Lectures 9-10.
Quicksort

Introduction to Algorithms
Da Nang University of Science and Technology

Dang Thien Binh
dtbinh@dut.udn.vn

Introduction of Quicksort

- Worst-case running time: $\Theta(n^2)$
- Expected running time: $\Theta(n \lg n)$
- Constants hidden in $\Theta(n \lg n)$ are small
- Another divide-and-conquer algorithm

Quicksort

- To sort the subarray $A[p \mathinner{.\,.} r]$
  - Divide
    - Partition $A[p \mathinner{.\,.} r]$, into two (possibly empty) subarrays $A[p \mathinner{.\,.} q-1]$ and $A[q+1 \mathinner{.\,.} r]$, such that each element of $A[p \mathinner{.\,.} q-1]$ is less than or equal to each element of $A[q+1 \mathinner{.\,.} r]$
  - Conquer
    - Sort the two subarrays by recursive calls to QUICKSORT
  - Combine
    - No work is needed to combine the subarrays because they are already sorted
- Perform the divide step by a procedure PARTITION, which returns the index q that marks the position separating the subarrays

Quicksort Pseudocode (1/2)

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

Quicksort Pseudocode (2/2)

```
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
```

Partition (1/2)

- Clearly, all the action takes place in the **`partition()`** function
  - Rearrange the subarray $A[p \, .. \, r]$ in place
  - End result:
    - Two subarrays
    - All values in first subarray $\leq$ all values in second one
  - Return index of the "$pivot$" element separating the two subarrays

Partition (2/2)

- PARTITION always selects the last element $A[r]$ in the subarray $A[p\,..\,r]$ 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

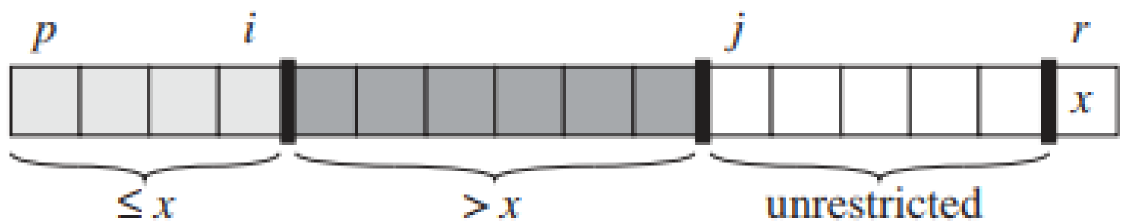
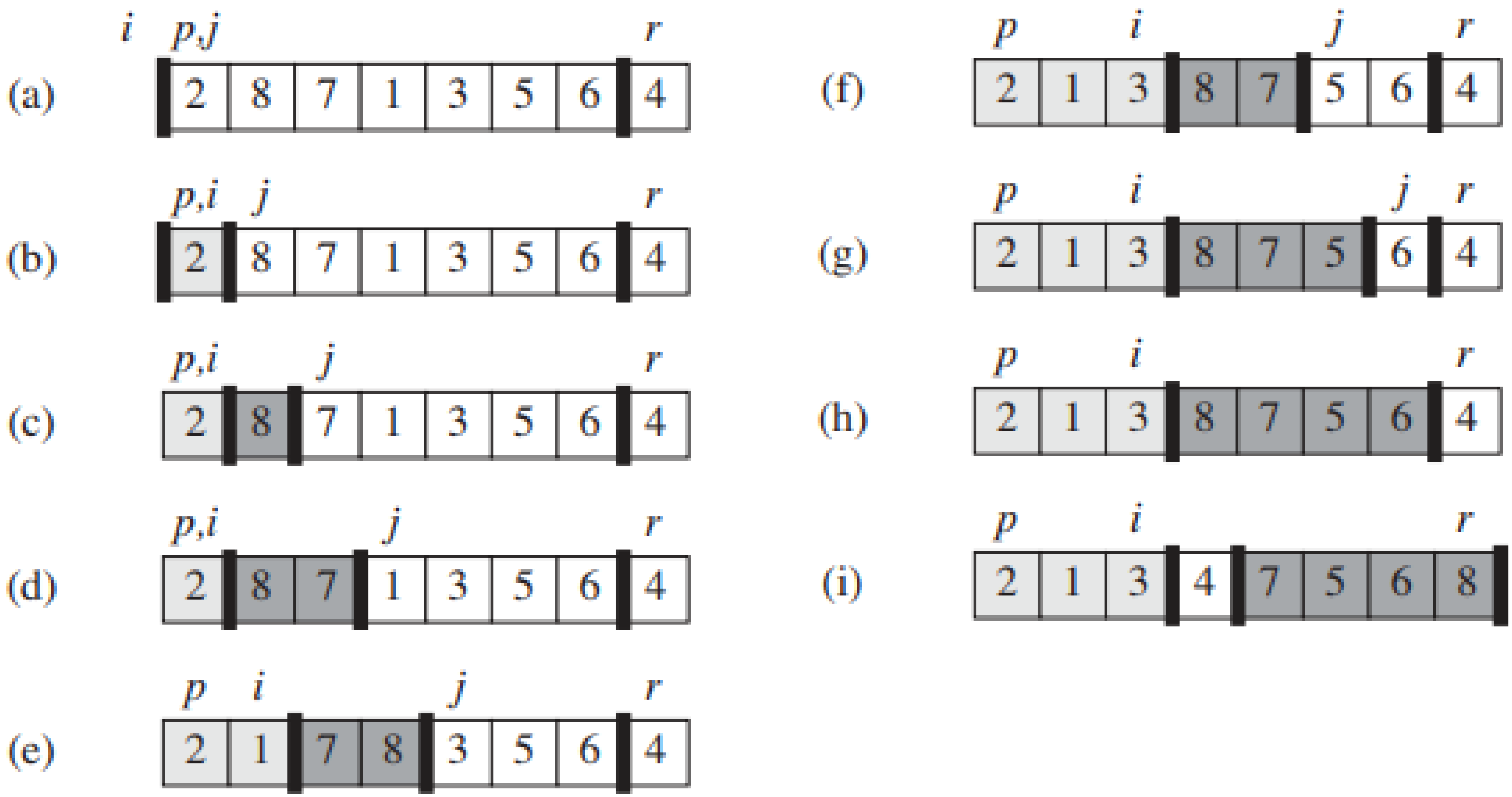


Figure 7.2 The four regions maintained by the procedure PARTITION on a subarray $A[p\,..\,r]$. The values in $A[p\,..\,i]$ are all less than or equal to x, the values in $A[i+1\,..\,j-1]$ are all greater than x, and $A[r] = x$. The subarray $A[j\,..\,r-1]$ can take on any values.

Partition Property

- **Loop invariant:** For any array index *i*
  1. All entries in $A[p \mathinner{..} i] \leq pivot$
  2. All entries in $A[i+1 \mathinner{..} j-1] > pivot$
  3. $A[r] = pivot$
- It's not needed as part of the loop invariant, but the fourth region is $A[j \mathinner{..} r-1]$, whose entries have *not yet been examined*, and so we don't know how they compare to the pivot.

Partition Example



Correctness of Loop Invariant (1/3)

- Initialization:
  - Before the loop starts, all the conditions of the loop invariant are satisfied, because $A[r]$ is the pivot and the subarrays $A[p \mathinner{..} i]$ and $A[i+1 \mathinner{..} j-1]$ are empty

Correctness of Loop Invariant (2/3)

- Maintenance: while the loop is running,
  - if $A[j] \leq pivot$, then $A[j]$ and $A[i+1]$ are swapped and i and j are incremented
  - If $A[j] > pivot$, then increment only j

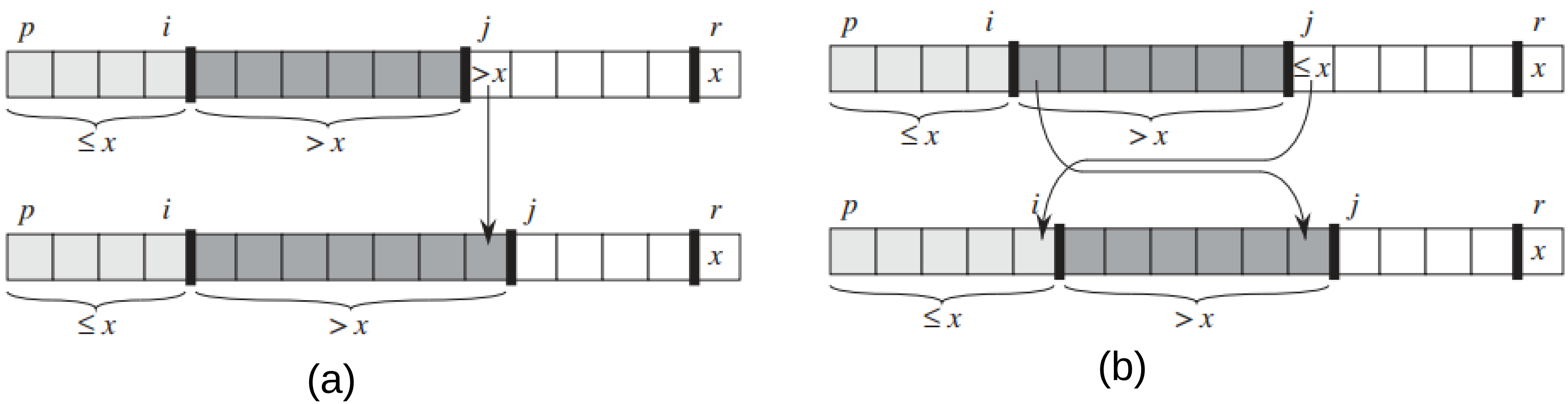


Figure 7.3 The two cases for one iteration of procedure PARTITION. **(a)** If $A[j] > x$, the only action is to increment j, which maintains the loop invariant. **(b)** If $A[j] \leq x$, index i is incremented, $A[i]$ and $A[j]$ are swapped, and then j is incremented. Again, the loop invariant is maintained.

Correctness of Loop Invariant (3/3)

- Termination:
  - When the loop terminates, $j = r$, so all elements in A are partitioned into one of the three cases:
    - $A[p\,..\,i\,] \leq pivot$, $A[i+1\,..\,r-1] > pivot$, and $A[r] = pivot$
- The last operation of PARTITION is to move the pivot from the end of the array to a position between two subarrays:
  - swapping the $pivot$ $A[r]$ and the first element of the second subarray $A[i + 1]$
- Time for partitioning:
  - $\Theta(n)$ to partition an n-element subarray

Quicksort Algorithm

- Video Content
  - An illustration of Quick Sort.

Quicksort Algorithm

Practice Problem

- The operation of PARTITION on an array $A[1..12] = [13,19,9,5,12,8,7,4,21,2,6,11]$ is performed. Then the given array is divided into $A[1..q]$ and $A[q+1..12]$ such that $A[i] \leq A[j]$ for all $1 \leq i \leq q$ and $q+1 \leq j \leq 12$. What are q and $A[q]$?
  - $q = 8$
  - $A[q] = 11$

Performance of Quicksort (1/9)

- The running time of Quicksort depends on the partitioning of the subarrays:
  - If the subarrays are unbalanced, then quicksort can run as slowly as insertion sort (worst case)
  - If the subarrays are balanced, then quicksort can run as fast as mergesort (best case)

Performance of Quicksort (2/9)

- Worst case
  - Occurs when the subarrays are completely unbalanced
  - Has 0 elements in one subarray and (n-1) elements in the other subarray
  - Get the recurrence:
    - $$\begin{aligned} T(n) &= T(n-1) + T(0) + \Theta(n) \\ &= T(n-1) + \Theta(n) = \Theta(n^2) \end{aligned}$$
  - 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

Performance of Quicksort (3/9)

- Best case
  - Occurs when the subarrays are completely balanced every time.
  - Each subarray has $\leq n/2$ elements: $\lfloor n/2 \rfloor$ and $(\lceil n/2 \rceil - 1)$
  - Get the recurrence:
    - $T(n) = 2T(n/2) + \Theta(n) = \Theta(n \lg n)$

Performance of Quicksort (4/9)

- Balanced partitioning
  - 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:
    - $T(n) \leq T(9n/10) + T(n/10) + \Theta(n)$
    - $O(n \lg n)$

Performance of Quicksort (5/9)

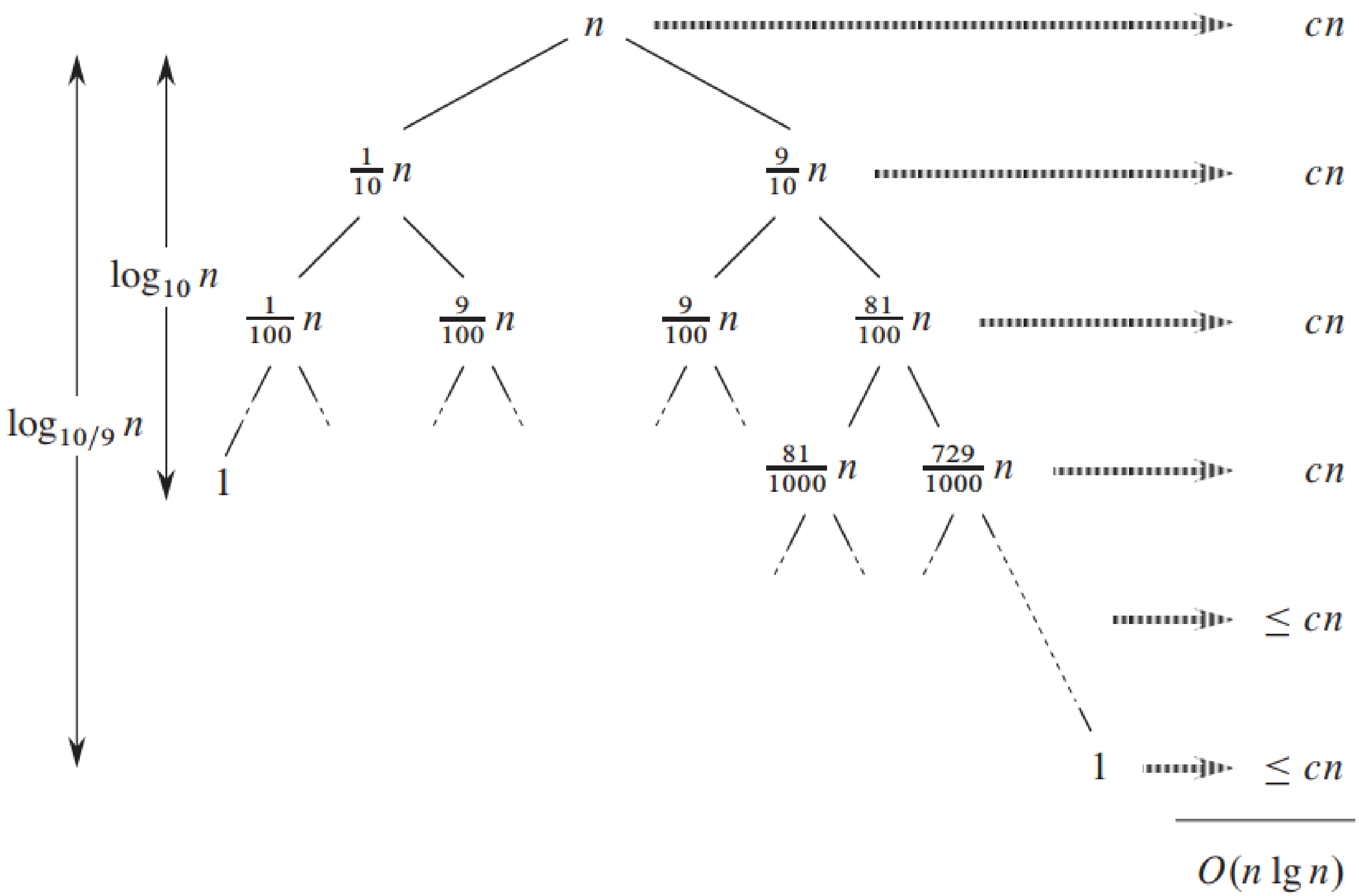


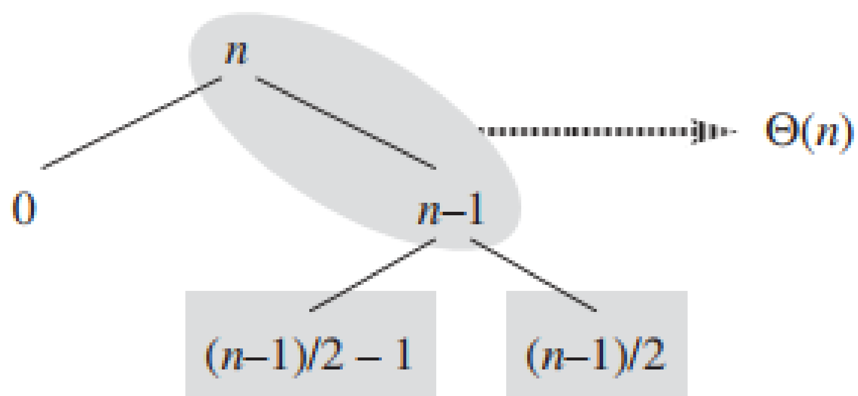
Figure 7.4 A recursion tree for QUICKSORT in which PARTITION always produces a 9-to-1 split, yielding a running time of $O(n \lg n)$. Nodes show subproblem sizes, with per-level costs on the right. The per-level costs include the constant c implicit in the $\Theta(n)$ term.

Performance of Quicksort (6/9)

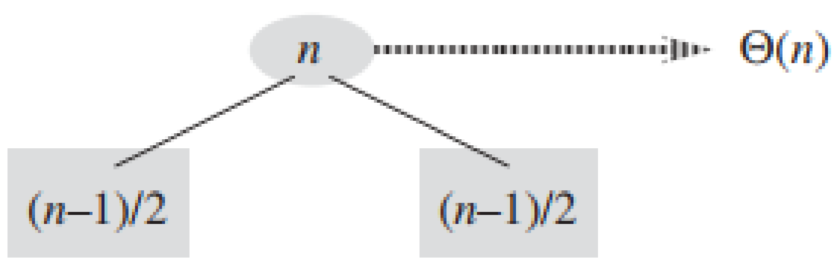
- Intuition for the average case
  - Partitioning will not always be constant
  - PARTITION produces a mix of “good” and “bad” splits
  - Assume that bad and good splits alternate levels in the tree. We will show that the running time is $O(nlgn)$, same as the best case

Performance of Quicksort (7/9)

- Intuition for the average case
  - Consider a bad split follow by a good split as figure (a)
  - There are 3 subarrays of size 0, $\left(\frac{n-1}{2} - 1\right)$, and $\frac{n-1}{2}$
  - Combined partitioning cost is $\Theta(n-1) + \Theta(n) = \Theta(n)$, same as a good split in figure (b)



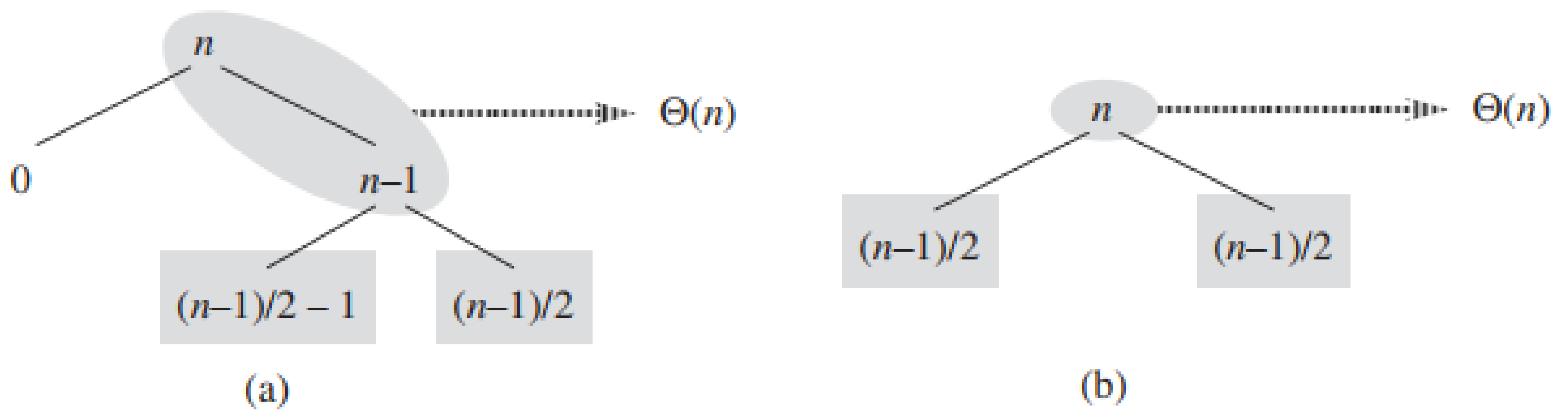
(a)



(b)

Performance of Quicksort (8/9)

- Intuition for the average case
  - The subproblems remaining to be solved in (a), shown with square shading, are no larger than the corresponding subproblems remaining to be solved in (b)



Performance of Quicksort (9/9)

- Intuition for the average case
  - Similar calculation as slide #20, the running time of quicksort, when levels alternate between good and bad splits is $O(nlgn)$, like the best case, but with a slightly larger constant hidden by the O-notation

Practice Problem

- What is the running time of Quicksort when all elements of array A have the same value?
  - The PARTITION algorithm puts all equal elements on one side of the pivot. This means the problem with size n is reduces to one sub-problem with size $(n-1)$, so the recurrence is

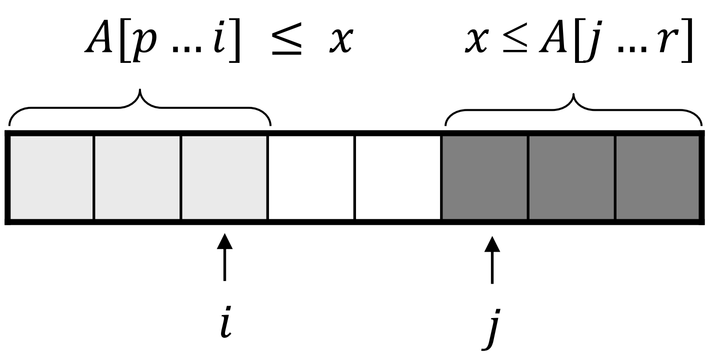
$$T(n) = T(n-1) + n = T(0) + 1 + \ldots + (n-1) + n = \frac{n(n+1)}{2}$$
$$= \Theta(n^2)$$

Another Way of Partitioning (1/2)

- Idea
  - Select a pivot element x around which to partition
  - Grows two regions

$A[p \dots i] \ \leq \ x$

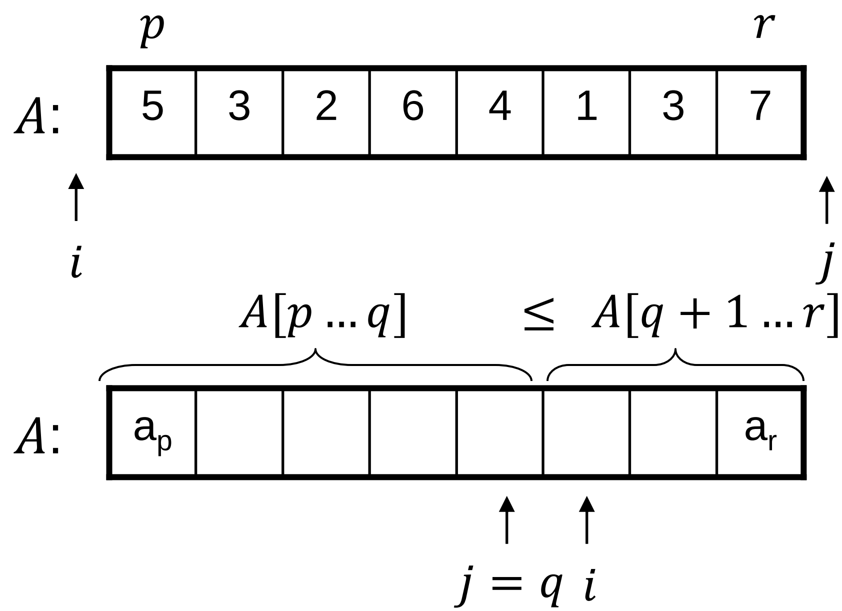
$x \leq A[j \dots r]$



Another Way of Partitioning (2/2)

PARTITION (A, p, r)

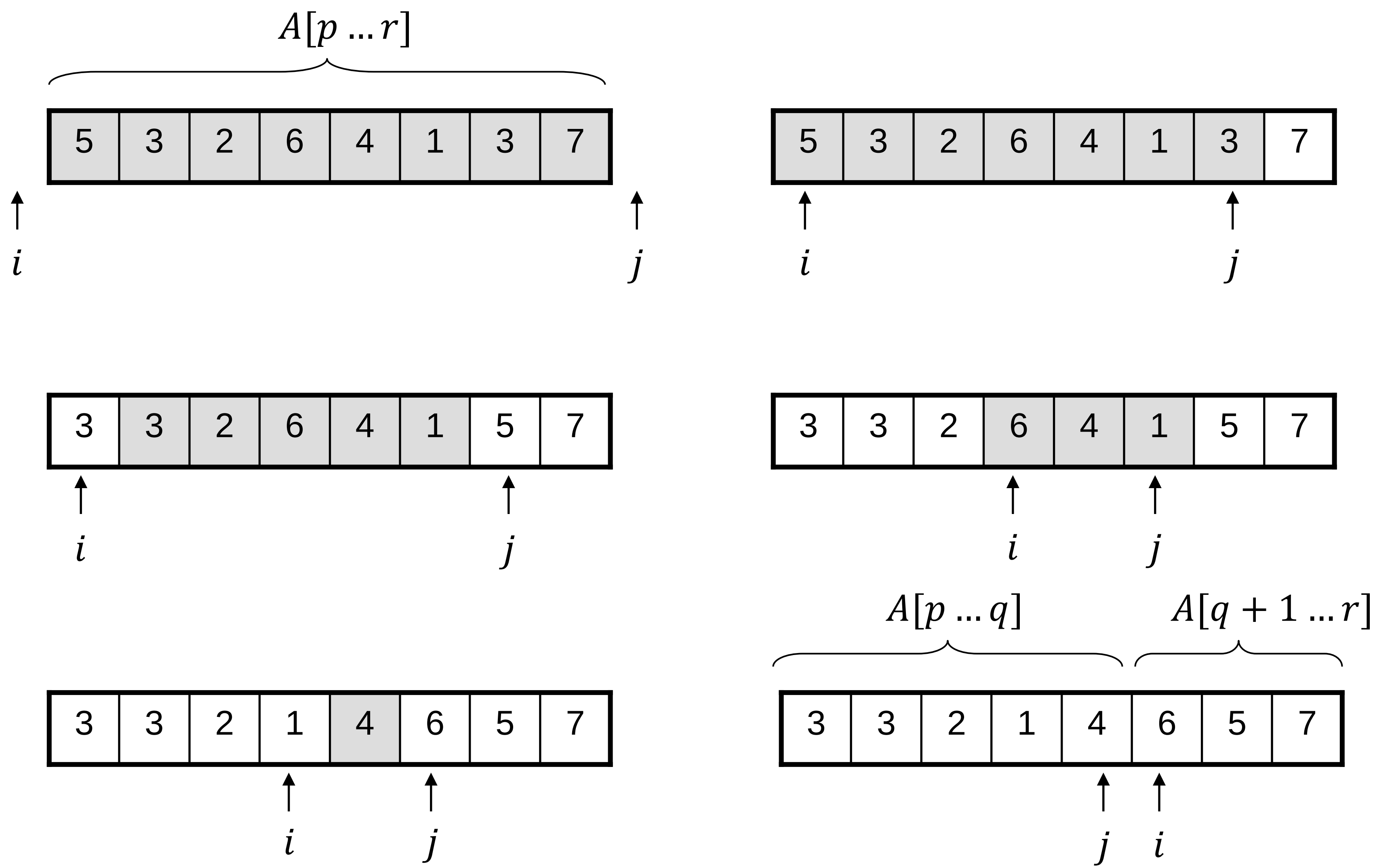
```
x ← A[p]
i ← p − 1
j ← r + 1
while TRUE
        do repeat j ← j − 1
                until A[j] ≤ x
           repeat i ← i + 1
                until A[i] ≥ x
           if i < j
                then exchange A[i] ↔ A[j]
        else return j
```



Running time: $\Theta(n)$

$n = r - p + 1$

Example



A[p ... r]
5 3 2 6 4 1 3 7
i
j
5 3 2 6 4 1 3 7
i
j
3 3 2 6 4 1 5 7
i
j
3 3 2 6 4 1 5 7
i
j
A[p ... q]
A[q + 1 ... r]
3 3 2 1 4 6 5 7
i
j
3 3 2 1 4 6 5 7
j
i

Quicksort Implementation (1/2)

```c
#include <stdio.h>

void quickSort( int[], int, int);
int partition( int[], int, int);

void main()
{
    int a[] = { 7, 12, 1, -2, 0, 15, 4, 11, 9};
    int i;
    printf("\n\nUnsorted array is:  ");
    for(i = 0; i < 9; ++i)
        printf(" %d ", a[i]);

    quickSort( a, 0, 8);

    printf("\n\nSorted array is:  ");
    for(i = 0; i < 9; ++i)
        printf(" %d ", a[i]);
}
```

Quicksort Implementation (2/2)

```
void quickSort( int a[], int l, int r)
{
    int j;

    if( l < r )
    {
     // divide and conquer
        j = partition( a, l, r);
       quickSort( a, l, j-1);
       quickSort( a, j+1, r);
    }
}
```

```
int partition( int a[], int l, int r) {
    int pivot, i, j, t;
    pivot = a[l];
    i = l; j = r+1;

    while( 1)
    {
     do ++i; while( a[i] <= pivot && i <= r );
     do --j; while( a[j] > pivot );
     if( i >= j ) break;
     t = a[i]; a[i] = a[j]; a[j] = t;
    }
    t = a[l]; a[l] = a[j]; a[j] = t;
    return j;
}
```

Randomized Version of Quicksort

- Select a random element as $pivot$
- Modify the PARTITION procedure
  - At each step of the algorithm, we exchange element $A[r]$ with a random element chosen from $A[p \ldots r]$
  - The $pivot\ x = A[r]$ is equally likely to be any element of the array

Randomized Partition Pseudocode

RANDOMIZED-PARTITION(A, p, r)

$i =$ RANDOM(p, r)
exchange $A[r]$ with $A[i]$
return PARTITION(A, p, r)

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$(A, q + 1, r)$

Worst-Case Analysis (1/2)

- $T(n)$ = worst-case running time
- $T(n) = \max_{0 \le q \le n-1} (T(q) + T(n-q-1)) + \Theta(n)$
- Use substitution method to show that the running time of Quicksort is $O(n^2)$
- Guess $T(n) = O(n^2)$
  - Induction goal: $T(n) \le cn^2$
  - Induction hypothesis: $T(k) \le ck^2$ for any $k \le n$

Worst-Case Analysis (2/2)

- Proof of induction goal:

$$T(n) \le \max_{0 \le q \le n-1} (cq^2 + c(n-q-1)^2) + \Theta(n)$$

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

- The expression $q^2 + (n-q-1)^2$ achieves a maximum over the range $0 \le q \le n-1$ at one of the endpoints

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

(see Exercise 7.4-3)

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

Random Variables and Expectation

- Consider **running time $\boldsymbol{T(n)}$** as a random variable
  - This variable associates a real number with each possible outcome (split) of partitioning
- Expected value (expectation, mean) of a discrete random variable X is:

$$\mathrm{E}[X] = \sum_x x \Pr(X = x)$$

  - "Average" over all possible values of random variable X

Indicator Random Variables

- Given a sample space S and an event A, we define the ***indicator random variable*** $I(A)$ associated with A:
  - $I(A) = \begin{cases} 1, & if\ A\ occurs \\ 0, & if\ A\ does\ not\ occurs \end{cases}$
- The expected value of an indicator random variable is:

$$\boldsymbol{E}[I(A)] = \Pr\{A\}$$

- Proof: $\boldsymbol{E}[I(A)] = 1 * \Pr\{I(A) = 1\} + 0 * \Pr\{I(A) = 0\}$

$$= \Pr\{I(A) = 1\}$$

$$= \Pr\{A\}$$

When Do We Compare Two Elements? (1/2)

z_2	z_9	z_8	z_3	z_5	z_4	z_1	z_6	z_{10}	z_7
2	9	8	3	5	4	1	6	10	7

$$Z_{1,6} = \{1, 2, 3, 4, 5, 6\}$$

- Rename the elements of A as $z_1, z_2, \ldots, z_n$, with z_i being the i^{th} smallest element
- Define the set $Z_{ij} = \{z_i, z_{i+1}, \ldots, z_j\}$ to be the set of elements between z_i and z_j

When Do We Compare Two Elements? (2/2)

z_2	z_9	z_8	z_3	z_5	z_4	z_1	z_6	z_{10}	z_7
2	9	8	3	5	4	1	6	10	7

$$Z_{1,6} = \{1, 2, 3, 4, 5, 6\}$$

- Pivot chosen such as: $z_i < x < z_j$
  - z_i and z_j will never be compared
- z_i or z_j is the pivot
  - z_i and z_j will be compared
  - only if one of them is chosen as pivot before any other element in range z_i to z_j
- Only the $pivot$ is compared with elements in both sets

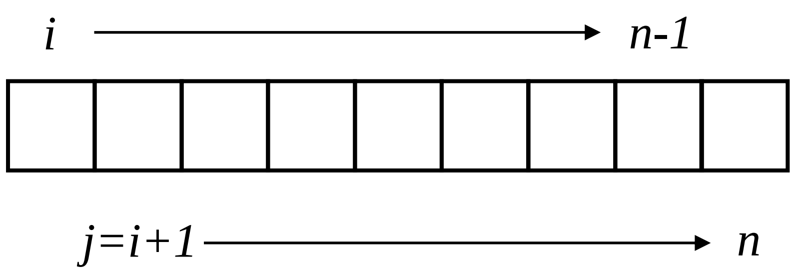
Number of Comparisons in PARTITION (1/5)

- Need to compute the **total number of comparisons** performed **in all calls to PARTITION**
- $X_{ij} = I\{z_i \text{ is compared to } z_j\}$
  - For any comparison during the entire execution of the algorithm, not just during one call to PARTITION

Number of Comparisons in PARTITION (2/5)

- Each pair of elements can be compared at most once
  - $X_{ij} = I\{z_i \text{ is compared to } z_j\}$

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



- X represents the total number of comparisons performed by the algorithm

Number of Comparisons in PARTITION (3/5)

- X is an indicator random variable
  - Compute the **expected value**

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

by linearity
of expectation

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

the expectation of X_{ij} is equal to the
probability of the event "z_i is compared to z_j"

Number of Comparisons in PARTITION (4/5)

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

There are $(j - i + 1)$ elements between z_i and z_j

- $Pivot$ is chosen randomly and independently

- The probability that any particular element is the first one chosen is $1/(j - i + 1)$

Number of Comparisons in PARTITION (5/5)

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

Expected running time of Quicksort using RANDOMIZED-PARTITION is $O(nlgn)$

Thanks to contributors

Mr. Pham Van Nguyen (2022)

Dr. Dang Thien Binh (2017 – 2022)

Prof. Hyunseung Choo (2017 – 2022)