \dots r]$ by the following procedure:

PARTITION(A, p, r)

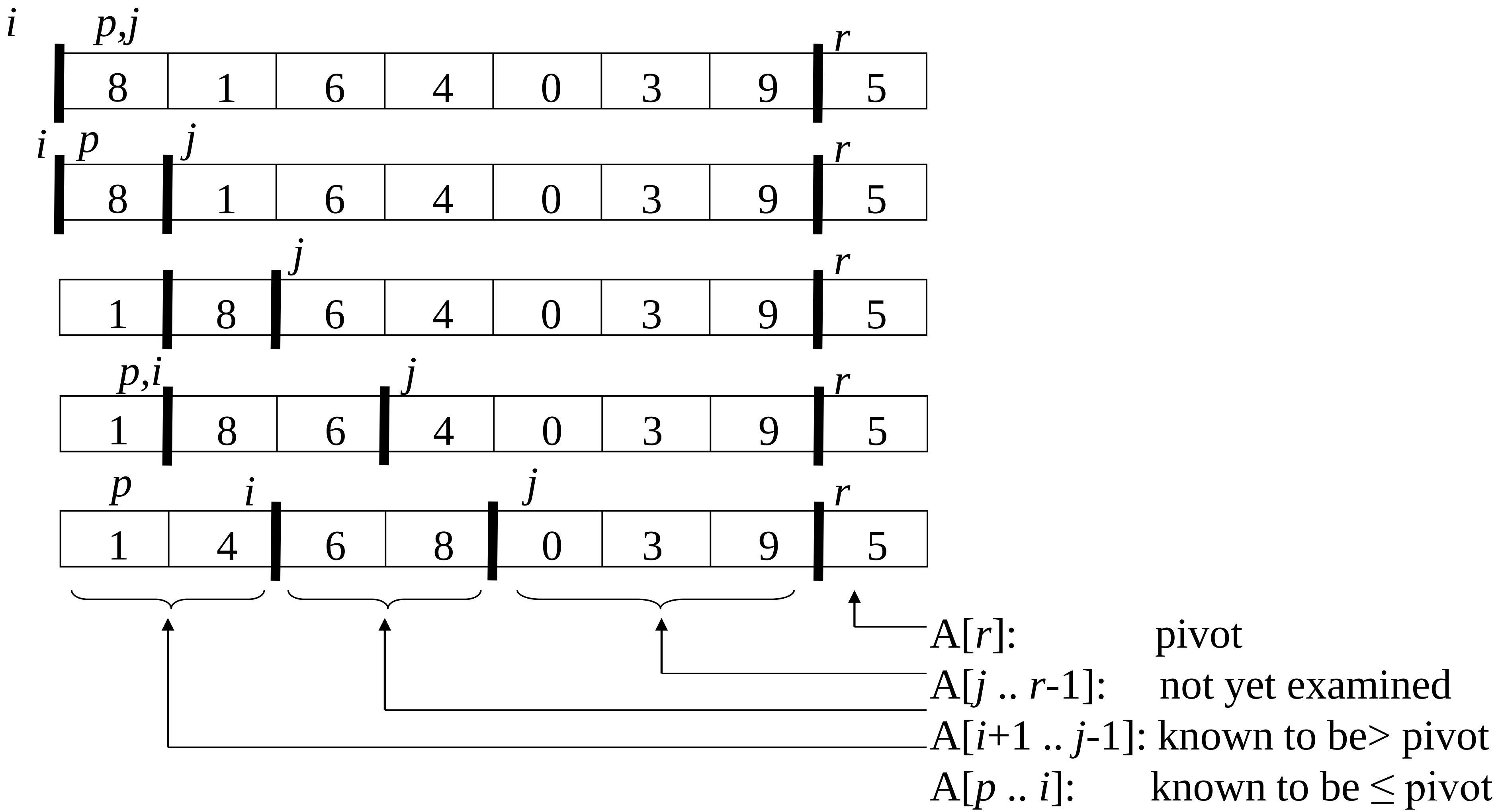
```
x ← A[r]
i ← p − 1
for j = p to r − 1 do
    if A[j] < x then
        i ← i + 1
        exchange A[i] ↔ A[j]
exchange A[i + 1] ↔ A[r]
return i + 1
```

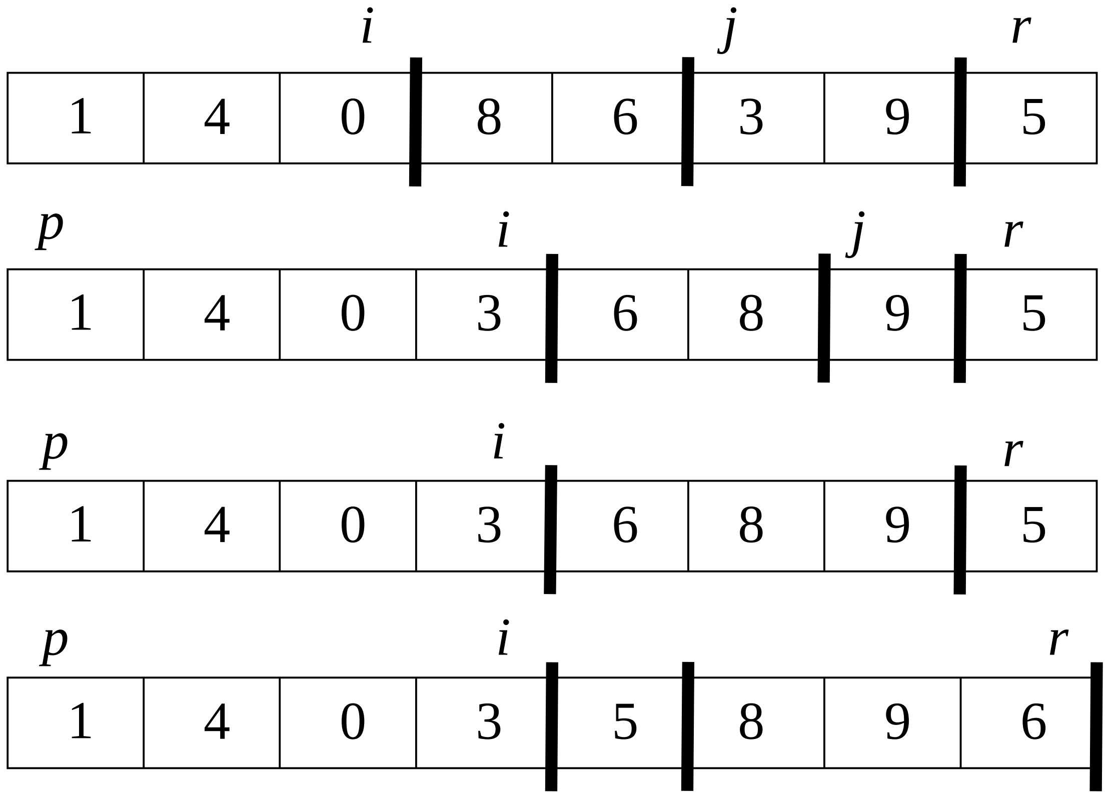
- PARTITION always selects the last element $A[r]$ in the subarray $A[p \dots r]$ as the pivot—the element around which to partition.

- As the procedure executes, the array is partitioned into four regions, some of which may be empty:
- **Loop invariant**:
  - All entries in $A[p \dots i]$ are $\leq pivot$.
  - All entries in $A[i+1 \dots j-1]$ are $> pivot$.
  - $A[r] = pivot$.

- It’ s not needed as part of the loop invariant, but the fourth region is $A[j \dots r-1]$, whose entries have not yet been examined, and so we don’ t know how they compare to the pivot.

- Example: On an 8-element subarray.





- [The index *j* disappears because it is no longer needed once the for loop is exited.]

- Correctness: Use the loop invariant to prove correctness of PARTITION.
  - Initialization: Before the loop starts, all the conditions of the loop invariant are satisfied, because r is the pivot and the subarrays $A[p \dots i]$ and $A[i+1 \dots j-1]$ are empty.
  - Maintenance: While the loop is running, if $A[j] \leq pivot$, then $A[j]$ and $A[i+1]$ are swapped and then i and j are incremented. If $A[j] > pivot$, then increment only j.
  - Termination: When the loop terminates, $j = r$, so all elements in A are partitioned into one of the three cases: $A[p \dots i] \leq pivot$, $A[i+1 \dots r-1] > pivot$, and $A[r] = pivot$.

- The last two lines of PARTITION move the pivot element from the end of the array to between the two subarrays. This is done by swapping the pivot and the first element of the second subarray, i.e., by swapping $A[i+1]$ and $A[r]$.
- Time for partitioning: $\Theta(n)$ to partition an n-element subarray.

Performance of Quicksort

- The running time of quicksort depends on the partitioning of the subarrays:
  - If the subarrays are balanced, then quicksort can run as fast as mergesort.
  - If they are unbalanced, then quicksort can run as slowly as insertion sort.
- **Worst case**
  - Occurs when the subarrays are completely unbalanced.
  - Have 0 elements in one subarray and $n-1$ elements in the other subarray.
  - Get the recurrence
    $$T(n) = T(n-1) + T(0) + \Theta(n) = T(n-1) + \Theta(n) = \Theta(n^2)$$
  - Same running time as insertion sort.
  - In fact, the worst-case running time occurs when quicksort takes a sorted array as input, but insertion sort runs in $O(n)$ time in this case.

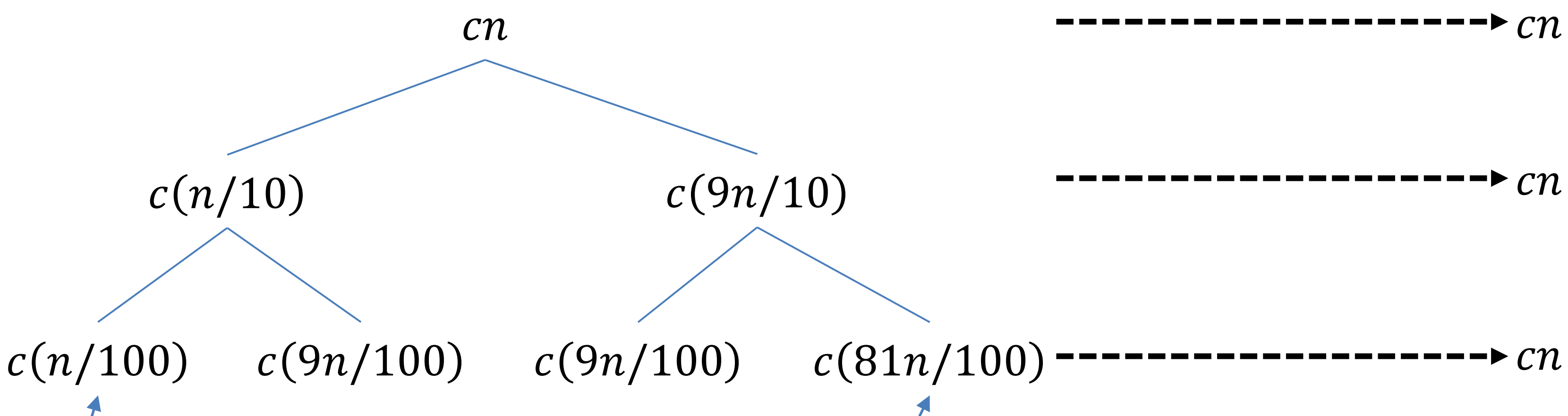
- The running time of QUICKSORT is $\Theta(n^2)$ when the array A contains distinct elements and is sorted in decreasing order.
  - PARTITION does a "worst-case partitioning" when the elements are in decreasing order. It reduces the size of the subarray under consideration by only 1 at each step, which we've seen has running time $\Theta(n^2)$.
  - In particular, PARTITION, given a subarray $A[p \ldots r]$ of distinct elements in decreasing order, produces an empty partition in $A[p \ldots q-1]$ puts the pivot (originally in $A[r]$) into $A[p]$, and produces a partition $A[p+1 \ldots r]$ with only one fewer element than $A[p \ldots r]$. The recurrence for QUICKSORT becomes $T(n) = T(n-1) + \Theta(n) = \Theta(n^2)$.

- **Best case**
  - Occurs when the subarrays are completely balanced every time.
  - Each subarray has $\leq n/2$ elements.
  - Get the recurrence

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

- **Balanced case**
  - Quicksort's average running time is much closer to the best case than to the worst case.
  - Imagine that PARTITION always produces a 9-to-1 split.
  - Get the recurrence

$$T(n) \leq T\left(\frac{9n}{10}\right) + T\left(\frac{n}{10}\right) + \Theta(n) = O(n \lg n)$$

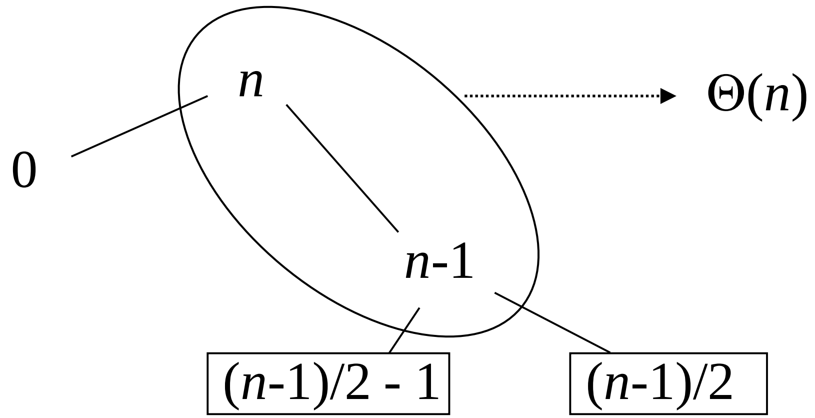


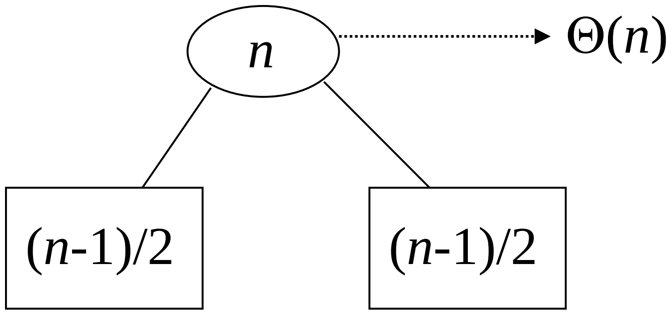
leftmost branch peters out after $\log_{10} n$ levels

rightmost branch peters out after $\log_{10/9} n$ levels

- Intuition: look at the recursion tree.
  - It's like the one for $T(n) = T(n/3) + T(2n/3) + O(n)$ in Section 4.2.
  - Except that here the constants are different; we get $\log_{10} n$ full levels and $\log_{10/9} n$ levels that are nonempty.
  - As long as it's a constant, the base of the log doesn't matter in asymptotic notation.
  - Any split of constant proportionality will yield a recursion tree of depth $\Theta(\lg n)$.

- **Intuition for the average case**
  - Splits in the recursion tree will not always be constant.
  - There will usually be a mix of good and bad splits throughout the recursion tree.
  - To see that this doesn't affect the asymptotic running time of quicksort, assume that levels alternate between best-case and worst-case splits.





- The extra level in the left-hand figure only adds to the constant hidden in the Θ-notation.
- There are still the same number of subarrays to sort, and only twice as much work was done to get to that point.
- Both figures result in $O(n \lg n)$ time, though the constant for the figure on the left is higher than that of the figure on the right.

- Suppose that the splits at every level of Quicksort are in the proportion $1 - \alpha$ to α, where $0 < \alpha \leq 1/2$ is a constant.
  - The minimum depth of a leaf in the recursion tree is approximately $-\lg n / \lg \alpha$ and
  - The maximum depth is approximately $-\lg n / \lg(1 - \alpha)$.

(if don' t worry about integer round-off)

- The minimum depth follows a path that always takes the smaller part of the partition— i.e., that multiplies the number of elements by α.
  - One iteration reduces the number of elements from n to αn, and i iterations reduces the number of elements to $\alpha^i n$.
  - At a leaf, there is just one remaining element, and so at a minimum-depth leaf of depth m, we have $\alpha^m n = 1$. Thus, $\alpha^m = 1/n$. Taking logs, we get $m \lg \alpha = -\lg n$, or $m = -\lg n / \lg \alpha$.

- Similarly, maximum depth corresponds to always taking the larger part of the partition, i.e., keeping a fraction $1-\alpha$ of the elements each time.
  - The maximum depth M is reached when there is one element left, that is, when $(1-\alpha)^M n = 1$.
  - Thus, $M = -\lg n \,/\, \lg(1-\alpha)$.

Randomized version of Quicksort

- We have assumed that all input permutations are equally likely.
- This is not always true.
- To correct this, we add randomization to quicksort.
- We could randomly permute the input array.
- Instead, we use **random sampling**, or picking one element at random.
- Don't always use $A[r]$ as the $pivot$. Instead, randomly pick an element from the subarray that is being sorted.

- We add this randomization by not always using $A[r]$ as the pivot, but instead randomly picking an element from the subarray that is being sorted.

- Randomly selecting the pivot element will, on average, cause the split of the input array to be reasonably well balanced.

RANDOMIZED-PARTITION(A, p, r)

1 $i \leftarrow$ RANDOM(p, r)
2 exchange $A[r] \leftrightarrow A[i]$
3 **return** PARTITION(A, p, r)

- Randomization of quicksort stops any specific type of array from causing worst-case behavior.
- For example, an already-sorted array causes worst-case behavior in non-randomized QUICKSORT, but not in RANDOMIZED QUICKSORT.

RANDOMIZED-QUICKSORT(A, p, r)

```
if p < r then
    q ← RANDOMIZED-PARTITION(A, p, r)
    RANDOMIZED-QUICKSORT(A, p, q − 1)
    RANDOMIZED-QUICKSORT(A, q + 1, r)
```

Analysis of Quicksort

- We will analyze
  - The worst-case running time of QUICKSORT and RANDOMIZED-QUICKSORT (the same), and
  - The expected (average-case) running time of RANDOMIZED QUICKSORT.

- **Worst-case analysis**
  We will prove that a worst-case split at every level produces a worst-case running time of $O(n^2)$.
  - Recurrence for the worst-case running time of QUICKSORT:
    $$T(n) = \max_{0 \le q \le n-1} \big(T(q) + T(n-q-1)\big) + \Theta(n)$$
  - Because PARTITION produces two subproblems, totaling size $n - 1$, q ranges from 0 to $n - 1$.
  - **Guess**: $T(n) \le cn^2$, for some c.
  - Substituting our guess into the above recurrence:
    $$T(n) \le \max_{0 \le q \le n-1} (cq^2 + c(n-q-1)^2) + \Theta(n)$$
    $$= c \max_{0 \le q \le n-1} (q^2 + (n-q-1)^2) + \Theta(n)$$

- $f(q) = q^2 + (n - q - 1)^2$
- $f'(q) = 2q - 2(n - q - 1) = 4q - 2n + 2$
- $f''(q) = 2 + 2 = 4$
- $f'(q) = 0$ when $q = \frac{1}{2}n - \frac{1}{2}$. $f'(q)$ is also continuous. $\forall q{:}\, f''(q) > 0$ which means that $f'(q)$ is negative left of $f'(q) = 0$ and positive right of it, which means that this is a local minima. In this case, $f(q)$ is decreasing in the beginning of the interval and increasing in the end, which means that those two points are the only candidates for a maximum in the interval.

$$f(0) = 0^2 + (n - 1)^2$$
$$f(n - 1) = (n - 1)^2 + 0^2$$

- The maximum value of $(q^2 + (n - q - 1)^2)$ occurs when q is either 0 or n – 1. (Second derivative with respect to q is positive.) This means that

$$(q^2 + (n - q - 1)^2) \leq (n - 1)^2 = n^2 - 2n + 1$$

- Therefore,

$$\begin{aligned} T(n) &\leq cn^2 - c(2n - 1) + \Theta(n) \\ &\leq cn^2 \quad \text{if } c(2n - 1) \geq \Theta(n) \end{aligned}$$

  Pick c so that $c(2n$ – $1)$ dominates $\Theta(n)$.

- Therefore, the worst-case running time of quicksort is $O(n^2)$.
- We can also show that the recurrence' s solution is $\Omega(n^2)$. Thus, the worst-case running time is $\Theta(n^2)$

- **Guess:** $T(n) \geq cn^2 - dn$, for some c:

$$T(n) = \max_{0 \leq q \leq n-1} \big(T(q) + T(n-q-1)\big) + \Theta(n)$$
$$\geq \max_{0 \leq q \leq n-1} (cq^2 - dq + c(n-q-1)^2 - dn + dq + d)$$
$$+ \Theta(n)$$
$$\geq c \max_{0 \leq q \leq n-1} (q^2 + (n-q-1)^2) - d(n-1) + \Theta(n)$$
$$= cn^2 - c(2n-1) - d(n-1) + \Theta(n)$$
$$= cn^2 - dn - c(2n-1) + d + \Theta(n)$$
$$\geq cn^2 - dn \quad \text{if} - c(2n-1) + d + \Theta(n) \geq 0$$

- **Guess**: $T(n) \geq cn^2$, for some c:

$$T(n) = \max_{0 \leq q \leq n-1} \left(T(q) + T(n-q-1) \right) + \Theta(n)$$
$$\geq \max_{0 \leq q \leq n-1} (cq^2 + c(n-q-1)^2) + \Theta(n)$$
$$\geq c \max_{0 \leq q \leq n-1} (q^2 + (n-q-1)^2) + \Theta(n)$$
$$= cn^2 - c(2n-1) + \Theta(n)$$
$$\geq cn^2 \quad \text{if} - c(2n-1) + \Theta(n) \geq 0$$

- **Average-case analysis**
  - The dominant cost of the algorithm is partitioning.
  - PARTITION removes the pivot element from future consideration each time.
  - Thus, PARTITION is called at most n times.
  - QUICKSORT recurses on the partitions.
  - The amount of work that each call to PARTITION does is a constant plus the number of comparisons that are performed in its **for** loop.
  - Let X = the total number of comparisons performed in all calls to PARTITION.
  - Therefore, the total work done over the entire execution is $O(n + X)$.

PARTITION(A, p, r)

$x \leftarrow A[r]$
$i \leftarrow p - 1$
for $j = p$ ***to*** $r - 1$ **do**
  if $A[j] < x$ **then**
    $i \leftarrow i + 1$
    exchange $A[i] \leftrightarrow A[j]$
exchange $A[i + 1] \leftrightarrow A[r]$
return $i + 1$

- We will now compute a bound on the overall number of comparisons.
- For ease of analysis:
  - Rename the element of A as $z_1, z_2, \dots, z_n$, with z_i being the ith smallest element.
  - Define the set $Z_{ij} = \{z_i, z_{i+1}, \dots, z_j\}$ to be the set of elements between z_i and z_j, inclusive.

- Each pair of elements is compared at most once, because elements are compared only to the pivot element, and then the pivot element is never in any later call to PARTITION.
- Let $X_{ij} = \mathrm{I}\{z_i \text{ is compared to } z_j\}$. (Considering whether z_i is compared to z_j at any time during the entire quicksort algorithm, not just during one call of PARTITION.)
- Since each pair is compared at most once, the total number of comparisons performed by the algorithm is

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}$$

- Lemma 5.1
  Given a sample space S and an even A in the sample space S, let $X_A = \mathrm{I}\{A\}$. Then $\mathrm{E}[X_A] = \Pr\{A\}$.
  Proof

$$\mathrm{E}[X_A] = \mathrm{E}[\mathrm{I}\{A\}] = 1 \cdot \Pr\{A\} + 0 \cdot \Pr\{\bar{A}\} = \Pr\{A\}$$

- Take expectations of both sides, use Lemma 5.1 and linearity of expectation:

$$E[X] = E\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}\right] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E\left[X_{ij}\right]$$

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{z_i \text{ is compared to } z_j\}$$

Now all we have to do is find the probability that two elements are compared.

- Think about when two elements are not compared.
  - For example, numbers in separate partitions will not be compared.
  - In the previous example, ⟨8, 1, 6, 4, 0, 3, 9, 5⟩ and the pivot is 5, so that none of the set ⟨1, 4, 0, 3⟩ will ever be compared to any of the set ⟨8, 6, 9⟩.
  - Once a pivot x is chosen such that $z_i < x < z_j$, then z_i and z_j will never be compared at any later time.
  - If either z_i or z_j is chosen before any other element of Z_{ij}, then it will be compared to all the elements of Z_{ij}, except itself.
  - The probability that z_i is compared to z_j is the probability that either z_i or z_j is the first element chosen.
  - There are $j - i + 1$ elements, and pivots are chosen randomly and independently. Thus the probability that any particular one of them is the first one chosen is $1/(j - i + 1)$.

- Therefore,

$$
\begin{aligned}
&\Pr\{z_i \text{ is compared to } z_j\} \\
&= \Pr\{z_i \text{ or } z_j \text{ is the first pivot chosen from } Z_{ij}\} \\
&= \Pr\{z_i \text{ is the first pivot chosen from } Z_{ij}\} \\
&+ \Pr\{z_j \text{ is the first pivot chosen from } Z_{ij}\} \\
&= \frac{1}{j-i+1} + \frac{1}{j-i+1} \\
&= \frac{2}{j-i+1}
\end{aligned}
$$

- [The second line follows because the two events are mutually exclusive.]

- Substituting into the equation for $E[X]$:

$$E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{z_i \text{ is compared to } z_j\}$$

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$$

- Harmonic series

  For positive integers n, the nth *harmonic number* is

$$
\begin{aligned}
H_n &= 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{n} \\
&= \sum_{k=1}^{n} \frac{1}{k} \\
&= \ln n + O(1)
\end{aligned}
\tag{A.7}
$$

- Evaluate by using a change in variables ($k = j - i$) and the bound on the harmonic series in equation (A.7):

$$E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j - i + 1} = \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} < \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k}$$

$$= \sum_{i=1}^{n-1} O(\lg n) = O(n \lg n)$$

- So the expected running time of quicksort, using RANDOMIZED-PARTITION, is $O(n \lg n)$.

- Quicksort's best-case running time is $\Omega(n \lg n)$.
  - Get the recurrence

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

Using master method, $f(n) = \Theta(n)$ and $n^{\log_b a} = n$. Since $f(n) = \Theta\left(n^{\log_b a}\right)$, case 2 applies and $T(n) = \Theta(n \lg n)$.

- Let $T(n)$ be the best-case time for the procedure QUICKSORT on an input of size n.
- We have the recurrence

$$T(n) = \min_{1 \le q \le n-1} \big(T(q) + T(n-q-1)\big) + \Theta(n)$$

- We guess that $T(n) \ge cn \lg n$ for some constant c. Substituting this guess into the recurrence, we obtain

$$\begin{aligned} &T(n) \\ &\ge \min_{1 \le q \le n-1} (cq \lg q + c(n-q-1)\lg(n-q-1)) + \Theta(n) \\ &= c \min_{1 \le q \le n-1} (q \lg q + (n-q-1)\lg(n-q-1)) + \Theta(n) \end{aligned}$$

- As we' ll show below, the expression $q \lg q + (n - q -$

- Choosing $q = (n-1)/2$ gives us the bound

$$\min_{1 \le q \le n-1} (q \lg q + (n-q-1)\lg(n-q-1))$$
$$\ge \frac{n-1}{2}\lg\frac{n-1}{2} + \frac{n-1}{2}\lg\frac{n-1}{2}$$
$$= (n-1)\lg\frac{n-1}{2}$$

- Continuing with our bounding of $T(n)$, we obtain, for $n \ge 2$,

$$T(n) \ge c(n-1)\lg\frac{n-1}{2} + \Theta(n)$$

$$
\begin{aligned}
T(n) &\geq c(n-1)\lg\frac{n-1}{2} + \Theta(n) \\
&= c(n-1)\lg(n-1) - c(n-1) + \Theta(n) \\
&= cn\lg(n-1) - c\lg(n-1) - c(n-1) + \Theta(n) \\
&\geq cn\lg\left(\frac{n}{2}\right) - c\lg(n-1) - c(n-1) + \Theta(n) \qquad \text{(Since } n \geq 2\text{)} \\
&= cn\lg n - cn - c\lg(n-1) - cn + c + \Theta(n) \\
&= cn\lg n - c(2n + \lg(n-1) - 1) + \Theta(n) \\
&\geq cn\lg n
\end{aligned}
$$

since we can pick the constant c small enough so that the $\Theta(n)$ term dominates the quantity $c(2n + \lg(n-1) - 1)$. Thus, the best-case running time of quicksort is $\Omega(n\lg n)$.

- Letting $f(q) = q \lg q + (n - q - 1) \lg(n - q - 1)$, we now show how to find the minimum value of this function in the range $1 \leq q \leq n - 1$. We need to find the value of q for which the derivative of f with respect to q is 0. We rewrite this function as

$$f(q) = \frac{q \ln q + (n - q - 1) \ln(n - q - 1)}{\ln 2}$$

and so

- The derivative $f(q)$ is 0 when $q = n - q - 1$, or when $q = (n-1)/2$. To verify that $q = (n-1)/2$ is indeed a minimum (not a maximum or an inflection point), we need to check that the second derivative of f is positive at $q = (n-1)/2$:

$$f''(q) = \frac{d}{dq}\left(\frac{\ln q - \ln(n-q-1)}{\ln 2}\right)$$

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

$$f''\left(\frac{n-1}{2}\right) = \frac{1}{\ln 2}\left(\frac{2}{n-1} + \frac{2}{n-1}\right) = \frac{1}{\ln 2} \cdot \frac{4}{n-1} > 0$$ (Since $n \geq 2$)