

CH-231-A

**Algorithms and Data Structures**

ADS

**Lecture 16**

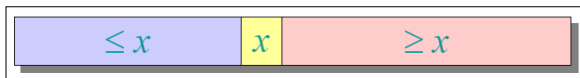
Dr. Kinga Lipskoch

Spring 2022

## Quicksort: Divide & Conquer

### 1. Divide:

Partition the array into two subarrays around a pivot  $x$  such that the elements in lower subarray  $\leq x \leq$  the elements in upper subarray.



### 2. Conquer:

Recursively sort the two subarrays

### 3. Combine:

Nothing to be done.

**Key observation:** Linear-time partitioning subroutine.

## Quicksort: Divide (1)

In the literature there are two popular division (partition) methods:

- ▶ Nico Lomuto's partition used in the textbook of Cormen and other textbooks
- ▶ C.A.R. Hoare's partition
- ▶ Hoare's partition is more efficient than Lomuto's partition because it does three times fewer swaps on average and it creates efficient partitions even when all values are equal
- ▶ But the time complexities are the same

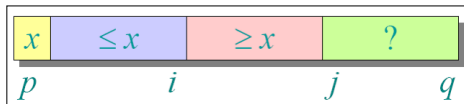
## Quicksort: Divide (2)

```

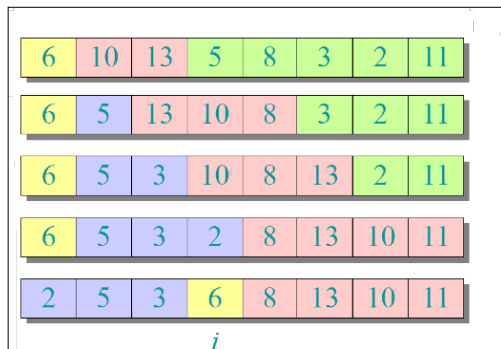
1 Partition(A, p, q)           //A[p..q]
2   x = A[p]                   //pivot = A[p]
3   i = p
4   for j = p + 1 to q
5       if A[j] <= x then
6           i = i + 1
7           exchange A[i] and A[j]
8   exchange A[p] and A[i]
9   return i

```

Invariant:



## Quicksort: Partition Example



## Quicksort: Divide Complexity

```
1 Partition(A, p, q)      //A[p. . q]
2   x = A[p]              //pivot = A[p]
3   i = p
4   for j = p + 1 to q
5       if A[j] <= x then
6           i = i + 1
7           exchange A[i] and A[j]
8   exchange A[p] and A[i]
9   return i
```

Time complexity:

For  $n = q - p + 1$  elements:  $T(n) = \Theta(n)$

## Quicksort: Conquer

```
1 QuickSort(A, p, r)
2   if p < r
3       q = Partition(A, p, r)
4       QuickSort(A, p, q - 1)
5       QuickSort(A, q + 1, r)
```

Initial call: QuickSort(A, 1, n)

## Runtime Analysis (1)

- ▶ Assume all input elements are distinct.
  - ▶ In practice, there are better partitioning algorithms for when duplicate input elements may exist.
- ▶ Let  $T(n)$  be the worst-case running time for  $n$  elements.
- ▶ Worst case:
  - ▶ Input sorted or reverse sorted.
  - ▶ Partition around min or max element.
  - ▶ One side of partition always has no elements.



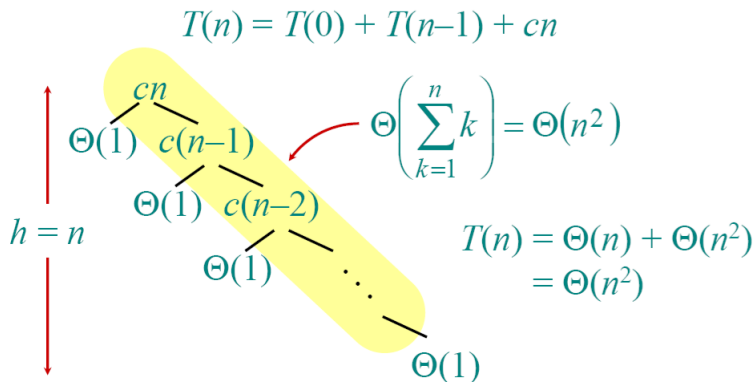
## Runtime Analysis (2)

Worst case:

$$\begin{aligned}T(n) &= T(0) + T(n-1) + \Theta(n) \\&= \Theta(1) + T(n-1) + \Theta(n) \\&= T(n-1) + \Theta(n) \\&= \Theta(n^2) \quad \textit{(arithmetic series)}\end{aligned}$$

## Runtime Analysis (3)

Worst-case recursion tree:



## Runtime Analysis (4)

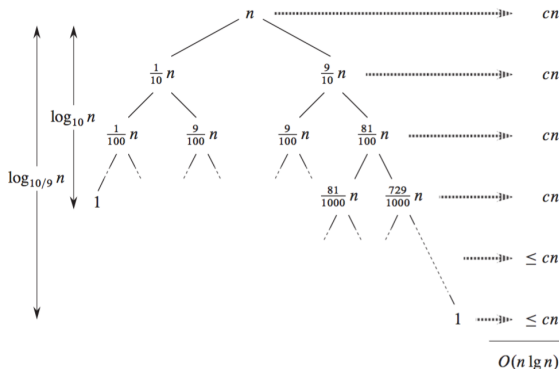
Best case:

- ▶ In best case partition splits the array evenly.
- ▶  $T(n) = 2T(n/2) + \Theta(n) = \Theta(n \lg n)$
- ▶ This is the same as Merge Sort.

## Runtime Analysis (5)

What if the split is  $1/10 : 9/10$ ?

$$T(n) = T\left(\frac{1}{10}n\right) + T\left(\frac{9}{10}n\right) + \Theta(n)$$



## Runtime Analysis

- ▶ What if we alternate between lucky and unlucky choices
  - ▶  $L(n) = 2U(n/2) + \Theta(n)$  **lucky**
  - ▶  $U(n) = L(n-1) + \Theta(n)$  **unlucky**
- ▶ Solving:
  - ▶ 
$$\begin{aligned} L(n) &= 2(L(n/2 - 1) + \Theta(n/2)) + \Theta(n) \\ &= 2L(n/2 - 1) + \Theta(n) \\ &= \Theta(n \lg n) \end{aligned}$$
- ▶ How can we make sure that this is usually happening?

## Randomized Quicksort (1)

- ▶ Idea: Partition around a **random** element.
- ▶ Running time is independent of the input order.
- ▶ No assumptions need to be made about the input distribution.
- ▶ No specific input elicits the worst-case behavior.
- ▶ The worst case is determined only by the output of a random-number generator.

## Randomized Quicksort (2)

```
RANDOMIZED-PARTITION( $A, p, r$ )  
1   $i = \text{RANDOM}(p, r)$   
2  exchange  $A[p]$  with  $A[i]$   
3  return PARTITION( $A, p, r$ )
```

RANDOMIZED-QUICKSORT( $A, p, r$ )

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

## Randomized Quicksort (3)

- ▶ Let  $T(n)$  be the random variable for the running time of the randomized quicksort on an input of size  $n$  (assuming random numbers are independent).

$$X_k = \begin{cases} 1 & \text{if PARTITION generates a } k : n-k-1 \text{ split,} \\ 0 & \text{otherwise.} \end{cases}$$

- ▶ For  $k = 0, 1, \dots, n-1$ , define indicator random variable
- ▶  $E[X_k] = \Pr\{X_k = 1\} = 1/n$ , since all splits are equally likely (assuming elements are distinct).



## Randomized Quicksort (4)

Recurrence:

$$T(n) = \begin{cases} T(0) + T(n-1) + \Theta(n) & \text{if } 0 : n-1 \text{ split,} \\ T(1) + T(n-2) + \Theta(n) & \text{if } 1 : n-2 \text{ split,} \\ \vdots & \\ T(n-1) + T(0) + \Theta(n) & \text{if } n-1 : 0 \text{ split,} \end{cases}$$

$$= \sum_{k=0}^{n-1} X_k (T(k) + T(n-k-1) + \Theta(n))$$

## Randomized Quicksort (5)

Calculating expectations:

$$\begin{aligned}
 E[T(n)] &= E\left[\sum_{k=0}^{n-1} X_k (T(k) + T(n-k-1) + \Theta(n))\right] \\
 &= \sum_{k=0}^{n-1} E[X_k (T(k) + T(n-k-1) + \Theta(n))] \\
 &= \sum_{k=0}^{n-1} E[X_k] \cdot E[T(k) + T(n-k-1) + \Theta(n)] \\
 &= \frac{1}{n} \sum_{k=0}^{n-1} E[T(k)] + \frac{1}{n} \sum_{k=0}^{n-1} E[T(n-k-1)] + \frac{1}{n} \sum_{k=0}^{n-1} \Theta(n) \\
 &= \frac{2}{n} \sum_{k=2}^{n-1} E[T(k)] + \Theta(n)
 \end{aligned}$$

## Randomized Quicksort (6)

- ▶ Use substitution method to solve recurrence.
- ▶ Guess:  $E[T(n)] = \Theta(n \lg n)$ .
- ▶ Prove:  $E[T(n)] \leq an \lg n$  for constant  $a > 0$ .
- ▶ Use:

$$\sum_{k=2}^{n-1} k \lg k \leq \frac{1}{2} n^2 \lg n - \frac{1}{8} n^2$$

(proof by induction)

## Randomized Quicksort (7)

Proof:

$$\begin{aligned}
 E[T(n)] &\leq \frac{2}{n} \sum_{k=2}^{n-1} ak \lg k + \Theta(n) \\
 E[T(n)] &\leq \frac{2}{n} \sum_{k=2}^{n-1} ak \lg k + \Theta(n) \\
 &= \frac{2a}{n} \left( \frac{1}{2} n^2 \lg n - \frac{1}{8} n^2 \right) + \Theta(n) \\
 &= an \lg n - \left( \frac{an}{4} - \Theta(n) \right) \\
 &\leq an \lg n,
 \end{aligned}$$

if  $a$  is chosen large enough.

## Quicksort: Conclusion

- ▶ Quicksort is a great general-purpose sorting algorithm.
- ▶ Quicksort is often the best practical choice because its expected runtime is  $\Theta(n \lg n)$  and the constant is quite small.
- ▶ Quicksort is typically over twice as fast as MergeSort.
- ▶ Quicksort is an in-situ sorting algorithm (debatable).
- ▶ Quicksort has a worst-case runtime of  $\Theta(n^2)$  when the array is already sorted.
- ▶ Visualization Randomized Quicksort:  
<http://www.sorting-algorithms.com/quick-sort>