Design and Analysis of Algorithms

Quicksort: Analysis

Quick Sort - Characteristics

- Sorts almost in “place”,

  It does not require an additional array/memory.
- Very practical, average sort performance $\Theta(n \log n)$ (with small constant factors)

  But worst case $\Theta(n^2)$

Quick Sort - The Principle

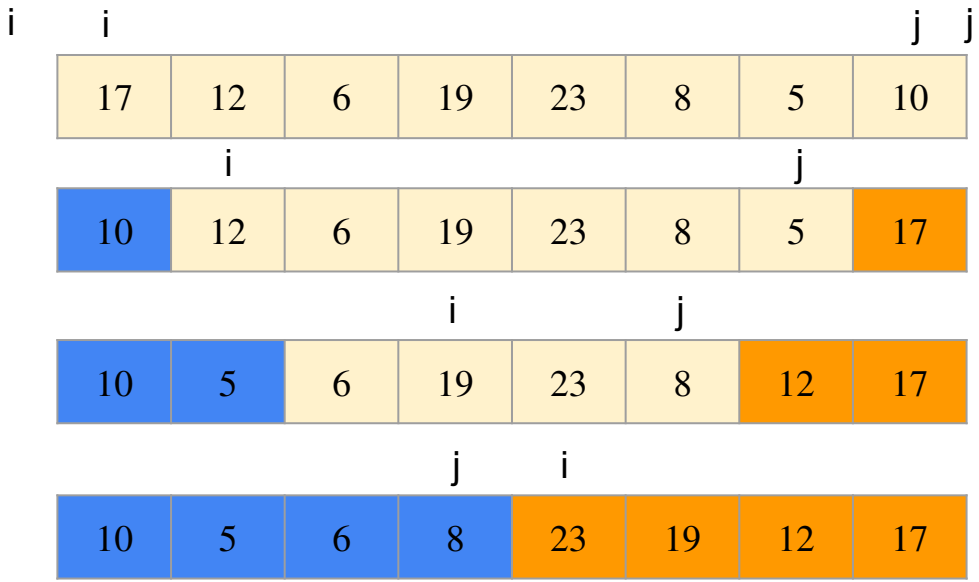
- To understand Quick Sort, let’s look at a high level description of the algorithm.
- A divide-and-conquer algorithm.
  - **Divide:** Partition array into 2 subarrays such that elements in the lower part <= elements in the higher part.
  - **Conquer:** Recursively sort the 2 subarrays.
  - **Combine:** Trivial since sorting is done in place.

Partitioning

- Linear Time Partitioning Algorithm
- Partitioning is done around the pivot element

$Partition(A, p, r)$

```
k = A[r];
i = p - 1;
j = r + 1;
while TRUE
    repeat j=j-1
    until A[j]<=k
    repeat i=i+1
    until A[i]>=k
    if i<j
        then exchange A[i] A[j]
    else
        return j
```



Time taken is $\Theta(n)$

Quick Sort Algorithm

- Initial call **Quicksort(A,1,length[A])**

```
Quicksort(A,p,r)
    if p < r
        then q = Partition(A,p,r)
        Quicksort(A,p,q)
        Quicksort(A,q+1,r)
```

Analysis of Quicksort

- Assume that all input points are distinct.
- The running time depends on the distribution of the split.

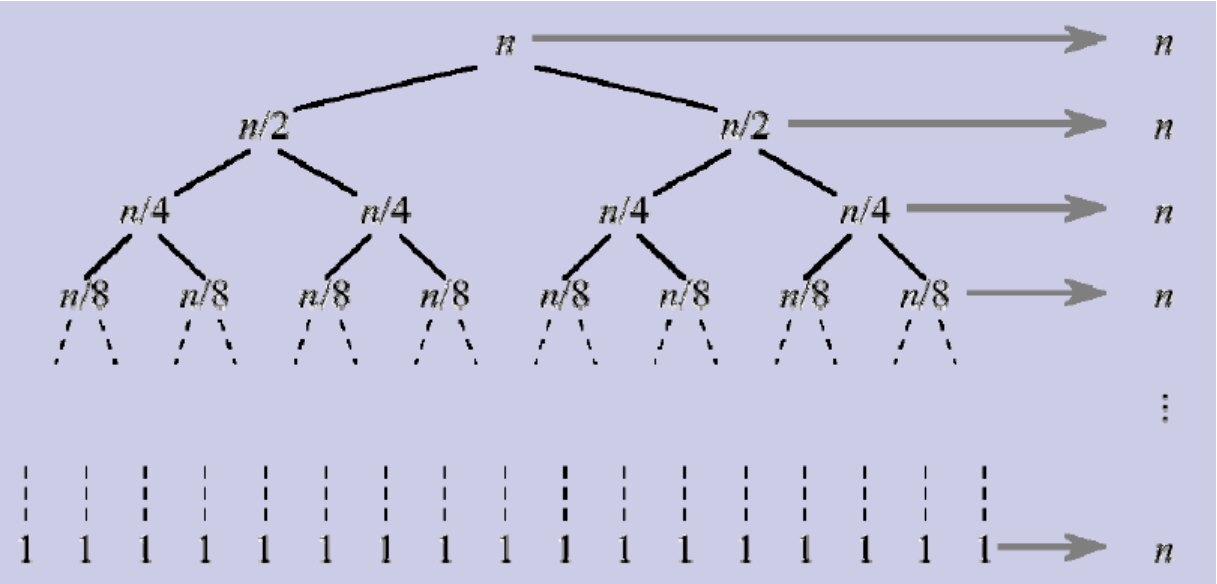
Best Case

- If we are lucky, Partition splits the array evenly

$$T(n) = 2T(n/2) + \Theta(n)$$

Time taken is

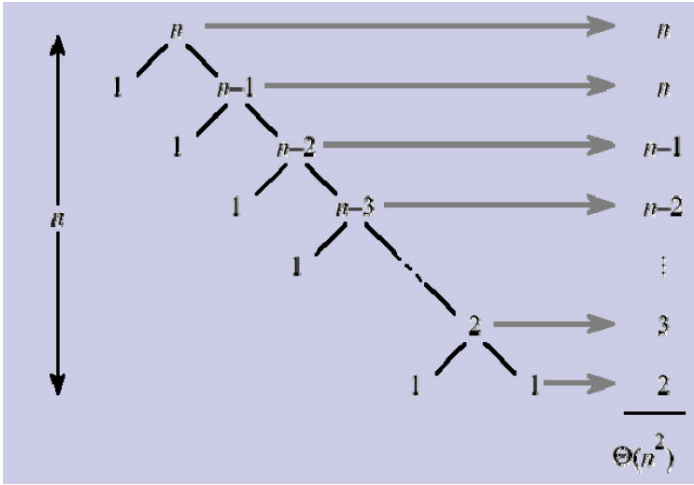
$\Theta(n \log n)$



Worst Case

- One side of the partition has only one elements.

$$
\begin{aligned}
T(n) &= T(1) + T(n-1) + \Theta(n) \\
&= T(n-1) + \Theta(n) \\
&= \Sigma_{k=1}^{n} \Theta(k) \\
&= \Theta(\Sigma_{k=1}^{n} k) \\
&= \Theta(n^2)
\end{aligned}
$$



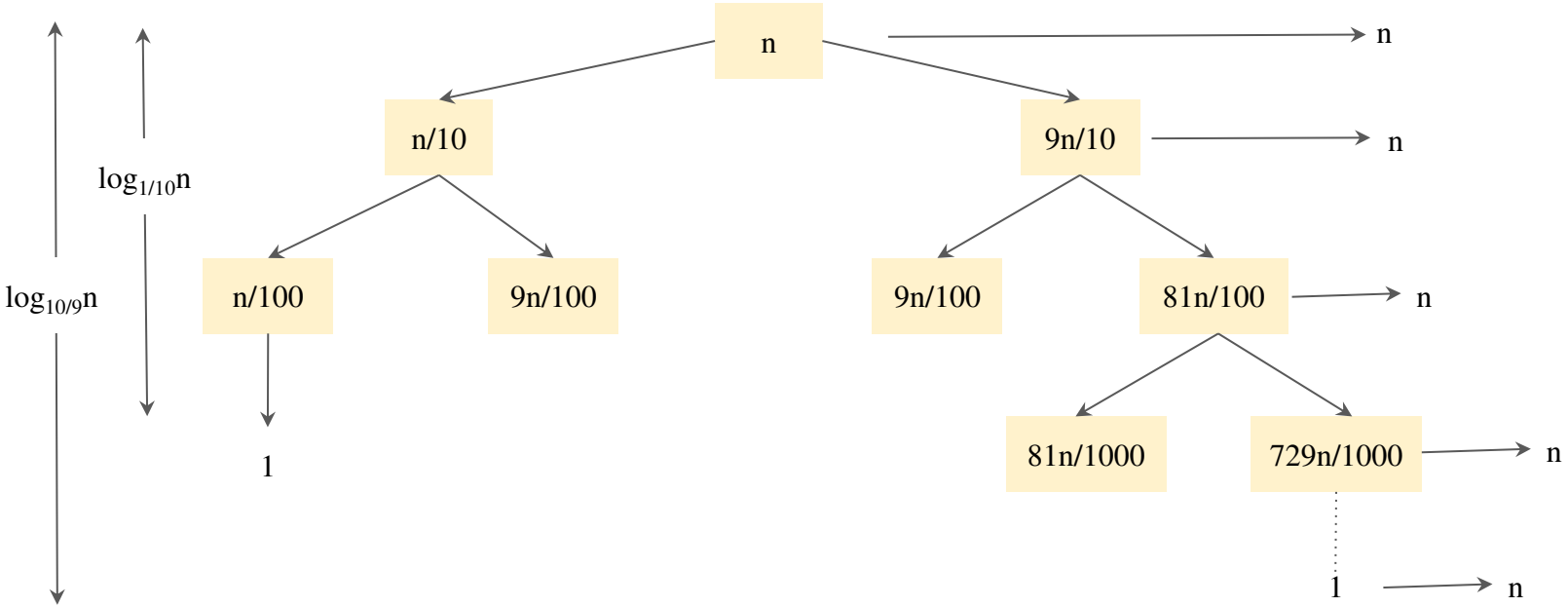
Worst Case

- When does the worst case appear?
  - Input is already sorted.
  - Input is reverse sorted.
- Same recurrence for the worst case of insertion sort.
- However, sorted input yields the best case for insertion sort.

Analysis of Quicksort

- Suppose the split is 1/10 : 9/10

$$T(n) = T(n/10) + T(9n/10) + \Theta(n) = \Theta(n \log n)!$$



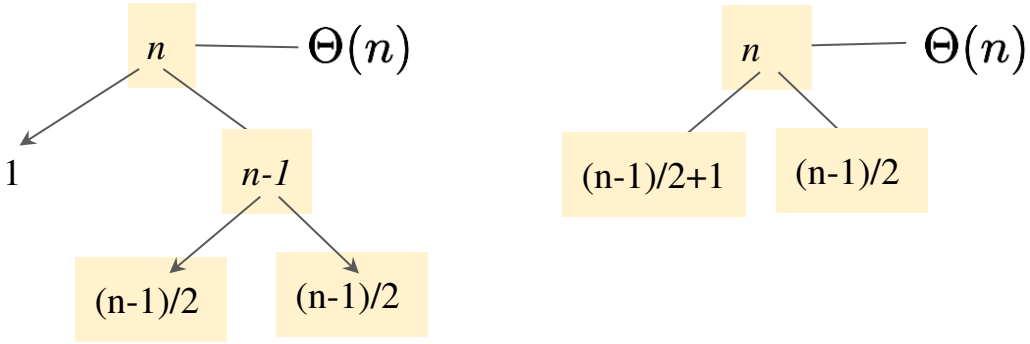
An Average Case Scenario

- Suppose, we alternate lucky and unlucky cases to get an average behaviour.

$$L(n) = 2U(n/2) + \Theta(n) \;\; lucky$$
$$U(n) = L(n-1) + \Theta(n) \;\; unlucky$$

we consequently get

$$\begin{aligned} L(n) &= 2(L(n/2-1)) + \Theta(n/2) + \Theta(n) \\ &= 2(L(n/2-1)) + \Theta(n) \\ &= \Theta(n \log n) \end{aligned}$$



An Average Case Scenario

- How can we make sure that we are usually lucky?
  - Partition around the middle ($n/2^{th}$) element.
  - Partition around a random element (works well in practise).
- Randomised Algorithm
  - Running time is independent of the input ordering.
  - No specific input triggers worst-case behaviour.
  - The worst case is only determined by the output of the random-number generator.

- Worst case running time of quicksort is $\Theta(n^2)$
- Best case running time of quicksort is $\Theta(n \log n)$
- Expected running time of quicksort is $\Theta(n \log n)$

What does expected running time mean?

- The running time of quicksort does not depend on the input. It depends on the random number provided by the generator.
- The average time taken over many different runs of the program would give us expected time.
- Same as saying that we are taking average over all possible random numbers sequences provided by the generator.

Thank you