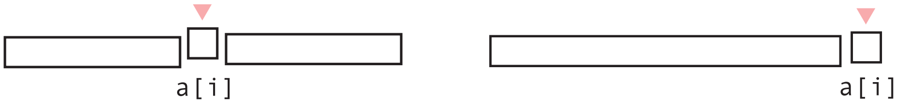
Quicksort Performace



One is the basis of the best case for *quicksort*, the other is the basis of the worst case.

➪ Extreme cases

✎ Split equally (at mid point)

✎ Split at edge (already partitioned)

Quiz
How many key comparisons occur in this case?

Exercise
Give examples of input instances which cause the worst efficiency in *quicksort*?

➪ Worst-case sequence 🕮

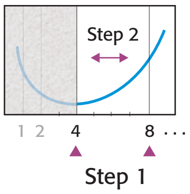
Quicksort Performance Analysis

Algorithm *quicksort*

```
if l < r then
    s ← partition(a[l .. r])
    quicksort (a[l .. s − 1])
    quicksort (s + 1 .. r])
```

Quiz Which steps depend on *n*?

Exercise
Use backward substitution to solve the worst-case recurrence for $n=2^k$.



Exercise
Compare results obtained from efficiency sequence (textbook) and *WolframAlpha* (recurrence).

➪ Choice of basic operation

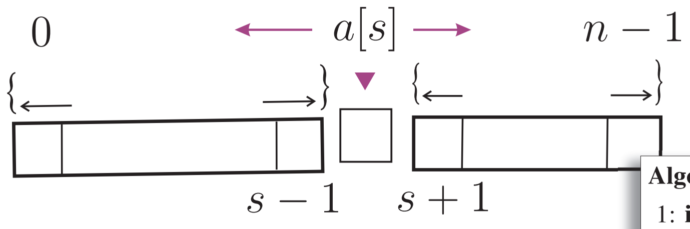
➪ Best-case recurrence

- ✎ Solve for $n = 2^k, k = 1, 2, \cdots$
- ✎ Use *smoothness rule* to extend

➪ Worst-case recurrence

Quicksort Performance
Average Case

Quiz
What is the <u>length</u> of each sublist in this typical partition scenario?



Algorithm *quicksort*

```
if l < r then
    s ← partition(a[l .. r])
    quicksort (a[l .. s − 1])
    quicksort (s + 1 .. r])
```

The 3 ingredients to calculate an average (expected value): data item, dataset, and probability distribution.

Questions

- ✎ How many comparisons typically?
- ✎ What possible positions for *s*?
- ✎ How likely each position?

Just need to know the number when all positions (=cases for *s*) are equally likely.

Quicksort Implementation
A Closer Look

First `while`-loop will stop since (eventually) a[*i*] = *p* is never < *p*.

Quiz
How many <u>key comparisons</u> are needed before a partition is achieved?

Algorithm *quicksort2*

```
if l < r then
    p ← a[r], i ← l − 1, j ← r
    loop
        while a[++i] < p do
        while a[−−j] > p do
        if i ≥ j then break
        swap a[i], a[j]
    swap a[i], a[r]
    quicksort2(a[l..i − 1])
    quicksort2(a[i + 1..r])
```

Reference: Robert Sedgewick, Algorithms in C, Addison-Wesley, 1990

Quiz
When will the second `while`-loop run out of bounds?

Quicksort Implementation
Issues to Consider

Exercise
Modify *quicksort2* to use a[*l*] as pivot.

❶ Choice of pivot element

Quiz
Suggest 2 methods to handle runaway scan index. Which one is preferred?

❷ Runaway inner loop (scan)

E S O R T I N G E X B M P L A

Exercise What if the pivot happens to be the smallest element?

❸ Scans can converge ($i = j$)

Careless coding may cause poor performance.

Fragility

Quicksort Performance Improvement

A modern quicksort

- ✎ Tiny inner loops with strong locality
- ✎ Handle small lists differently
- ✎ Better pivot handling
- ✎ 3-way partition is *the* way*

* Sedgewick and Bently "Quicksort is Optimal"

In terms of choice, and efficiently dealing with repeated pivot.

Exercise
Lookup efficiency of insertion sort for nearly sorted lists, report your findings and sources in the discussion group.

Quicksort Performance
Conclusions

- **In-place, time efficiency**

$$\Omega(n \log n) \longleftarrow C_{av} \in \Theta(n \log n) \longrightarrow O(n^2)$$

- **Choice of pivot is important**

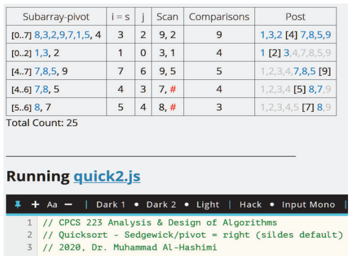
Quiz
What is the space efficiency of *quicksort*?

- **Compare with *mergsort***

Divide-Conquer Sort
Quicksort

Turing Award
1980

- ➪ **Invented 1960 by C.A.R. Hoare**
- ➪ **Good general-purpose sort**
  - ✎ Easy to implement
  - ✎ Well-known characteristics
  - ✎ Performs well widely
  - ✎ Low space (in-place)
- 📖 ➪ **Fragile, not stable**



Subarray-pivot	i = s	j	Scan	Comparisons	Post
[0..7] 8,3,2,9,7,1,5, 4	3	2	9, 2	9	1,3,2 [4] 7,8,5,9
[0..2] 1,3, 2	1	0	3, 1	4	1 [2] 3,4,7,8,5,9
[4..7] 7,8,5, 9	7	6	9, 5	5	1,2,3,4,7,8,5 [9]
[4..6] 7,8, 5	4	3	7, #	4	1,2,3,4 [5] 8,7,9
[5..6] 8, 7	5	4	8, #	3	1,2,3,4,5 [7] 8,9

Total Count: 25

Running quick2.js


```
// CPCS 223 Analysis & Design of Algorithms
// Quicksort - Sedgewick/pivot = right (sildes default)
// 2020, Dr. Muhammad Al-Hashimi
```