Quick Sort

Quicksort

- Another **Divide and Conquer** Problem
- **Divide:**
  - Partition (rearrange) the array $A[p..r]$ into two subarrays $A[p..q-1]$ and $A[q+1..r]$
  - Each element of $A[p..q-1]$ is less than or equal to $A[q]$, which is, in turn, less than or equal to each element of $A[q+1..r]$
  - Compute the index q as part of this partitioning procedure
- **Conquer:**
  - Sort the two subarrays $A[p..q-1]$ and $A[q+1..r]$ by recursive calls to Quicksort
- **Combine:**
  - No work is needed to combine the subarrays (already sorted)
  - The entire array $A[p..r]$ is now sorted

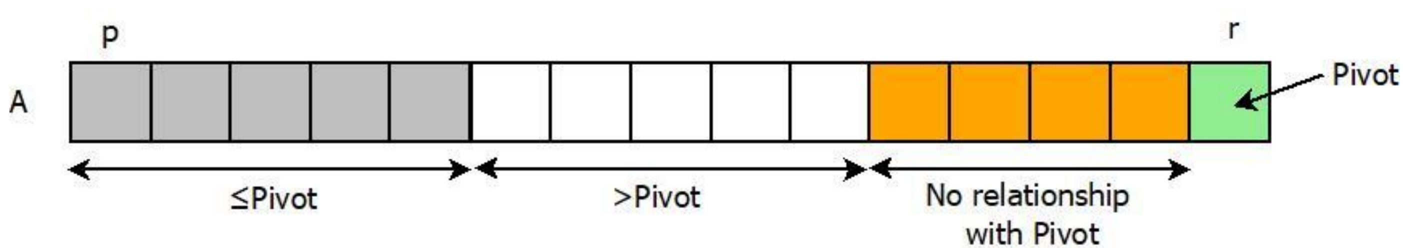
Quicksort

- Quicksort Pseudocode

  QUICKSORT(A, p, r)

  if $p < r$
      $q = \text{PARTITION}(A, p, r)$ ← Key Procedure
      QUICKSORT$(A, p, q - 1)$
      QUICKSORT$(A, q + 1, r)$

- PARTITION Procedure
  - PARTITION Procedure selects a ***pivot*** element
    - Usually ***pivot*** is the first/last element of the subarray $A[p\,..\,r]$
  - It partitions the array into four (possibly empty) regions
  - These regions satisfy certain properties → Loop Invariants



PARTITION Procedure

- Pseudocode

```
PARTITION(A, p, r)
1  x = A[r]
2  i = p - 1
3  for j = p to r - 1
4      if A[j] ≤ x
5          i = i + 1
6          exchange A[i] with A[j]
7  exchange A[i + 1] with A[r]
8  return i + 1
```

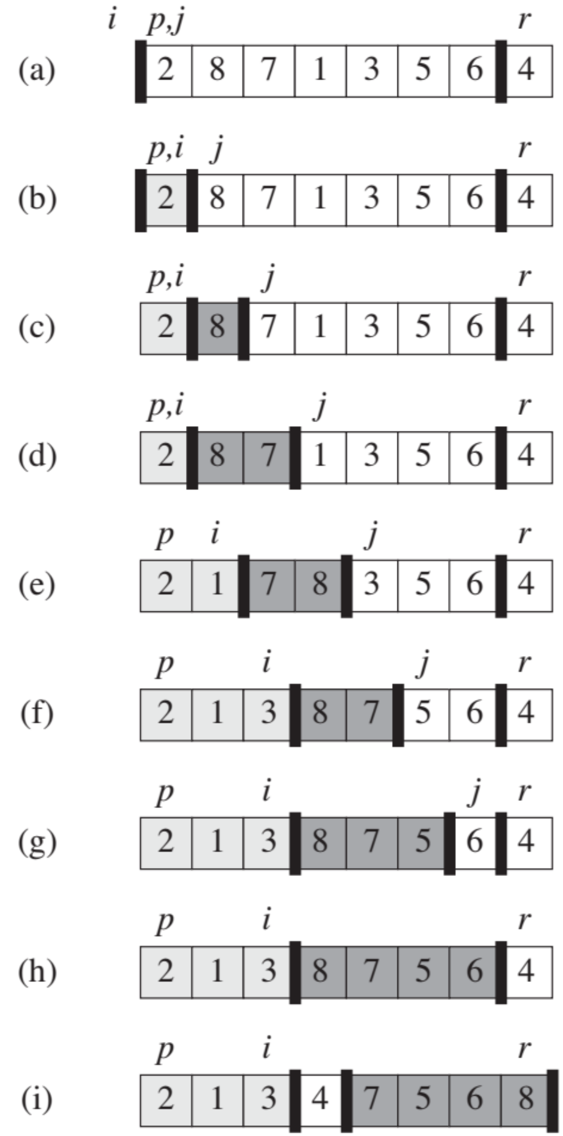
- Loop invariant: At the beginning of each iteration of the loop of lines 3–6, for any array index k
  - If $p \leq k \leq i$, then $A[k] \leq x$
  - If $i + 1 \leq k \leq j - 1$, then $A[k] > x$
  - If $k = r$, then $A[k] = x$

PARTITION Example

- Example: 2, 8, 7, 1, 3, 5, 6, 4
  - p and r are starting and end index of array A
  - Pivot $x = A[r] = 4$
  - Initially $i = p - 1 = 0$ and $j = p = 1$
  - ***for*** loop

```
for j = p to r − 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
```

  - The pivot element is swapped so that it lies between the two partitions



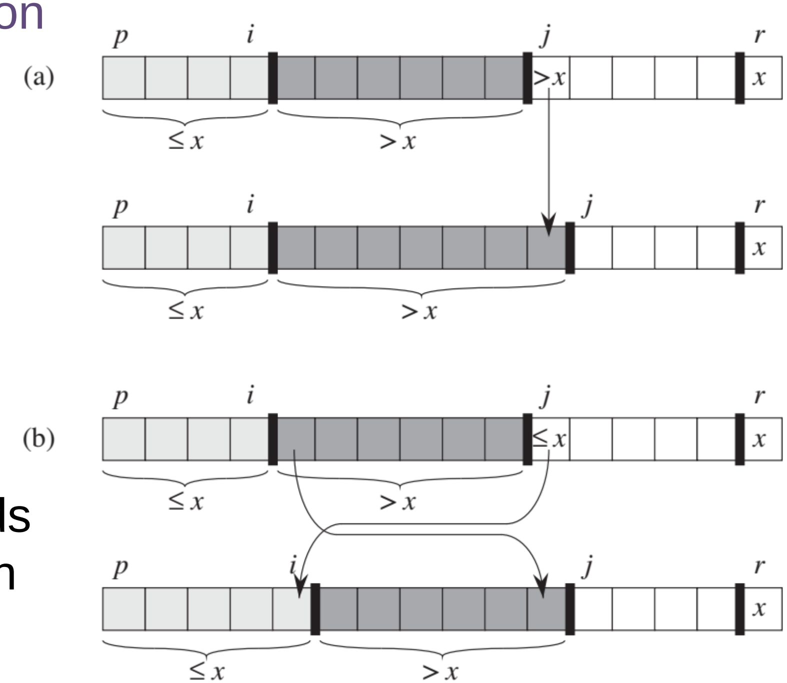
Loop Invariant

- Loop invariant: At the beginning of each iteration of the loop of lines 3–6, for any array index k
  - If $p \leq k \leq i$, then $A[k] \leq x$
  - If $i + 1 \leq k \leq j - 1$, then $A[k] > x$
  - If $k = r$, then $A[k] = x$

- **Initialization:**
  - Prior to the first iteration of the loop, $i = p - 1 = 0$ and $j = p = 1$
  - No values lie between p and i and between $i + 1$ and $j - 1$, the first two conditions of the loop invariant are trivially satisfied

Loop Invariant

- Loop invariant: At the beginning of each iteration of the loop of lines 3–6, for any array index k
  - If $p \leq k \leq i$, then $A[k] \leq x$
  - If $i+1 \leq k \leq j-1$, then $A[k] > x$
  - If $k = r$, then $A[k] = x$
- **Maintenance: Two cases**
  - If $A[j] > x$: Just increment j
    - After j is incremented, condition 2 holds for $A[j-1]$ and all other entries remain unchanged
  - If $A[j] \leq x$: Increments i
    - Swaps $A[i]$ and $A[j]$
    - Then increments j
    - Because of the swap, we now have that $A[i] \leq x$, condition 1 is satisfied
    - Similarly, we also have that $A[j-1] > x$, since the item that was swapped into $A[j-1]$ is greater than x.



Loop Invariant

- Loop invariant: At the beginning of each iteration of the loop of lines 3–6, for any array index k
  - If $p \leq k \leq i$, then $A[k] \leq x$
  - If $i + 1 \leq k \leq j - 1$, then $A[k] > x$
  - If $k = r$, then $A[k] = x$
- **Termination:**
  - At termination, $j = r$
  - Every entry in the array is in one of the three sets described by the invariant
  - Partitioned into three sets: less than or equal to x, greater than x, and a singleton set containing x
- Post Termination
  - Pivot element swapped with the leftmost element greater than x
    - Moving pivot into its **correct** place in the partitioned array

Performance of Quicksort

- PARTITION Procedure

```
PARTITION(A, p, r)
x = A[r]
i = p − 1
for j = p to r − 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

- The running time of PARTITION on the subarray $A[p..r]$ is $\Theta(n)$, where $n = r - p + 1$

Performance of Quicksort

- Quicksort Algorithm

QUICKSORT(A, p, r)

if $p < r$
  $q = \text{PARTITION}(A, p, r)$
  $\text{QUICKSORT}(A, p, q-1)$
  $\text{QUICKSORT}(A, q+1, r)$

- Quicksort running time depends on whether
  - Partitioning is balanced or unbalanced
  - Hence, on the input sequence and the choice of pivot element

Worst-Case Partitioning

- When PARTITION procedure produces
  - One subproblem with $n - 1$ elements and one with 0 elements
    - When the input array is already completely sorted
  - Unbalanced partitioning in each recursive call

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

- Using substitution method

$$T(n) = \Theta(n^2)$$

- Comparing with ***Insertion sort***
  - When the input array is already completely sorted: Best Case

$$T(n) = \Theta(n)$$

Best-Case Partitioning

- PARTITION produces two subproblems: Each of size approximately $n/2$
  - One of size $\lfloor n/2 \rfloor$ and another of size $\lceil n/2 \rceil - 1$
  - Balanced partitioning in each recursive call

$$T(n) = 2\mathrm{T}(n/2) + \Theta(n)$$

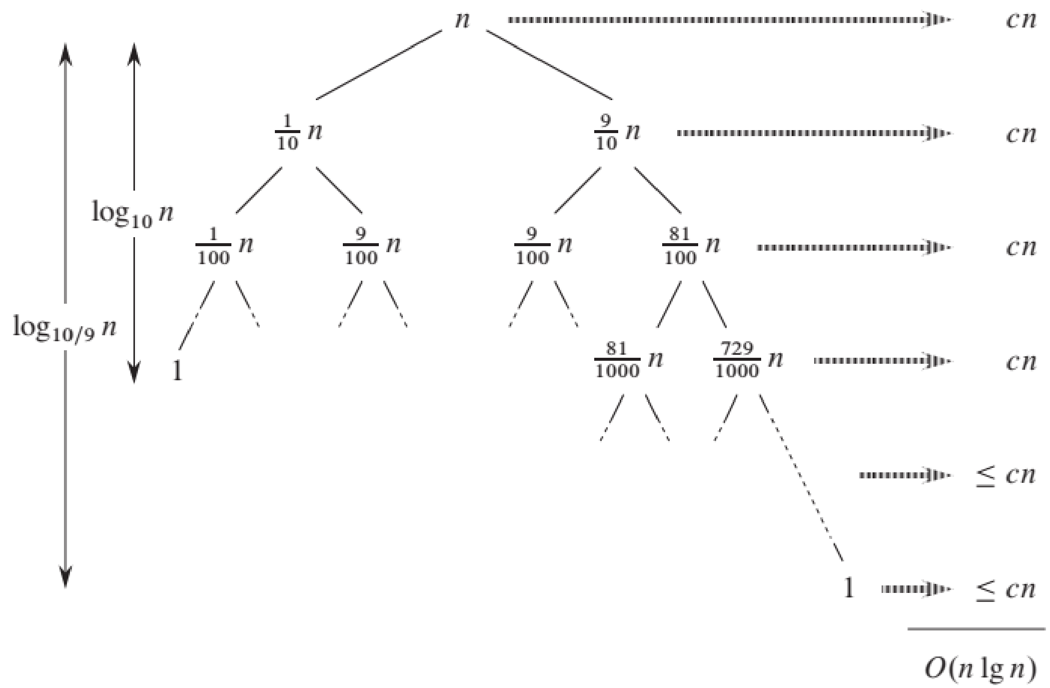
- Using Master method

$$T(n) = \Theta(n \log n)$$

Best vs. Worst Case

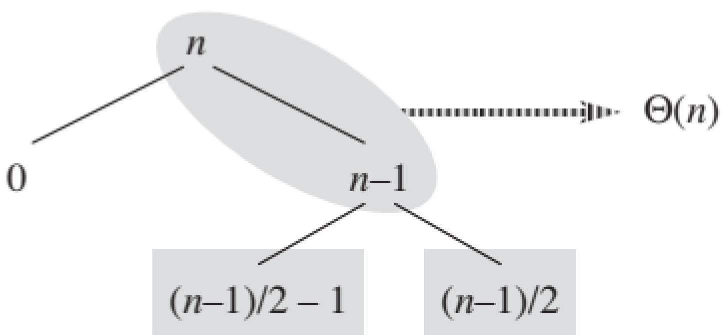
- Balanced partitioning is good $\Theta(n \log n)$
- Unbalanced partitioning is bad $\Theta(n^2)$
  - All unbalanced partitions are bad?
- Example: If PARTITION procedure always produces 9-to-1 proportional split

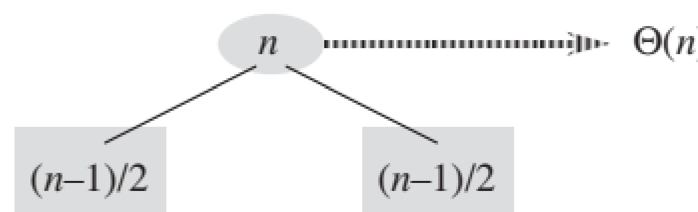
$$T(n) = \mathrm{T}(9n/10) + \mathrm{T}(\mathrm{n}/10) + \Theta(n)$$



Intuition for Average Case

- Real world scenario: Partition almost never happen in the same way **at every level**
- **In the average case,** PARTITION will produce a mix of Balanced ("good") and Unbalanced ("bad") splits





$\Theta(n) + \Theta(n-1) = \Theta(n)$

$\Theta(n)$

- Cost of bad split is absorbed into $\Theta(n)$ cost of the good split

Randomized Quick Sort

Not a part of DSA syllabus

Randomized Quicksort

- Till now, we assumed: All permutations of input number are equally likely
  - Alternate "good" and "bad" split is an acceptable assumption
- **Random Sampling**: Instead of $A[r]$, a randomly chosen element used as $Pivot$
  - Pivot element is equally likely to be any of the $r - p + 1$ elements
  - Split is expected to be reasonably balanced, on average
- Worst case may no longer be the worst case

RANDOMIZED QUICKSORT

- RANDOMIZED QUICKSORT Pseudocode

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

- RANDOMIZED PARTITION Procedure

```
RANDOMIZED-PARTITION(A, p, r)
i = RANDOM(p, r)
exchange A[r] with A[i]
return PARTITION(A, p, r)
```

Analysis of Quicksort: Worst Case

- Applies to both QUICKSORT and RANDOMIZED-QUICKSORT
- Let $T(n)$ be the worst-case runtime for the QUICKSORT

$$T(n) = \max_{0 \le q \le n-1} (T(q) + T(n-q-1)) + \Theta(n)$$

parameter q ranges from 0 to $n-1$

- Using the substitution method
  - Running time $\mathrm{T(n)} = O(n^2)$. Hence $T(n) \le cn^2$

$$\begin{aligned} T(n) &\le \max_{0 \le q \le n-1} (cq^2 + c(n-q-1)^2) + \Theta(n) \\ &= c \cdot \max_{0 \le q \le n-1} (q^2 + (n-q-1)^2) + \Theta(n) \end{aligned}$$

Analysis of Quicksort: Worst Case

– The expression $q^2 + (n - q - 1)^2$ achieves a maximum when $q = 0$ or $q = n - 1$

$$\max_{0 \le q \le n-1} (q^2 + (n - q - 1)^2) \le (n - 1)^2$$
$$= n^2 - 2n - 1$$

$$T(n) \le cn^2 - c(2n - 1) + \Theta(n)$$
$$\le cn^2$$
$$= O(n^2)$$

Expected Running Time

- QUICKSORT and RANDOMIZED-QUICKSORT
  - Differ only in **how** they select **pivot** elements
- The running time of QUICKSORT is dominated by the time spent in the **PARTITION** procedure
- Each time the PARTITION procedure is called
  - A pivot element is selected
  - This pivot element is never included in future calls of QUICKSORT and PARTITION
  - Hence, at most n calls to PARTITION can be made over the entire execution of QUICKSORT
  - Each call of PARTITION: Execution time proportional to number of iteration of the for loop
  - Each iteration of the for loop: Pivot is compared with another element

Expected Running Time

- ***Lemma:*** Let X be the number of comparisons performed in line 4 of PARTITION over the entire execution of QUICKSORT on an n-element array. Then the running time of QUICKSORT is $O(n + X)$.

- Question:

1) When the algorithm compares two elements of the array and when it does not

Renaming elements of A as $z_1, z_2, .., z_n$ such that $z_1 < z_2 < \cdots < z_n$

Defining Set $Z_{ij} = \{z_i, z_{i+1}, \ldots, z_j\}$

Expected Running Time

- When does the algorithm compare z_i and z_j?
  - Each pair of elements is compared at most once
  - Elements are compared only to the pivot element
  - Pivot element used in that call is never again compared to any other elements

$$X_{ij} = 1 \quad if\ z_i \text{ is compared to } z_j$$
$$= 0\ otherwise$$

- Since each pair is compared at most once, total number of comparisons performed

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

Expected Running Time

- Taking expectation on both the sides

$$
\begin{aligned}
\mathrm{E}[X] &= \mathrm{E}\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}\right] \\
&= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathrm{E}[X_{ij}] \\
&= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{z_i \text{ is compared to } z_j\}
\end{aligned}
$$

- Only thing left to compute $\Pr\{z_i \text{ is compared to } z_j\}$

Expected Running Time

- Pivots are chosen randomly chosen
  - The probability that any given element is the first one chosen as a pivot is $1/(j - i + 1)$

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

- Hence,

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

Expected Running Time

- Sum of harmonic series

$$
\begin{aligned}
\mathrm{E}[X] &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \\
&= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} \\
&< \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k} \\
&= \sum_{i=1}^{n-1} O(\lg n) \\
&= O(n \lg n) .
\end{aligned}
$$

- The expected running time of quicksort is $O(n \lg n)$

Exercises

7.4-1

Show that in the recurrence

$$T(n) = \max_{0 \le q \le n-1} (T(q) + T(n-q-1)) + \Theta(n) ,$$

$T(n) = \Omega(n^2)$.

7.4-2

Show that quicksort's best-case running time is $\Omega(n \lg n)$