QuickSort Analysis

Wait a min, we need a whole lecture for this?

QuickSort

- Easy to understand! (divide-and-conquer…)
- Moderately hard to implement correctly.
- Harder to analyze. (Randomization…)
- Challenging to optimize.

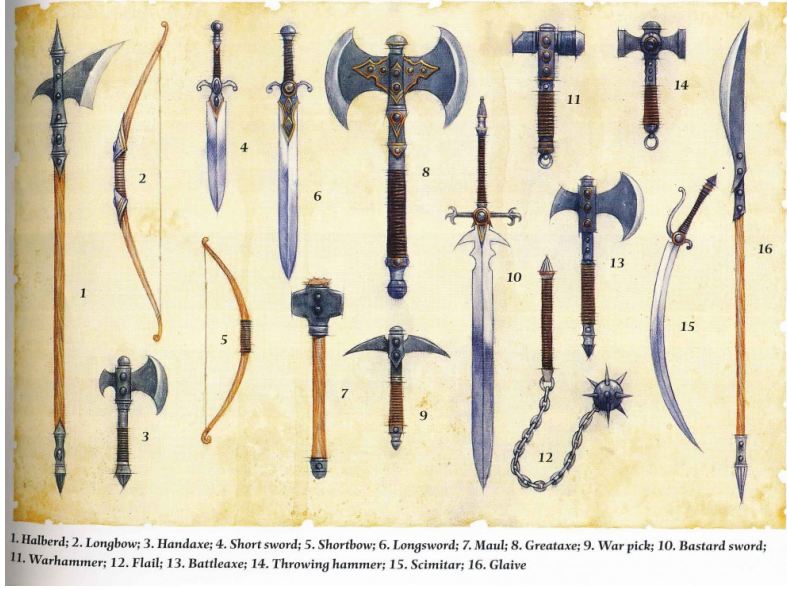
Let’s Revise Some Probability

Probability

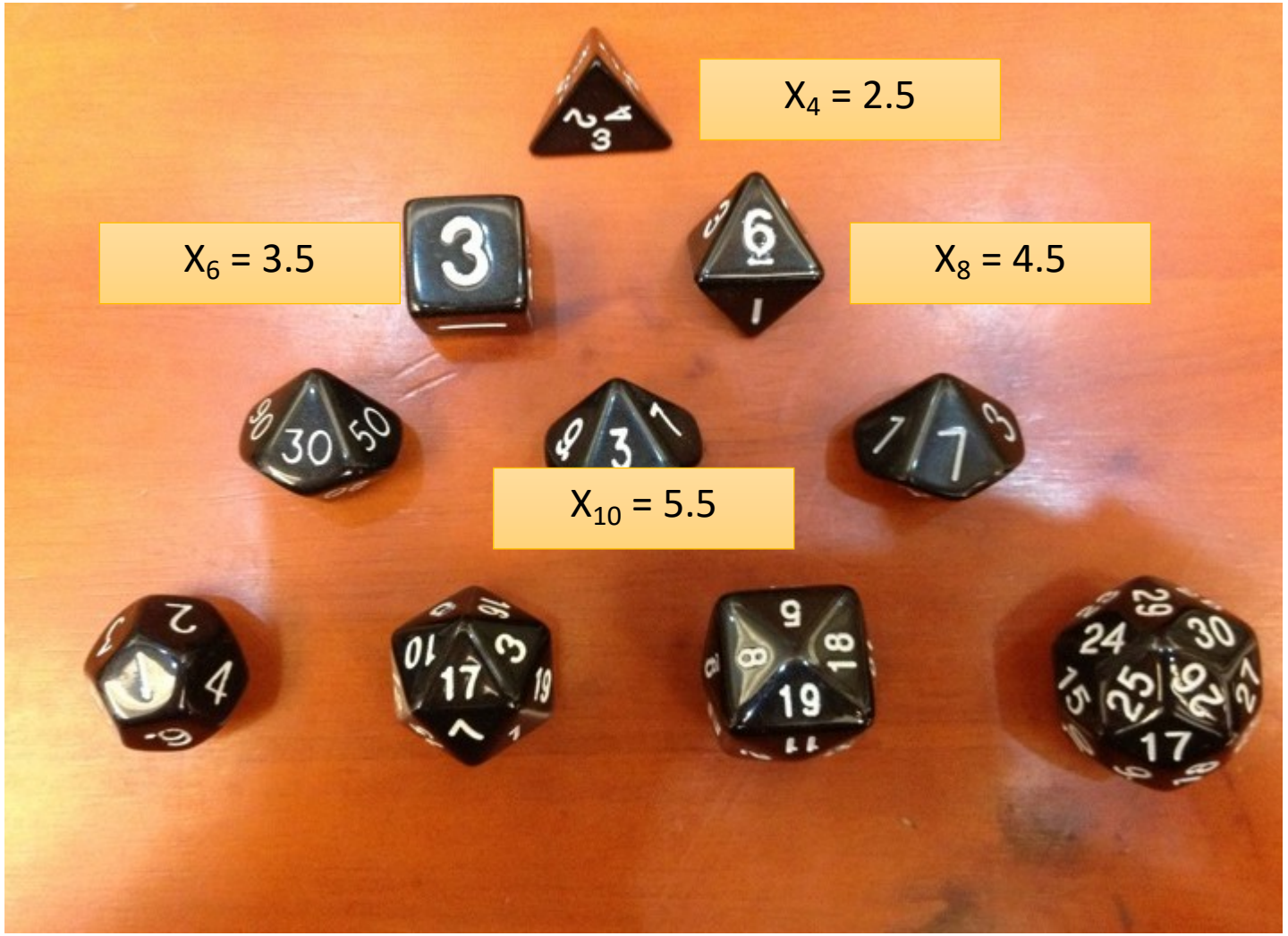

3
6
30
50
3
7
17
19
25
17

Which Weapon will you pick?

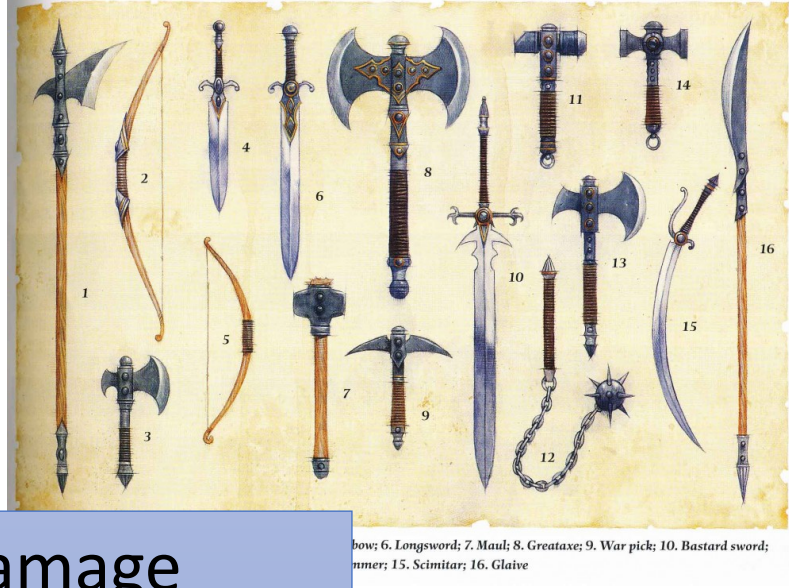
Name	Source	Prof	Damage
Falchion	PHB	+3	2d4
Glaive	PHB	+2	2d4
Greataxe	PHB	+2	1d12
Greatsword	PHB	+3	1d10
Halberd	PHB	+2	1d10
Heavy flail	PHB	+2	2d6
Heavy war pick	AV	+2	1d12



1. Halberd; 2. Longbow; 3. Handaxe; 4. Short sword; 5. Shortbow; 6. Longsword; 7. Maul; 8. Greataxe; 9. War pick; 10. Bastard sword; 11. Warhammer; 12. Flail; 13. Battleaxe; 14. Throwing hammer; 15. Scimitar; 16. Glaive


$X_4 = 2.5$
$X_6 = 3.5$
$X_8 = 4.5$
$X_{10} = 5.5$

Which Weapon will you pick?



bow; 6. Longsword; 7. Maul; 8. Greataxe; 9. War pick; 10. Bastard sword;
nmer; 15. Scimitar; 16. Glaive

Name	Source	Prof	Damage	
Falchion	PHB	+3	2d4	-
Glaive	PHB	+2	2d4	-
Greataxe	PHB	+2	1d12	-
Greatsword	PHB	+3	1d10	-
Halberd	PHB	+2	1d10	-
Heavy flail	PHB	+2	2d6	-
Heavy war pick	AV	+2	1d12	-

Expected Damage

5

5

6.5

5.5

5.5

7

6.5

If you have a fair coin

- The Probability of getting a head = ½
- The Probability of getting a tail = ½

- If you flip the coin 10^{10} times, about how many heads do you expect?
  - ½ × 10^{10} heads
- If you flip the coin n times, about how many heads do you expect?
  - ½ × n heads

If you have a fair coin

- The Probability of getting a head = ½
- The Probability of getting a tail = ½

- If you flip the coin n times, you expect ½ × n heads
- How many times (n) do you need to flip so that you expect to have one head?
  - $½ \times n = 1$
  - $n = (½)^{-1} = 2$
- It means that if I flip the coins two times, I expect there will be one head

If an event has a probability of p

- The Probability of an event is p

- If you repeat n times, you the event appears $p \times n$ times
- How many times (n) do you need to repeat so that you the event to happen once?
  - $p \times n = 1$
  - $n = (p)^{-1} = 1/p$
- It means that if repeat $1/p$ times, I expect the event will happen once.
- If I want the event to happen once, the expected number of repetition is $1/p$.

If an event has a probability of p

- The Probability of an event is p
- If I want the event to happen once, the expected number of repetition is $1/p$.

- Example:
  - How many time do I need to draw a card randomly from a pile of 52 cards in order to get a King?
  - Probably of drawing a King is p = 1/13
  - If I draw $1/p = 13$ cards, I expect to get one King

QuickSort Time Complexity

Recap: Time Complexity?

```
QuickSort(A[1..n], n)
    if (n==1) then return;
    else
        p = ThreeWayPartition(A[1..n], n)
        x = QuickSort(A[1..p-1], p-1)
        y = QuickSort(A[p+1..n], n-p)
```

$O(n)$

$T(p)$

$T(n-p)$

- Lucky case
  - If $p = n/2$ all the time
- $T(n) = cn + 2\ T(n/2)$
- Same as MergeSort!

The pivot we picked is always the median of the array

Recap: Time Complexity?

```
QuickSort(A[1..n], n)
    if (n==1) then return;
    else
        p = ThreeWayPartition(A[1..n], n)
        x = QuickSort(A[1..p-1], p-1)
        y = QuickSort(A[p+1..n], n-p)
```

$O(n)$

$T(p)$

$T(n\text{-}p)$

- But what if $p = 1$ all the time?
- $T(n) = cn + T(n - 1) + T(1)$

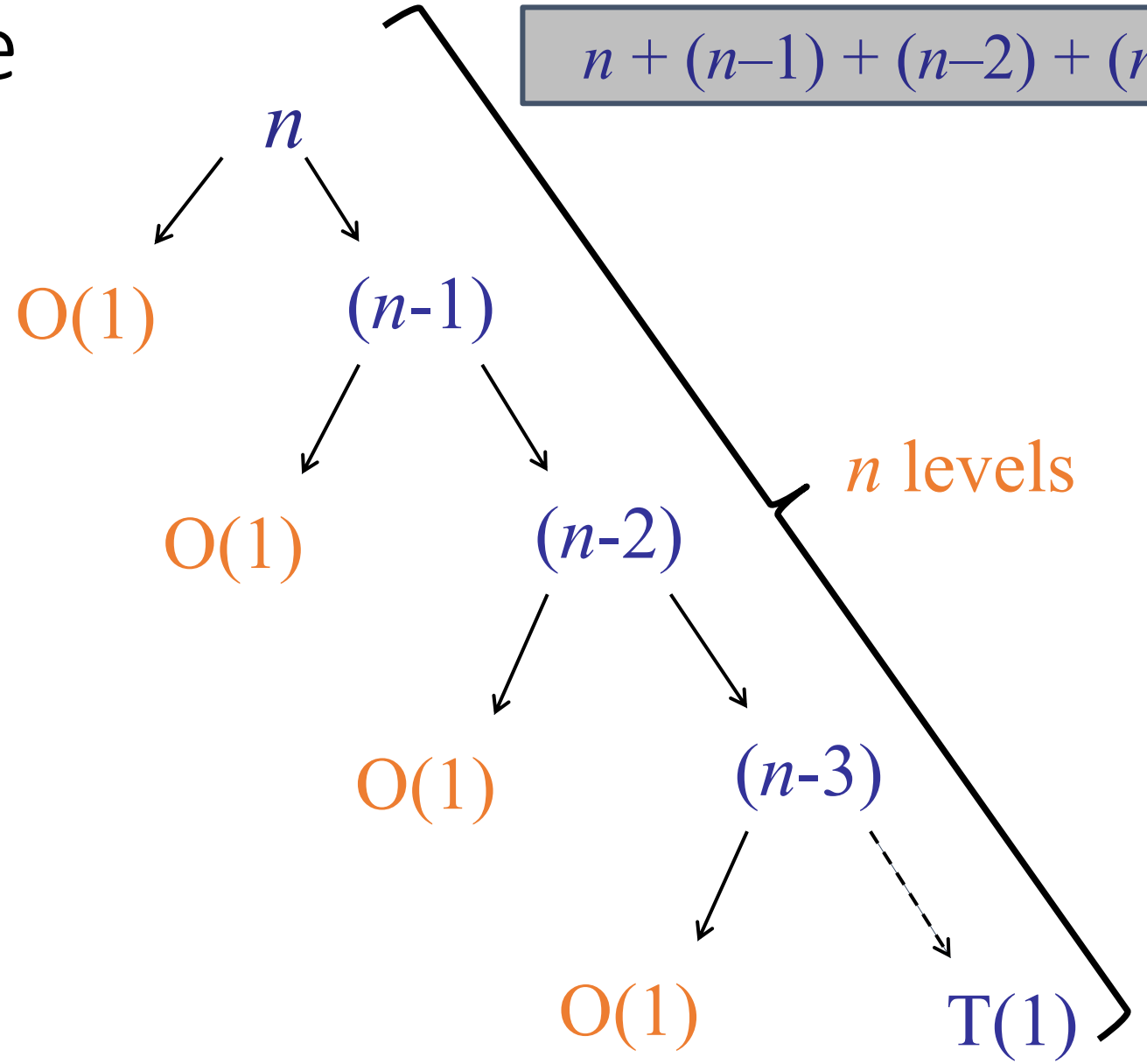
$= cn + c(n - 1) + T(n - 2) + T(1) + T(1)$

$= cn + c(n - 1) + c(n - 2) + T(n - 2) + T(1) + T(1) + T(1)$

$= c(n + (n - 1) + (n - 2) + (n - 3) + \ldots + 1) + T(n) = O(n^2)$

Worst-case

$n + (n–1) + (n–2) + (n–3) + \ldots = \mathbf{O(n^2)}$



Time Complexity

- Lucky case
  - If $p = n/2$ all the time
  - $T(n) = cn + 2\ T(n/2) = \mathrm{O}(n \log n)$
- Worst case
  - if $p = 1$ all the time
  - $T(n) = \mathrm{O}(n^2)$

Today

- Next: How about choose something in the middle?
  - E.g. $n/10 > p > 9n/10$?
  - That will give $T(n) = \mathrm{O}(n \log n)$!!!

Ways to Choose a Pivot

- Choose the First

- Choose the Last

- Choose the Middle

- Choose the Median

- Choose randomly

Ways to Choose a Pivot

- Choose the First

- Choose the Last

- Choose the Middle

- All may results in $\mathrm{O}(n^2)$
- Challenge: Try to reverse engineer such a input list?

Ways to Choose a Pivot

- Choosing the Median

  - $O(n \log n)$ Great!
  - But how to choose the Median?
- Choose randomly

  - Expected time $O(n \log n)$
  - Huh?
  - Isn't it the best?
    - (Why do we still bother to find the median!?!)

Idea

```
Repeat
    Partition with a random pivot
Until the pivot is good.
```

- We said that a pivot is good if it divides the array into two pieces, each of which is size at least n/10.
- Or technically:
- After partitioning, let
  - L = no. of elements that are smaller than the pivot
  - H = no. of elements that are larger than the pivot
  - $L > n/10$ and $H > n/10$

Question 1

How many times we need to repeat?

Question 2

How does this lead to an O (n log n) algorithm

Paranoid QuickSort

```
ParanoidQuickSort(A[1..n], n)
     if (n == 1) then return;
     else
          repeat
               pIndex = random(1, n)
               p = partition(A[1..n], n, pIndex)
          until p > (1/10)*n and p < (9/10)*n

          x = QuickSort(A[1..p-1], p-1)
          y = QuickSort(A[p+1..n], n-p)
```

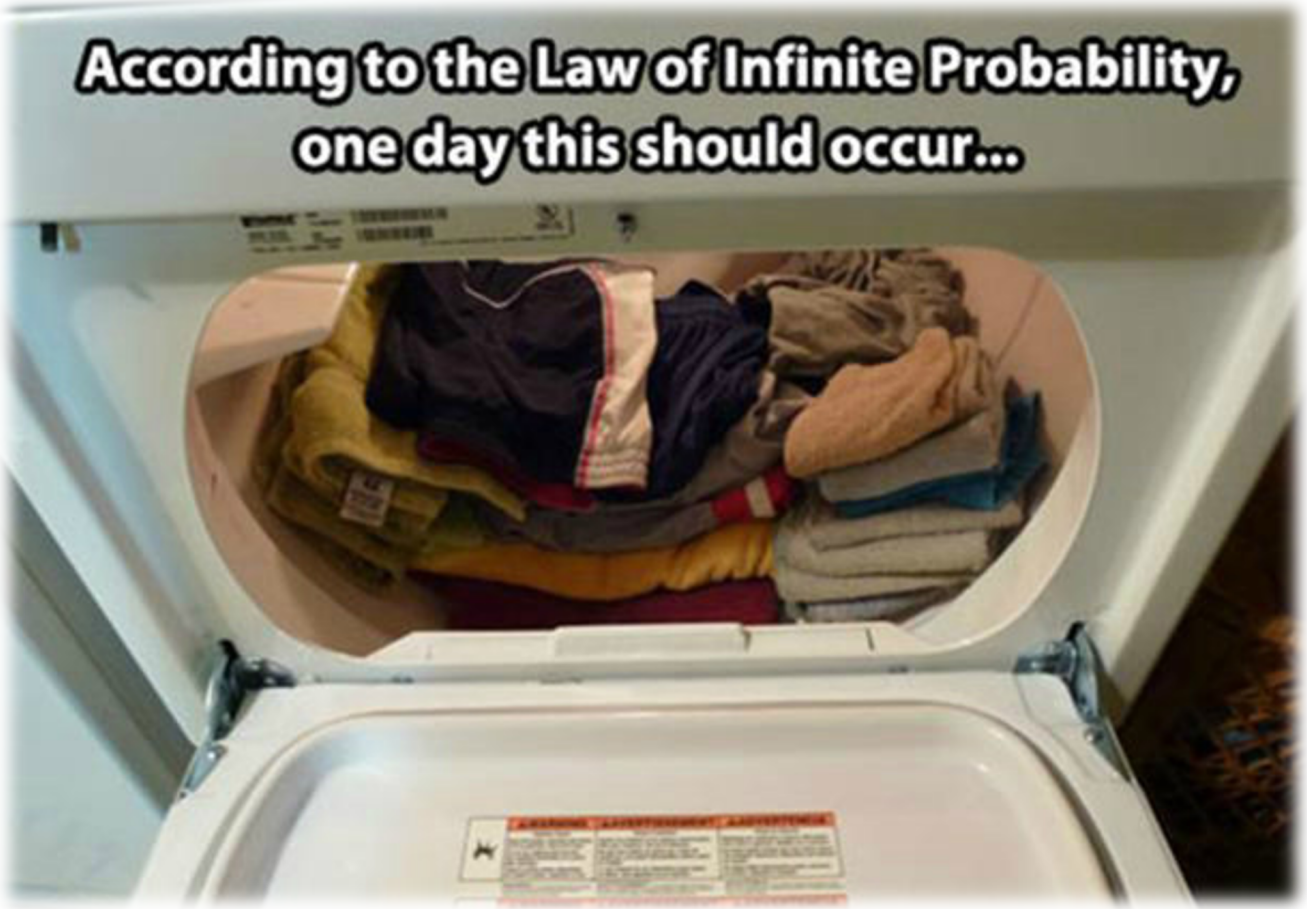
Question 1

How many times we need to repeat?

If we need to repeat $O(n)$ time, then it is $O(n^2)$ for ONE partitioning

Question 1

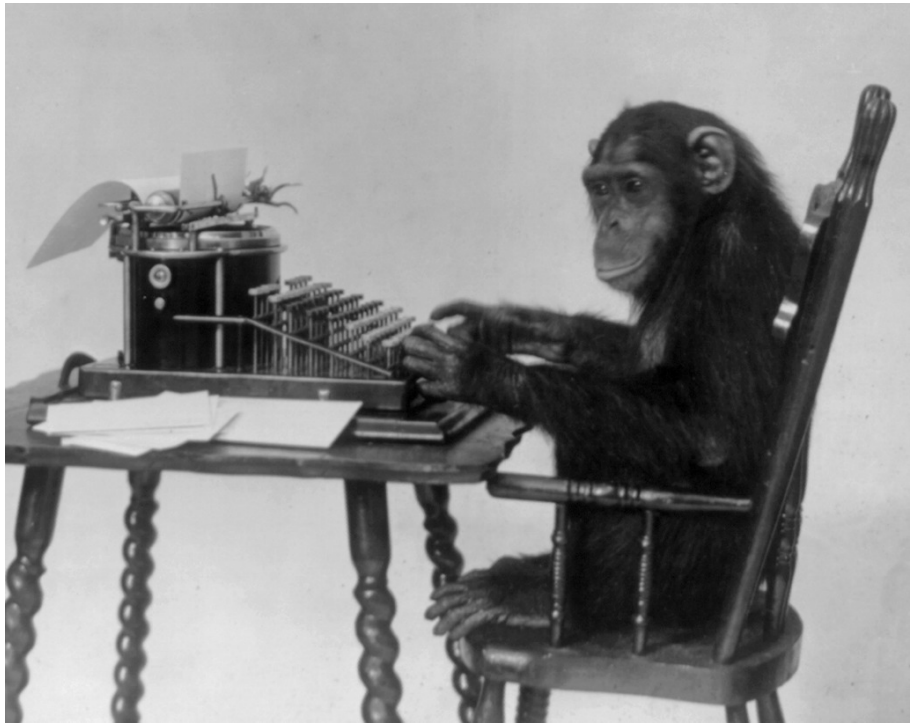
- How many repetitions until the partitioning is good?



Infinite monkey theorem

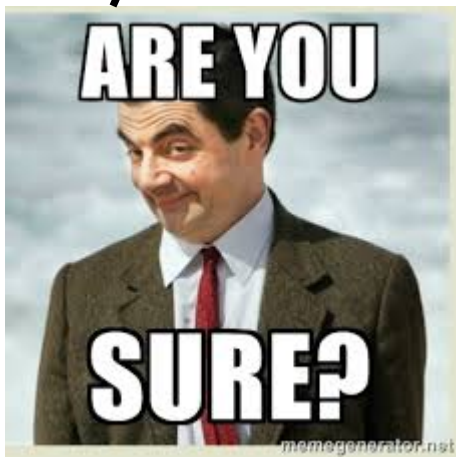
- The infinite monkey theorem states that a monkey hitting keys at random on a typewriter keyboard for an infinite amount of time will almost surely type any given text, such as the complete works of William Shakespeare.

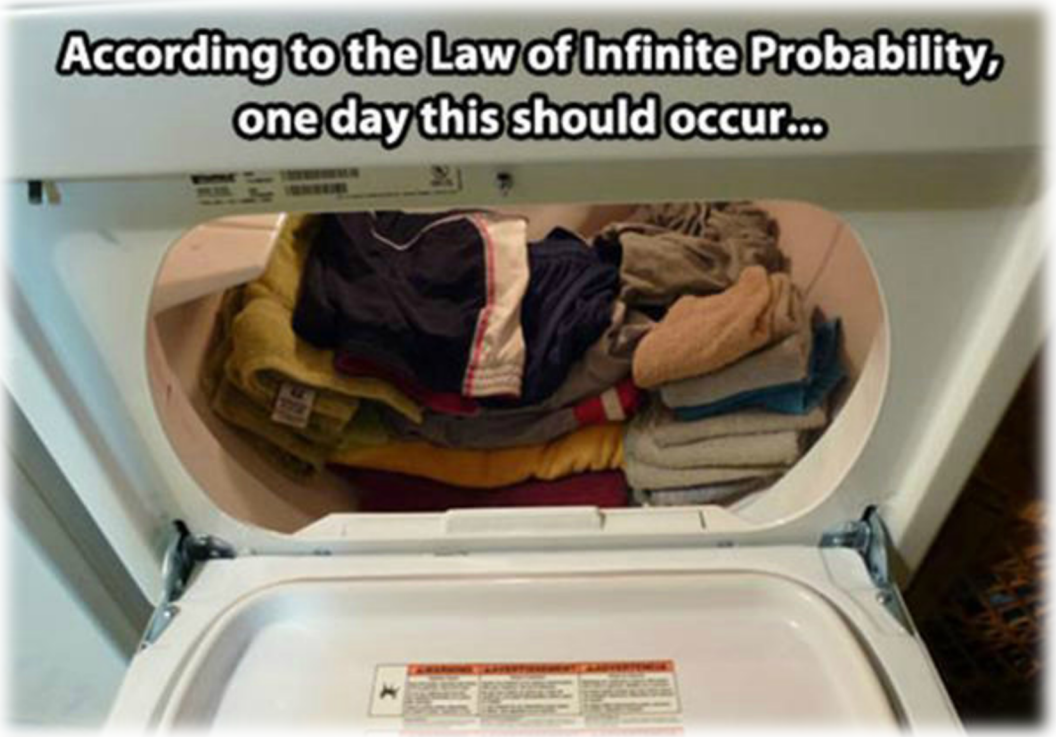
BUT THE WAITING TIME IS INFINITY!



Question 1

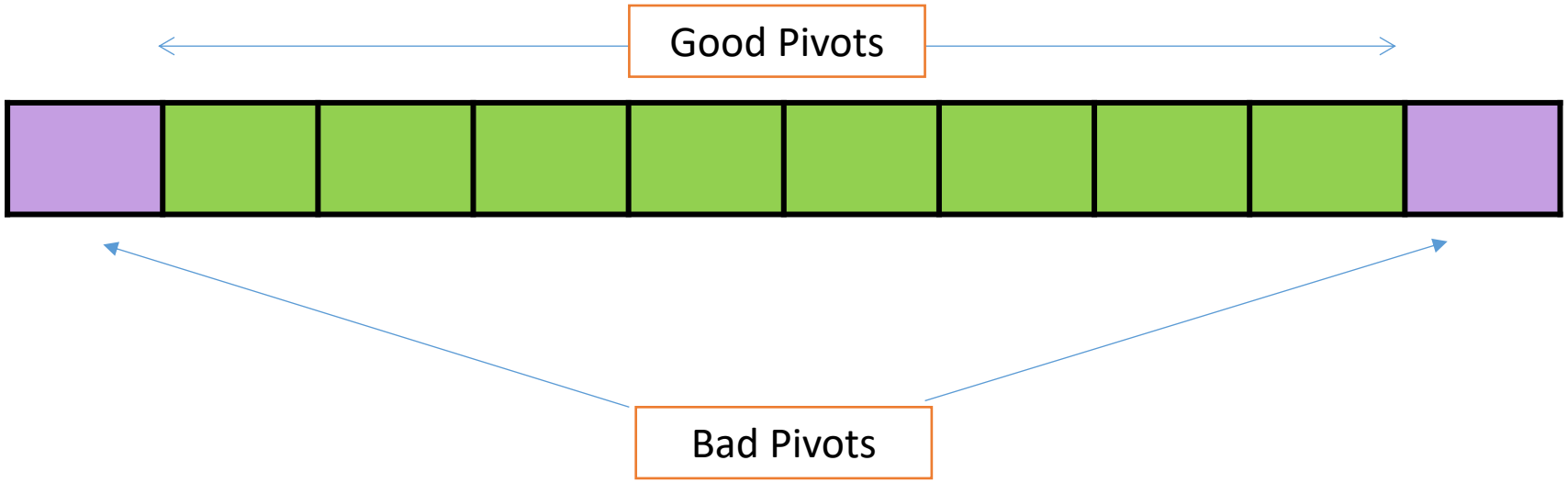
- How many repetitions until the partitioning is good?
- And we claim that
  - Only need to repeat $O(1)$ Time!
  - Yes, for n equals to any integer!





Question 1

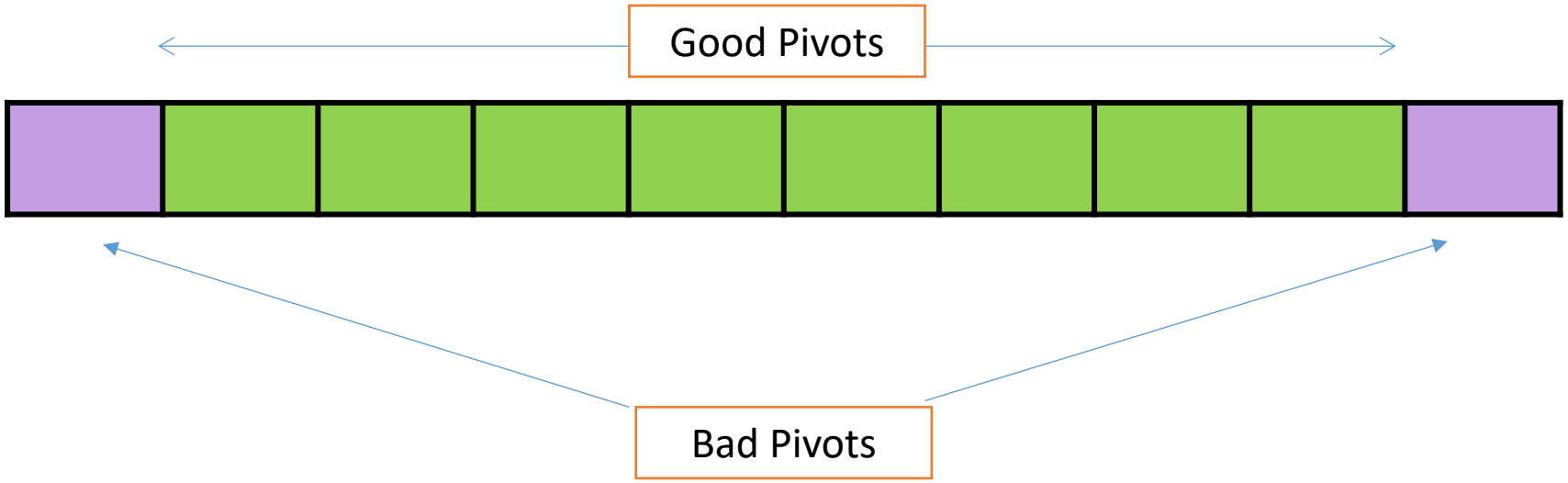
- How many repetitions until the partitioning is good?
- A pivot will make the partitioning good if
  - The pivot is larger than 10% of the numbers in the array, and
  - The pivot is smaller than 10% of the numbers in the array,
- If the array is sorted, and each of box below is 10% of the array



Question 1

- What is the probability of picking a good pivot randomly?
  - p = 8/10
- Will the probability be different when the list is not sorted?

- If the array is sorted, and each of box below is 10% of the array



Question 1

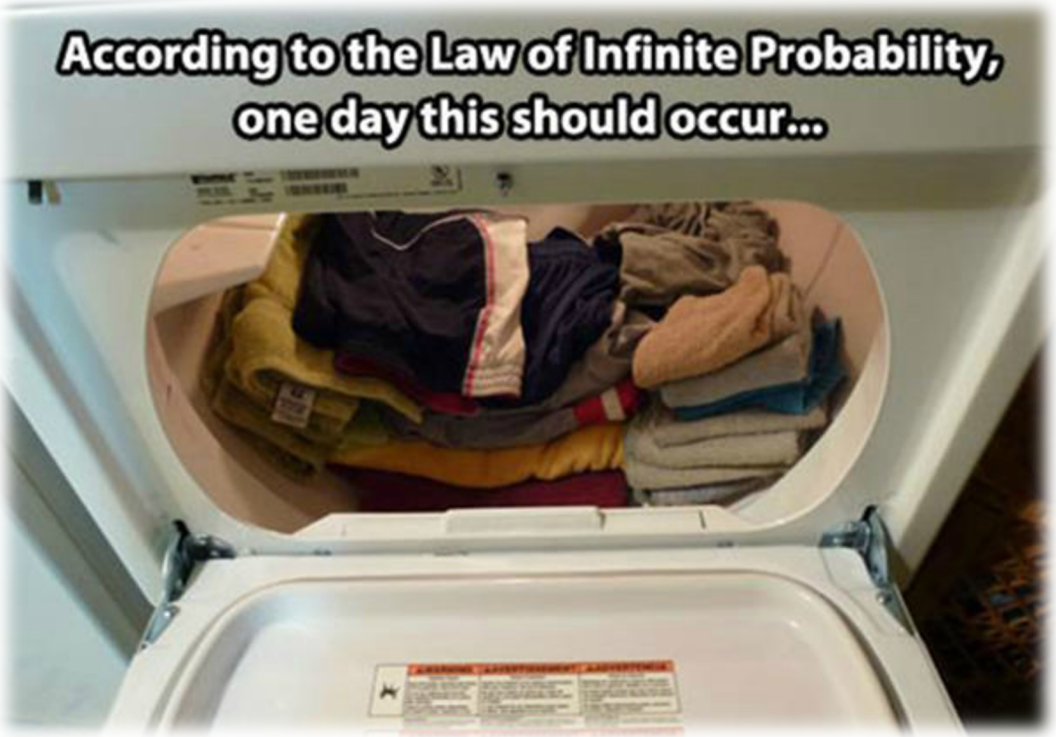
- How many repetitions until the partitioning is good?
- The probability of picking a good pivot is $p = 8/10$
- If we pick a pivot randomly, how many times we need to pick such that we finally end up with a good pivot?

$$\#\text{time} = 1/p = 10/8 = 1.25$$

- It means, if pick a pivot randomly for two times (> 1.25), we will expect one of them is a good pivot!

Question 1

- How many repetitions until the partitioning is good?
- And we claim that
  - Only need to repeat $O(1)$ Time!
  - Yes, for n equals to any integer!



Paranoid QuickSort

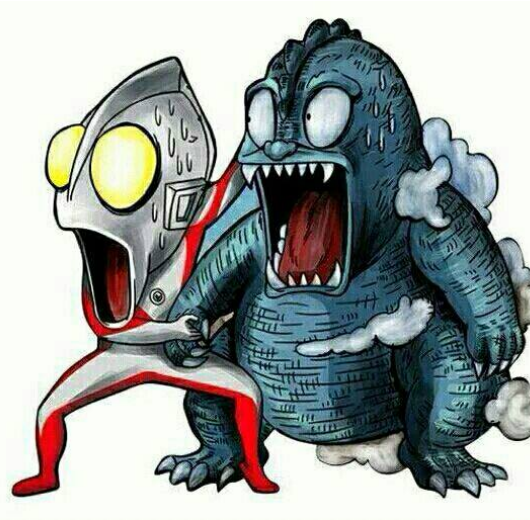
```
ParanoidQuickSort(A[1..n], n)
     if (n == 1) then return;
     else
          repeat
               pIndex = random(1, n)
               p = partition(A[1..n], n, pIndex)
          until p > (1/10)*n and p < (9/10)*r

          x = QuickSort(A[1..p-1], p-1)
          y = QuickSort(A[p+1..n], n-p)
```

Question 1

How many times we need to repeat?

We only need 2 = $O(1)$ times to expect a good pivot

Expected time = $O(n)$



Paranoid QuickSort

```
ParanoidQuickSort(A[1..n], n)
     if (n == 1) then return;
     else
          repeat
               pIndex = random(1, n)
               p = partition(A[1..n], n, pIndex)
          until p > (1/10)*n and p < (9/10)*n

          x = QuickSort(A[1..p-1], p-1)
          y = QuickSort(A[p+1..n], n-p)
```

Expected time = $O(n)$

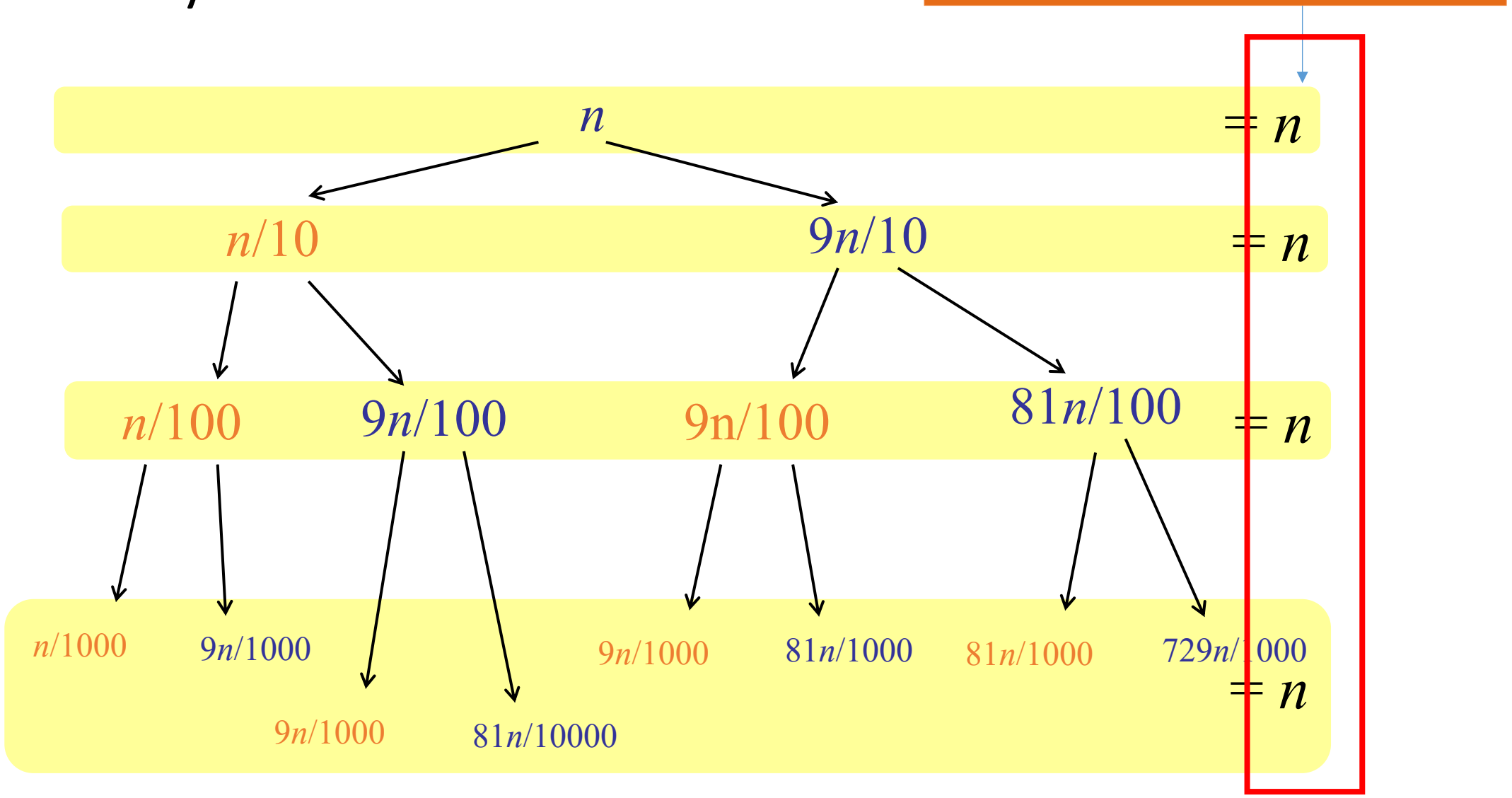
$$T(n) = cn + T(p) + T(n\text{-}p)$$

Paranoid QuickSort Time Complexity

- $T(n) = cn + T(p) + T(n\text{-}p)$
- And $p = n \times 1/10$ (or $p = n \times 9/10$)
- $T(n) = cn + T(n/10) + T(9n/10)$
- And we claim that

$$T(n) = \mathrm{O}(n \log n)$$

How many levels?

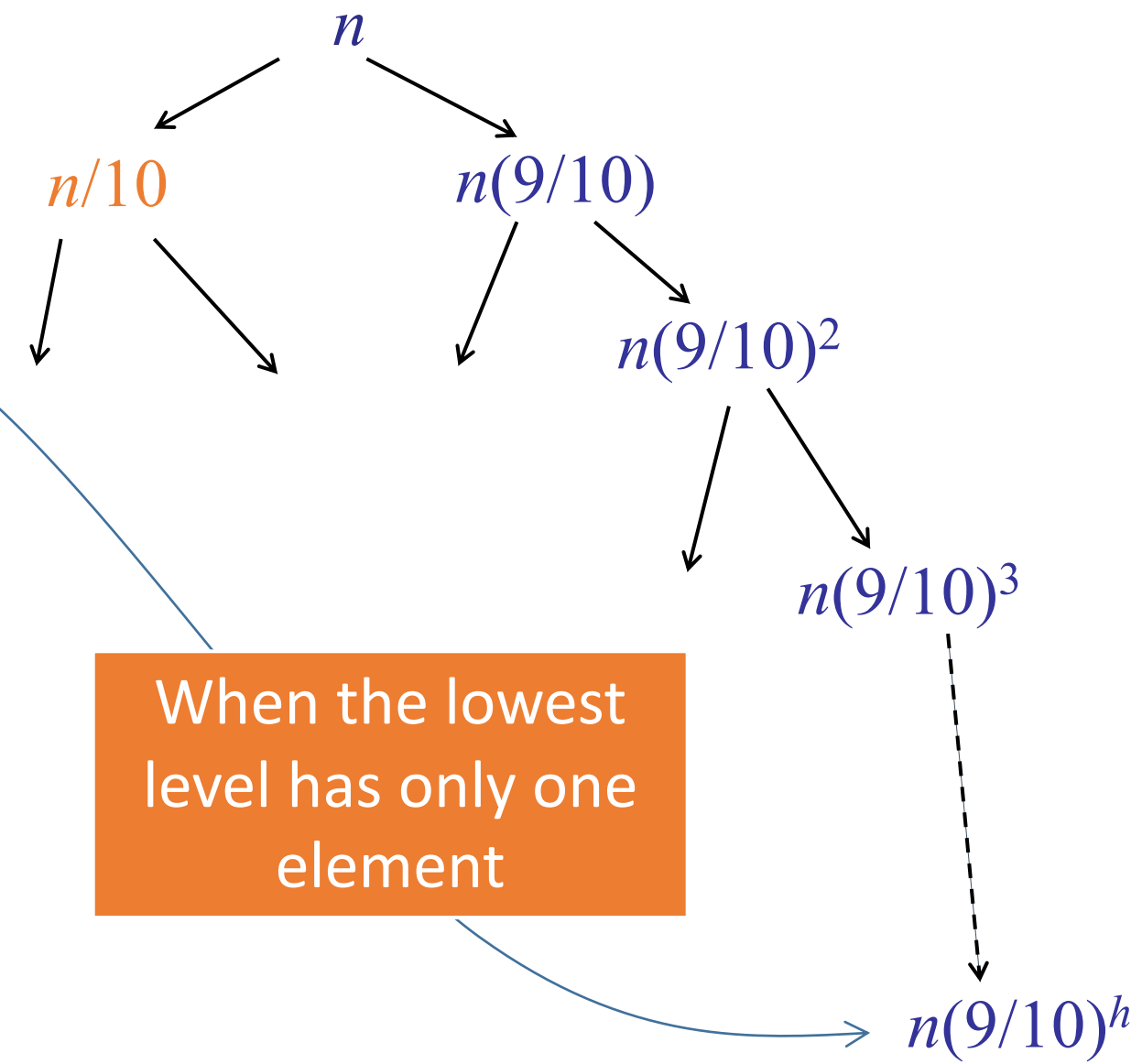


How many levels?

$$1 = n(9/10)^h$$

$$(10/9)^h = n$$

$$h = \log_{10/9}(n) = O(\log n)$$



QuickSort Summary

- If we could split the array $(1/10) : (9/10)$
- Good performance: $O(n \log n)$

Paranoid QuickSort

```
ParanoidQuickSort(A[1..n], n)
     if (n == 1) then return;
     else
          repeat
               pIndex = random(1, n)
               p = partition(A[1..n], n, pIndex)
          until p > (1/10)*n and p < (9/10)*n

          x = QuickSort(A[1..p-1], p-1)
          y = QuickSort(A[p+1..n], n-p)
```

Expected time = $O(n)$

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

QuickSort

- Key Idea:
  - Choose the pivot at random.
- Randomized Algorithms:
  - Algorithm makes decision based on randomness (coin flips)
  - Can “fool” the adversary (who provides bad input)
  - Running time is a random variable.

Randomization

Randomized algorithm:

- Algorithm makes random choices
- For every input, there is a good probability of success.

Average-case analysis:

- Algorithm (may be) deterministic
- “Environment” chooses random input
- Some inputs are good, some inputs are bad
- For most inputs, the algorithm succeeds

QuickSort Tips

- Optimize the partition routine
  - Most important aspect of a good QuickSort is partitioning.
- Choose a pivot carefully (e.g., at random)
  - Bad pivots lead to bad performance.
- Plan for arrays with duplicate values.
  - Equal elements can cause bad performance.

QuickSort Optimizations

- For small arrays, use InsertionSort.
  - Recursion has overhead.
  - QuickSort is slow on small arrays.
  - Idea: if the array is small, switch to InsertionSort

- Details:
  - Once recursion reaches a small array, use InsertionSort (instead of partition/recurse).
  - Once recursion reaches 8 elements, hand-code?

QuickSort Optimizations

- Two-pivot Quicksort
  - Recently shown that two pivots is faster than one!
  - Choose two pivots, partition around both.
  - What about three pivots? Four?
  - Experiment!

But aren’t all these still O(n log n)? What is the reason of being faster?

A. They bluff

B. Some can achieve O(n)

C. It’s about the constant c in O

D. They divided into O(n) partition

QuickSort Summary

- Algorithm basics: divide-and-conquer
- How to partition an array in O(n) time.
- How to choose a good pivot.
- Paranoid QuickSort.
- Randomized analysis.