UNIT 2:
Divide and Conquer

Quicksort Analysis

Quicksort

ALGORITHM $Quicksort(A[l..r])$

//Sorts a subarray by quicksort
//Input: A subarray $A[l..r]$ of $A[0..n-1]$,
// defined by its left and right indices l and r
//Output: Subarray $A[l..r]$ sorted in nondecreasing order
if $l < r$
$s \leftarrow Partition(A[l..r])$ //s is a split position
$Quicksort(A[l..s-1])$
$Quicksort(A[s+1..r])$

Quicksort...

ALGORITHM *Partition*$(A[l..r])$

//Partitions a subarray by using its first element as a pivot
//Input: A subarray $A[l..r]$ of $A[0..n-1]$, defined by its left and right
// indices l and r $(l < r)$
//Output: A partition of $A[l..r]$, with the split position returned as
// this function's value

```
// Assume min and max indices are low and high
pivot = a[l] // can do better
i = l+1, j = r
while (true) {
  while (a[i] < pivot) i++
  while (a[j] > pivot) j--
  if (i >= j) break
  swap(a, i, j)
}
swap(a, l, j)          // moves the pivot to the
                       // correct place
return j
```

NOTE:
Assumption: List has no duplicates.
If duplicates are allowed,
then use <= in the left to right scan

Quicksort algorithm analysis

1. input's size: **n –** number of elements to be sorted. (Assuming for simplicity that n is a power of 2)
2. basic operation: **comparison**
3. **worst, average, and best cases exists**
4. Let T(n) = number of times the basic operation is executed.

Quicksort : Best case

- **Balanced split: happens in the middle of array**
- number of key comparisons made before a partition is achieved is **n,** if the scanning indices cross over, **n-1** if they coincide

$C_{best}(n) = 2C_{best}(n/2) + n$ for $n > 1$, $C_{best}(1) = 0$

Using Master method:

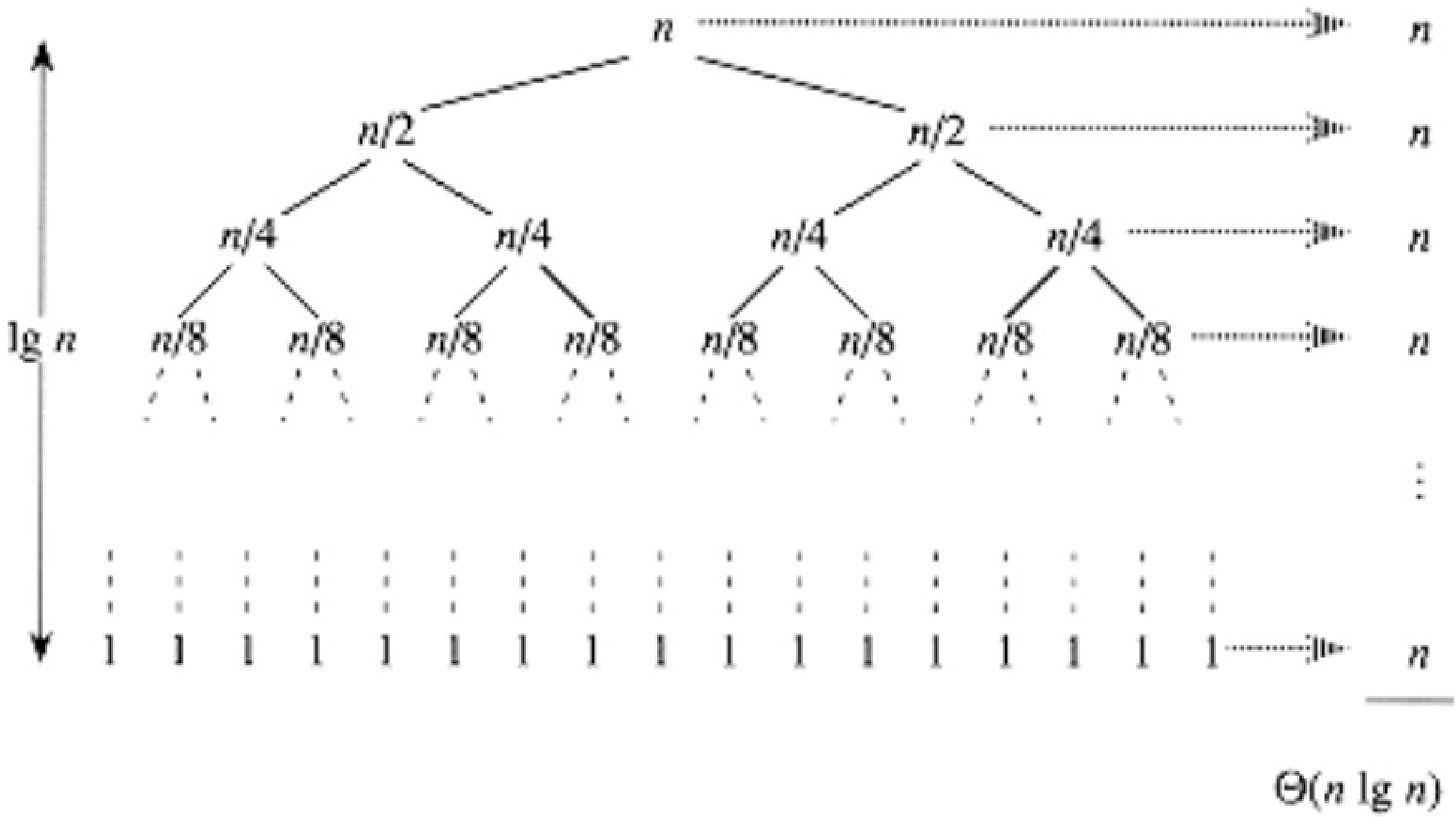
$a = 2$, $b = 2$, $f(n) = n$, $d=1$

$2 = 2^{1,}$ i.e., $a = b^d$

Case 3 of Master method holds good. Therefore

$C_{best} = \Theta(n^1 \log n) = \Theta(n \log n)$

$$T(n) \in \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log n) & \text{if } a = b^d \\ \Theta(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$



n
n/2
n/2
n/4
n/4
n/4
n/4
lg n
n/8
n/8
n/8
n/8
n/8
n/8
n/8
n/8
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
n
n
n
n
n
Θ(n lg n)

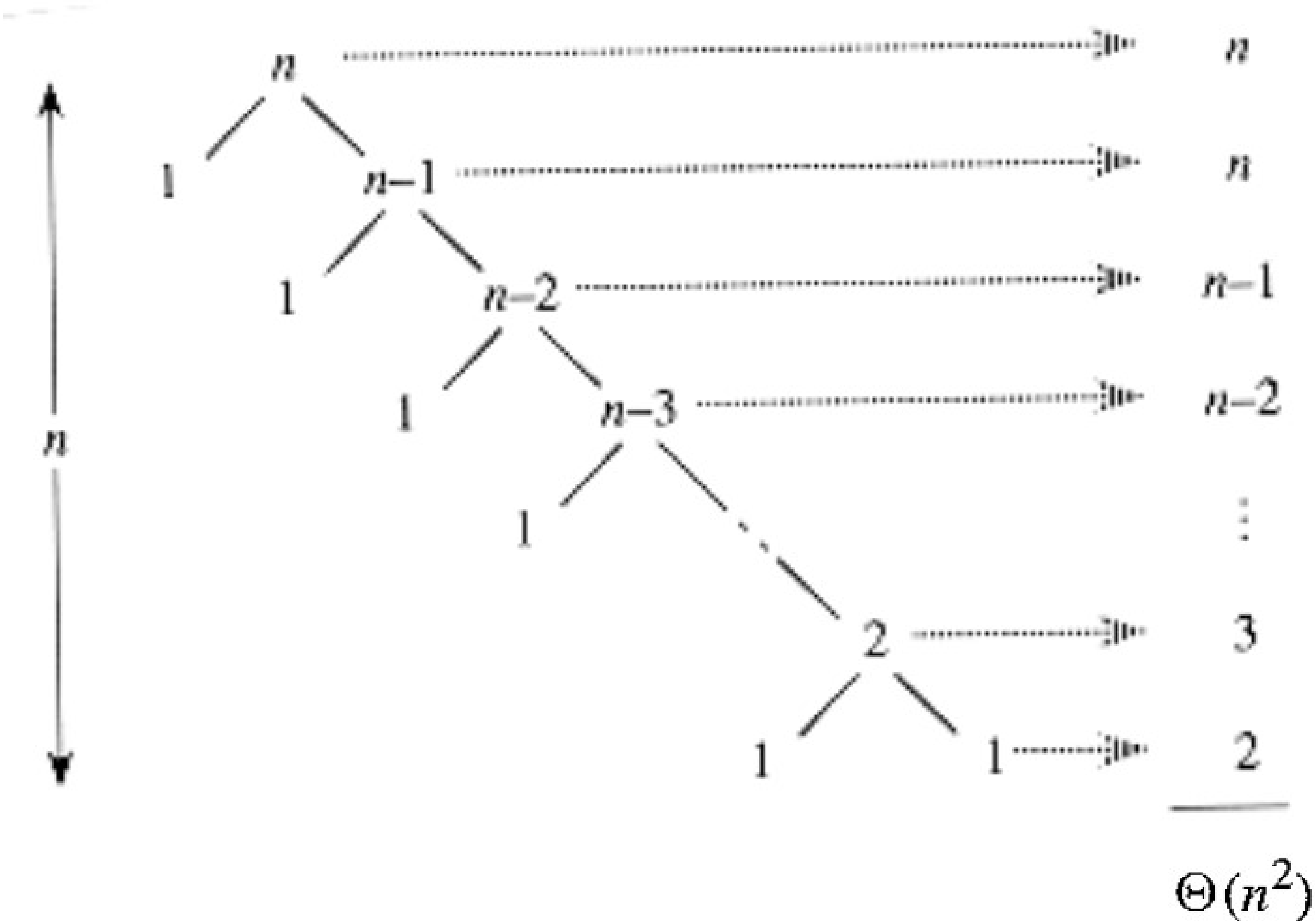
Quicksort : Worst case

- all the splits will be skewed to the extreme: one of the two subarrays will be **empty**, while the size of the other will be **(n-1)**
- The total number of key comparisons made will be equal to

$$C_{worst}(n) = n + (n-1) + \ldots + 3 + 2$$

$$\approx \frac{(n)(n+1)}{2} - 1$$

$$\in \Theta(n^2)$$



worst case running time of quicksort occurs when the input array is already completely sorted - a common situation in which insertion sort runs in O(n) time.

Quicksort : Average case

- A partition split can happen in any position s ($0 \le s \le n-1$) after n+1 comparisons are made to achieve the partition.
- After the partition, the left and right subarrays will have s and $n - 1 - s$ elements, respectively;
- Assuming that the partition split can happen in each position s with the same probability 1/n, we get the following recurrence relation.

Let $C_{avg}(n)$ be the average number of key comparisons made by quicksort on a randomly ordered array of size n.

$$C_{avg}(n) = \frac{1}{n}\sum_{s=0}^{n-1}[\, n + C_{avg}(s) + C_{avg}(n-1-s)] \quad \text{for } n > 1$$

$$C_{avg}(0) = 0$$

$$C_{avg}(1) = 0$$

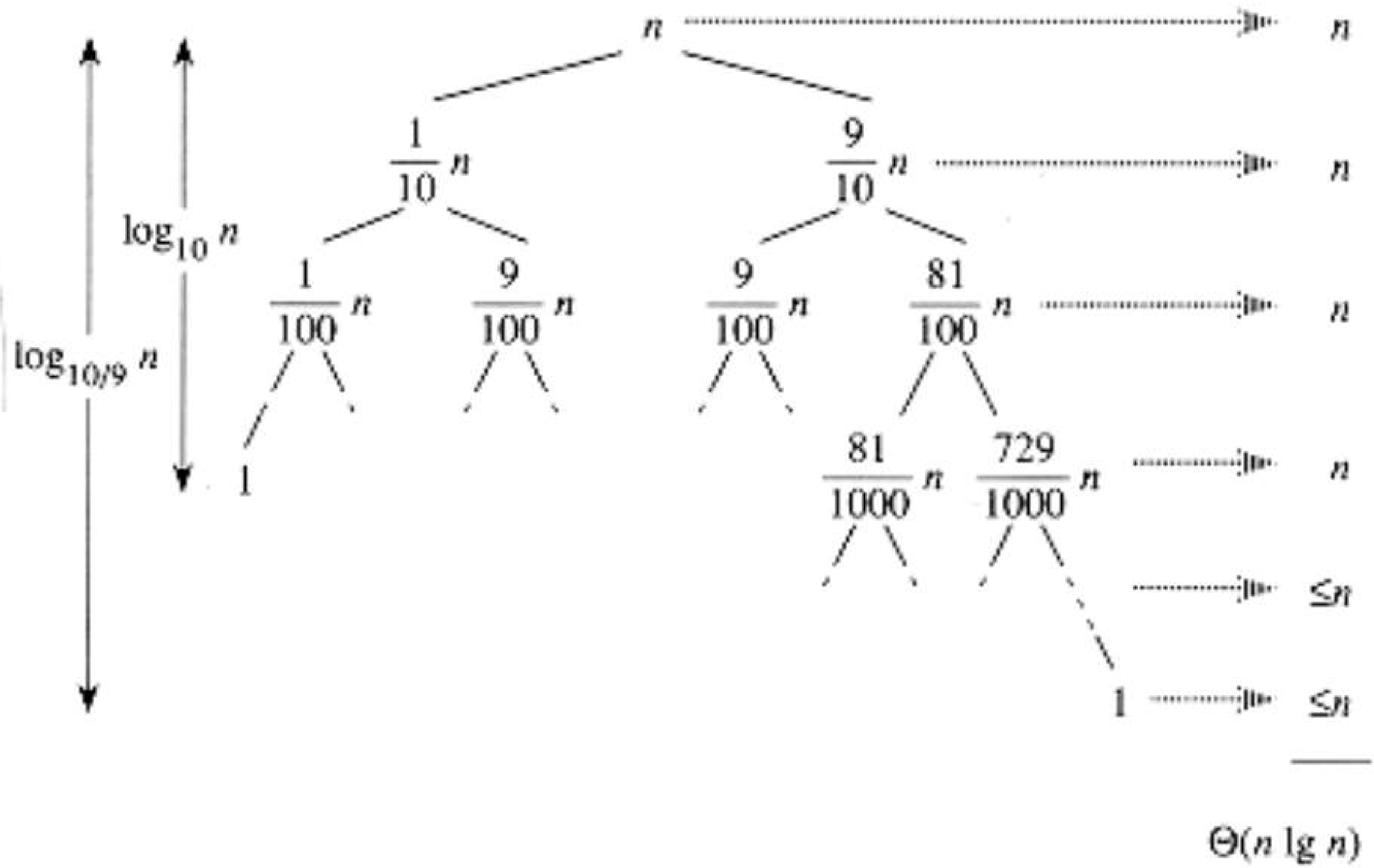
$$\boxed{C_{avg}(n) \approx 2n \ln n \approx 1.38 n \log_2 n}$$

Thus, on the average, quicksort makes only 39% more comparisons than in the best case

Let's check our understanding (1)

What is the running time of the Quicksort if the partitioning algorithm always produces a 9-to-1 proportional split?

$$T(n) = T(9n/10) + T(n/10) + n$$



n
n
$\frac{1}{10}n$
$\frac{9}{10}n$
n
$\log_{10} n$
$\log_{10/9} n$
$\frac{1}{100}n$
$\frac{9}{100}n$
$\frac{9}{100}n$
$\frac{81}{100}n$
n
1
$\frac{81}{1000}n$
$\frac{729}{1000}n$
n
$\leq n$
1
$\leq n$
$\Theta(n \lg n)$

Competitors for Quicksort

- **Heapsort**
  but its average running time is usually considered slower than in-place quicksort.
- **Introsort**
  variant of quicksort that switches to heapsort when a bad case is detected to avoid quicksort's worst-case running time.
- **Merge sort**
  stable sort, has excellent worst-case performance, works well on linked lists, good choice for external sorting of very large data sets stored on slow-to-access media such as disk storage or network-attached storage.
- **Bucket sort**
  with two buckets, it is very similar to quicksort; the pivot is effectively the value in the middle of the value range, which does well on average for uniformly distributed inputs.

Let’s check our understanding

QUIZ time!!!

Attempt the quiz using the given link:

https://forms.gle/rY4o2sxrkM2hA8ZC6

Time: 15 min

Marks: 10

No. of questions: 10