Quick Sort

- Characteristics
  - sorts almost in "place," i.e., does not require an additional array
  - very practical, average sort performance O(n log n) (with small constant factors), but worst case O(n^2)

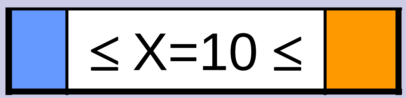
Quick Sort – the Principle

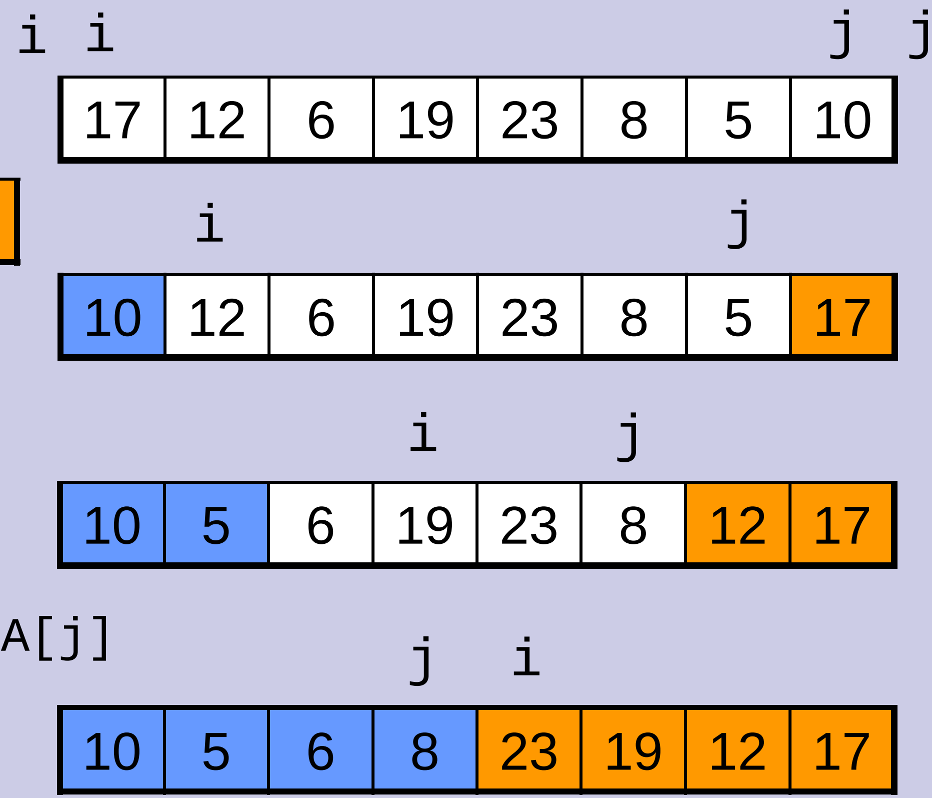
- To understand quick-sort, let's look at a high-level description of the algorithm
- A divide-and-conquer algorithm
  - **Divide**: partition array into 2 subarrays such that elements in the lower part <= elements in the higher part
  - **Conquer**: recursively sort the 2 subarrays
  - **Combine**: trivial since sorting is done in place

Partitioning

- Linear time partitioning procedure

```
Partition(A,p,r)
x←A[r]
i←p-1
j←r+1
while TRUE
    repeat j←j-1
        until A[j] ≤x
    repeat i←i+1
        until A[i] ≥x
    if i<j
        then exchange A[i]↔A[j]
        else return j
```





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)
```

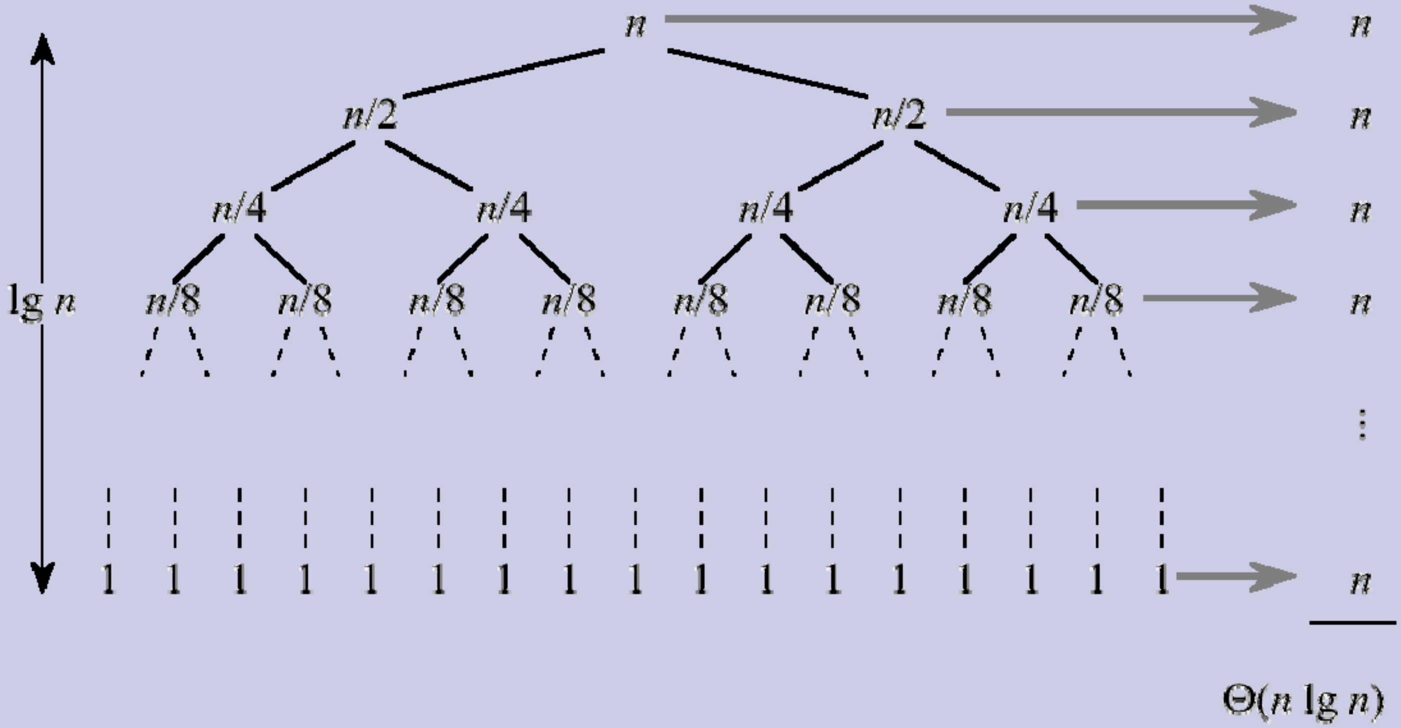
Analysis of Quicksort

- Assume that all input elements are distinct
- The running time depends on the distribution of splits

Best Case

- If we are lucky, Partition splits the array evenly

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

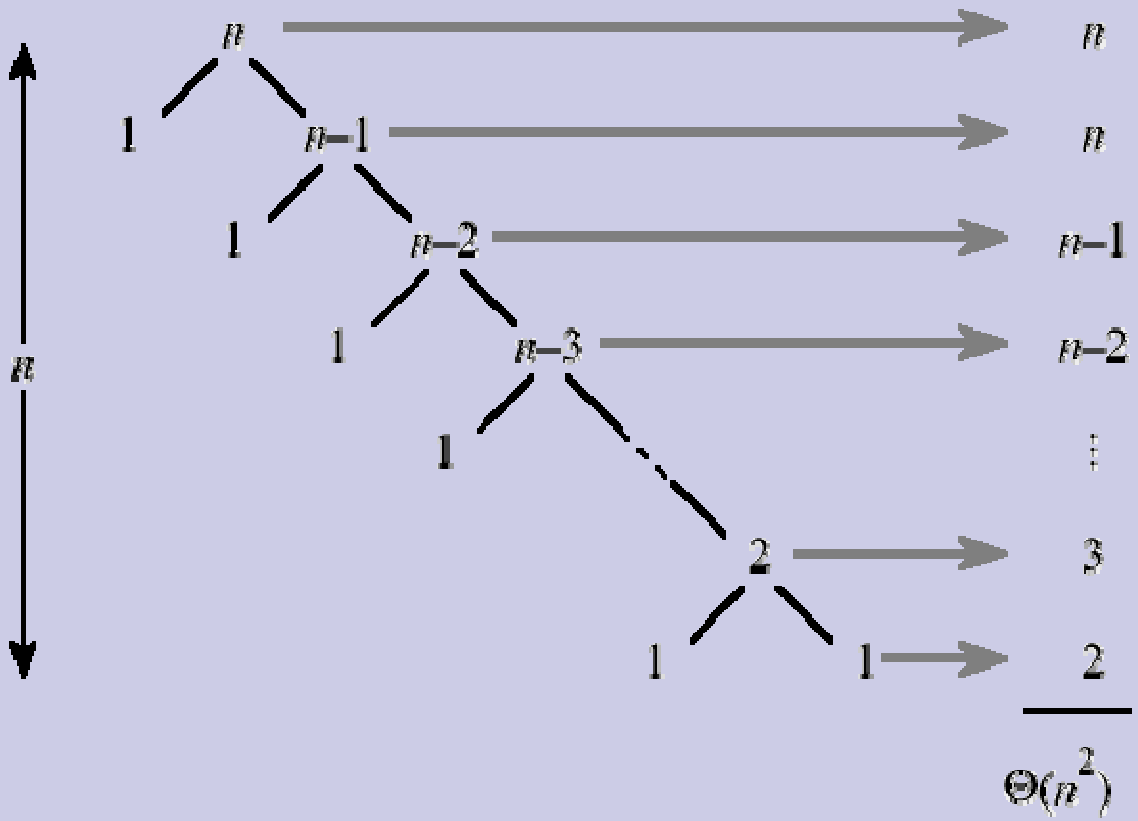


Worst Case

- What is the worst case?
- One side of the parition has only one element

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

Worst Case (2)



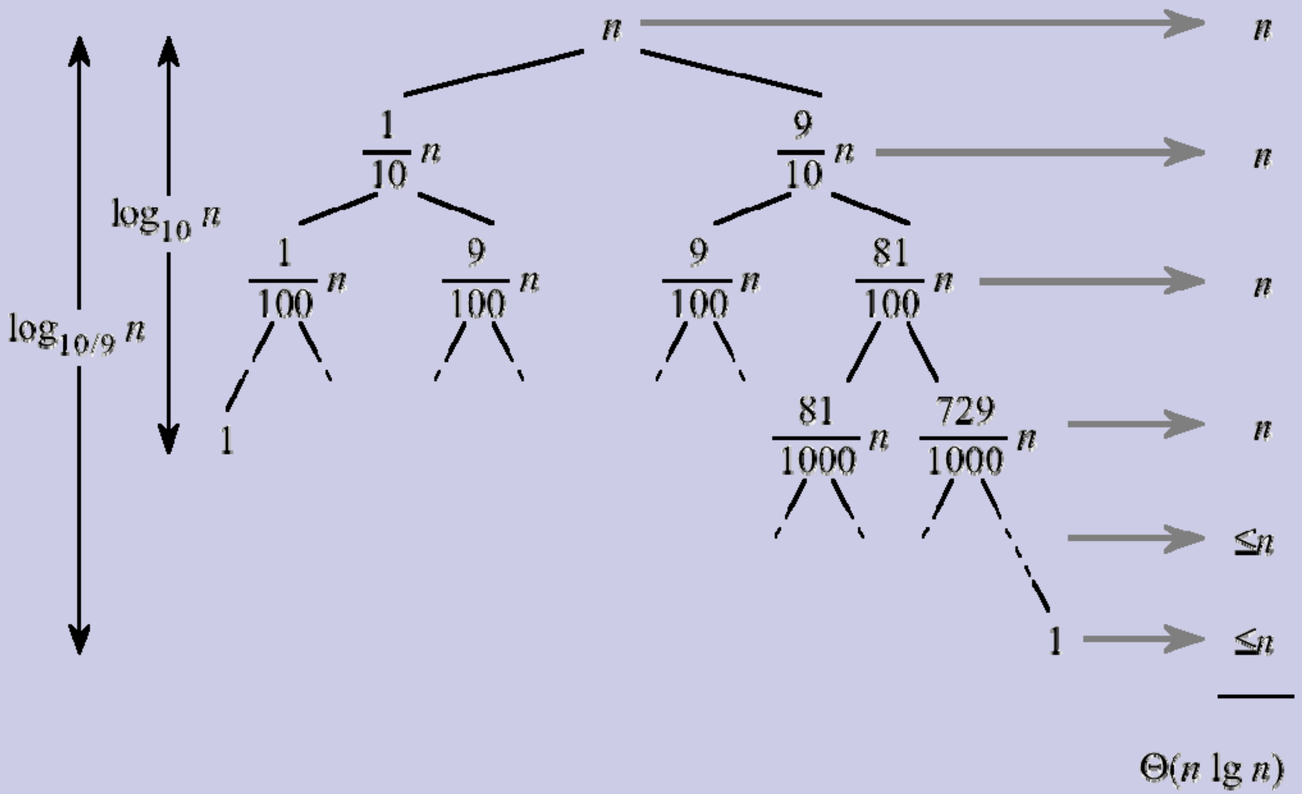
Worst Case (3)

- When does the worst case appear?
  - input is sorted
  - input 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

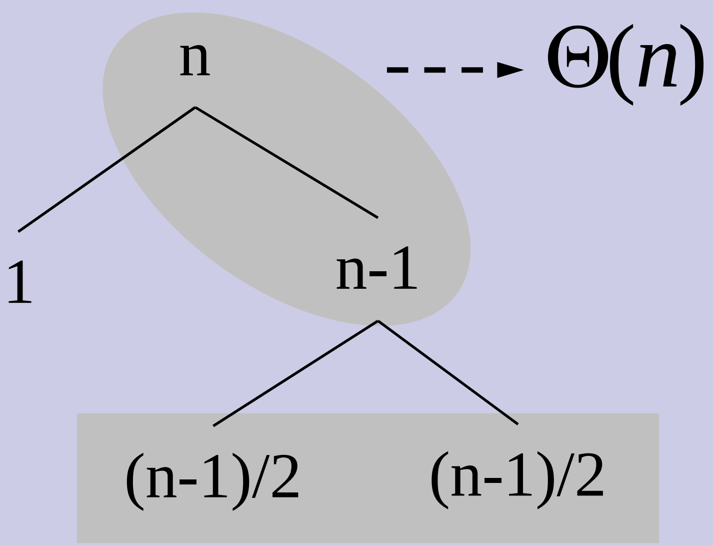
- Suppose, we alternate lucky and unlucky cases to get an average behavior

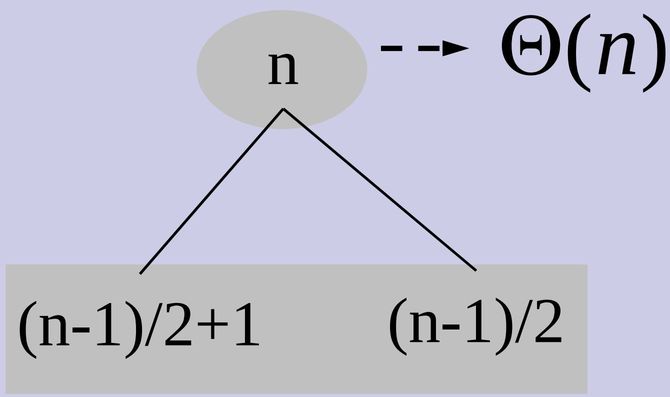
$$L(n) = 2U(n/2) + \Theta(n) \quad \text{lucky}$$

$$U(n) = L(n-1) + \Theta(n) \quad \text{unlucky}$$

we consequently get

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





An Average Case Scenario (2)

- How can we make sure that we are usually lucky?
  - Partition around the ”middle” (n/2th) element?
  - Partition around a random element (works well in practice)
- Randomized algorithm
  - running time is independent of the input ordering
  - no specific input triggers worst-case behavior
  - the worst-case is only determined by the output of the random-number generator

Randomized Quicksort

- Assume all elements are distinct
- Partition around a random element
- Consequently, all splits (1:n-1, 2:n-2, ..., n-1:1) are equally likely with probability 1/n

- Randomization is a general tool to improve algorithms with bad worst-case but good average-case complexity

Randomized Quicksort (2)

Randomized-Partition(A,p,r)

```
i ← Random(p,r)
exchange A[r] ↔ A[i]
return Partition(A,p,r)
```

Randomized-Quicksort(A,p,r)

```
if p < r then
   q ← Randomized-Partition(A,p,r)
   Randomized-Quicksort(A,p,q)
   Randomized-Quicksort(A,q+1,r)
```

Randomized Quicksort Analysis

- Let T(n) be the expected number of comparisons needed to quicksort n numbers.
- Since each split occurs with probability 1/n, T(n) has value T(i-1)+T(n-i)+n-1 with probability 1/n.
- Hence,

$$
\begin{aligned}
T(n) &= \frac{1}{n}\sum_{j=1}^{n}\left(T(j-1)+T(n-j)+n-1\right) \\
&= \frac{2}{n}\sum_{j=0}^{n-1} T(j)+n-1
\end{aligned}
$$

Randomized Quicksort Analysis(2)

- We have seen this recurrence before.
- It is the recurrence for the expected number of comparisons required to insert a randomly chosen permutation of n elements.
- We proved that $T(n) = O(n\log_2 n)$.
- Hence expected number of comparisons required by randomized quicksort is $O(n\log_2 n)$

Randomized Quicksort running times

- Worst case running time of quicksort is $O(n^2)$
- Best case running time of quicksort is $O(n\log_2 n)$
- Expected running time of quicksort is $O(n\log_2 n)$

What does expected running time mean?

- The running time of quicksort does not depend on the input. It depends on the random numbers provided by the generator.
- Thus for the same input the program might take 3sec today and 5sec tomorrow.
- The average time taken over many different runs of the program would give us the expected time.
- Same as saying that we are taking average over all possible random number sequences provided by the generator.

Analysis of insertion in BST

- When creating a binary search tree on n elements the running time does depend on the order of the elements.
- Our algorithm for insertion did not employ an random bits.
- Given a specific input order the algorithm takes the same time each day.
- However, the time taken is different for different input orders.
- The average time taken over all possible input orders is $O(n\log_2 n)$.