COMP251: Probabilistic analysis

Jérôme Waldispühl & Giulia Alberini
School of Computer Science
McGill University

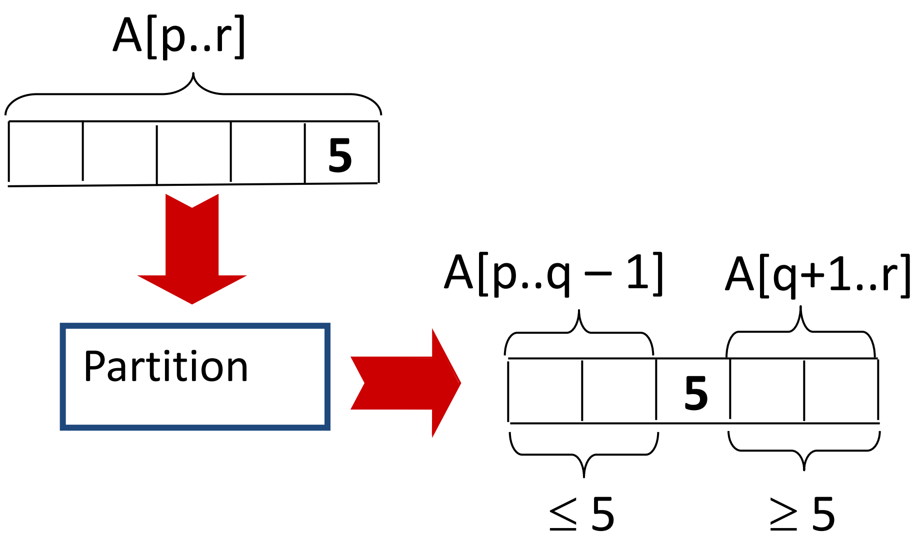
Based on slides from Lin & Devi (UNC)

Review of Quicksort

QuickSort: Review

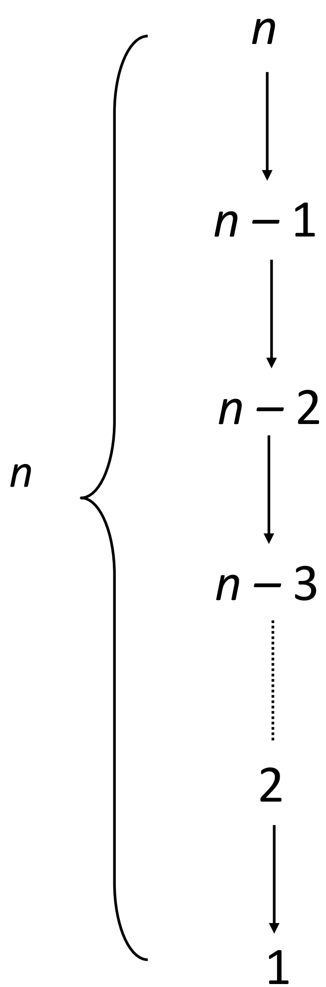
```
Quicksort(A, p, r)
    if p < r then
        q := Partition(A, p, r);
        Quicksort(A, p, q – 1);
        Quicksort(A, q + 1, r)
    fi
```

```
Partition(A, p, r)
    x, i  := A[r], p – 1;
    for j := p to r – 1 do
        if A[j] ≤ x then
            i := i + 1;
            A[i] ↔ A[j]
        fi
    od;
    A[i + 1] ↔ A[r];
    return i + 1
```



Worst-case Partition Analysis

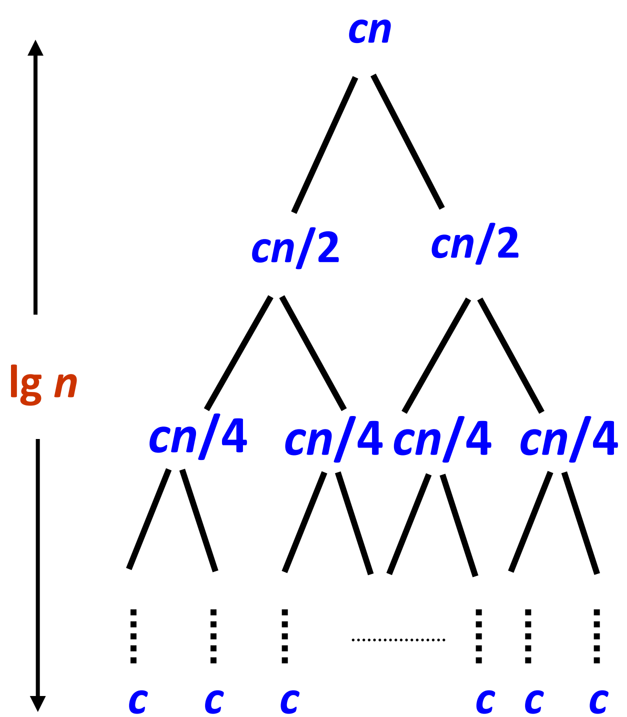
Recursion tree for worst-case partition



Split off a single element at each level:

$$T(n) = T(n - 1) + T(0) + \text{PartitionTime}(n)$$
$$= T(n - 1) + \Theta(n)$$
$$= \sum_{k=1 \text{ to } n} \Theta(k)$$
$$= \Theta\left(\sum_{k=1 \text{ to } n} k\right)$$
$$= \Theta(n^2)$$

Best-case Partitioning



- Each subproblem size $\leq n/2$.
- Recurrence for running time
  - $T(n) \leq 2T(n/2) + \text{PartitionTime}(n)$
    $= 2T(n/2) + \Theta(n)$
- $T(n) = \Theta(n \lg n)$

Variations

- Quicksort is not very efficient on small lists.
- This is a problem because Quicksort will be called on lots of small lists.
- **Fix 1:** Use Insertion Sort on small problems.
- **Fix 2:** Leave small problems unsorted. Fix with one final Insertion Sort at end.

Why? Insertion Sort is very fast on almost-sorted lists.

Average case analysis

Unbalanced Partition Analysis

What happens if we get poorly-balanced partitions,

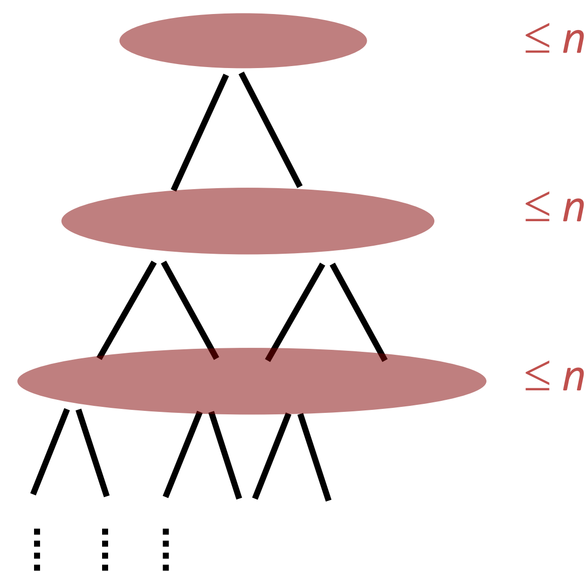
e.g., something like: $T(n) \leq T(9n/10) + T(n/10) + \Theta(n)$?

Still get $\Theta(n \lg n)$!! (As long as the split is of constant proportionality)

Intuition: Can divide n by $c > 1$ only $\Theta(\lg n)$ times before getting 1.

$$n \downarrow n/c \downarrow n/c^2 \downarrow \vdots \downarrow 1 = n/c^{\log_c n}$$

Roughly $\log_c n$ levels;
Cost per level is $O(n)$.



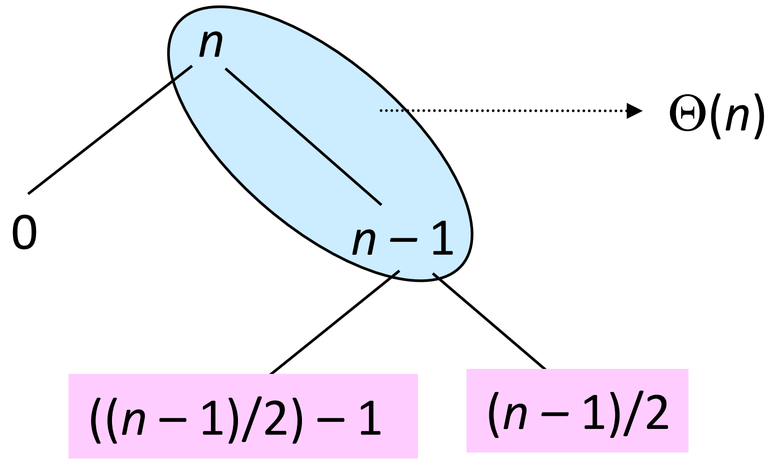
Note: Different base logs are related by a constant.

Intuition for the Average Case

- Partitioning is unlikely to happen in the same way at every level.
  - Split ratio is different for different levels. (Contrary to our assumption in the previous slide.)
- Partition produces a mix of "good" and "bad" splits, distributed randomly in the recursion tree.

What is the running time likely to be in such a case?

Intuition for the average case

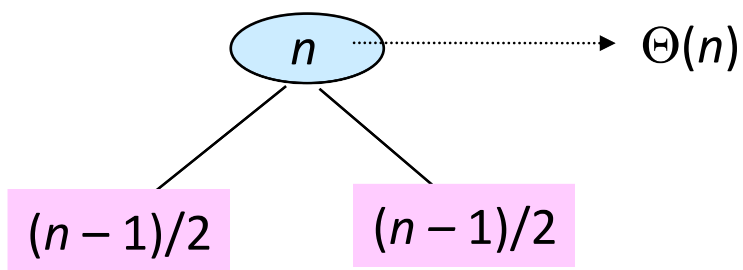


Bad split followed by a good split:

Produces subarrays of sizes 0, $(n-1)/2-1$, and $(n-1)/2$.

Cost of partitioning :

$\Theta(n) + \Theta(n\text{-}1) = \Theta(n)$.



Good split at the first level:

Produces two subarrays of size $(n-1)/2$.

Cost of partitioning :

$\Theta(n)$.

Situation at the end of case 1 is not worse than that at the end of case 2.

When splits alternate between good and bad, the cost of bad split can be absorbed into the cost of good split.

Thus, the running time is $O(n \lg n)$, though with larger hidden constants.

Randomized quicksort

Randomized Quicksort

- Want to make running time independent of input ordering.
- How can we do that?
  - » Make the algorithm randomized.
  - » Make every possible input equally likely.
    - Can randomly shuffle to permute the entire array.
    - For quicksort, it is sufficient if we can ensure that every element is equally likely to be the *pivot*.
    - So, we choose an element in $A[p..r]$ and exchange it with $A[r]$.
    - Because the *pivot* is randomly chosen, we expect the partitioning to be well balanced on average.

Variations (Continued)

- Input distribution may not be uniformly random.
- **Fix 1:** Use "randomly" selected pivot.
  - We will analyze this in detail.
- **Fix 2:** Median-of-three Quicksort.
  - Use median of three fixed elements (say, the first, middle, and last) as the pivot.
  - To get $O(n^2)$ behavior, we must continually be unlucky to see that two out of the three elements examined are among the largest or smallest of their sets.

Randomized Version

Want to make running time independent of input ordering.

```
Randomized-Partition(A, p, r)
    i := Random(p, r);
    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 – 1);
        Randomized-Quicksort(A, q + 1, r)
    fi
```

Average case analysis

Average Case Analysis of **Randomized Quicksort**

Random Variable X = # comparisons over all calls to Partition.

Why is it a good measure? Intuition: there are at most n calls to Partition. The running time of these calls is upper bounded by the number of comparisons.

Notation:

- Let $z_1, z_2, \ldots, z_n$ denote the list items (in sorted order).
- Let $Z_{ij} = \{z_i, z_{i+1}, \ldots, z_j\}$.

Let RV $X_{ij} = \begin{cases} 1 & \text{if } z_i \text{ is compared to } z_j \\ 0 & \text{otherwise} \end{cases}$

X_{ij} is an **indicator random variable**.
$X_{ij} = I\{z_i \text{ is compared to } z_j\}$.

Thus, $$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}.$$

Analysis (Continued)

We have:

$$E[X] = E\left[\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} X_{ij}\right]$$

$$= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} E[X_{ij}]$$

$$= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} P[z_i \text{ is compared to } z_j]$$

Reminder:

$E[X_{ij}] = 0 \cdot P[X_{ij}=0] + 1 \cdot P[X_{ij}=1]$

$= P[X_{ij}=1]$

So, all we need to do is to compute P[z_i is compared to z_j].

Analysis (Continued)

Claim: z_i and z_j are compared iff the first element to be chosen as a pivot from Z_{ij} is either z_i or z_j.

Hint: what happens if an element in Z_{ij} other than z_i or z_j is chosen as pivot first?

Exercise: Prove this.

So, $$\begin{aligned} \mathrm{P}[z_i \text{ is compared to } z_j] &= \mathrm{P}[z_i \text{ or } z_j \text{ is first pivot from } Z_{ij}] \\ &= \mathrm{P}[z_i \text{ is first pivot from } Z_{ij}] \\ &\quad + \mathrm{P}[z_j \text{ is first pivot from } Z_{ij}] \\ &= \frac{1}{j-i+1} + \frac{1}{j-i+1} \\ &= \frac{2}{j-i+1} \end{aligned}$$

We choose the pivot uniformly at random

Analysis (Continued)

Therefore,

$$E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j - i + 1}$$

$$= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k + 1}$$

Substitute $k = j - i$.

$$< \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k}$$

$$= \sum_{i=1}^{n-1} O(\lg n)$$

$$\sum_{k=1}^{n} \frac{1}{k} = H_n$$

$$H_n = \ln n + O(1)$$

(n^{th} Harmonic number)

$$= O(n \lg n).$$

Deterministic vs. Randomized Algorithms

- Deterministic Algorithm : Identical behavior for different runs for a given input.
- Randomized Algorithm : Behavior is generally different for different runs for a given input.