COMP251: Probabilistic analysis

Jérôme Waldispühl

School of Computer Science

McGill University

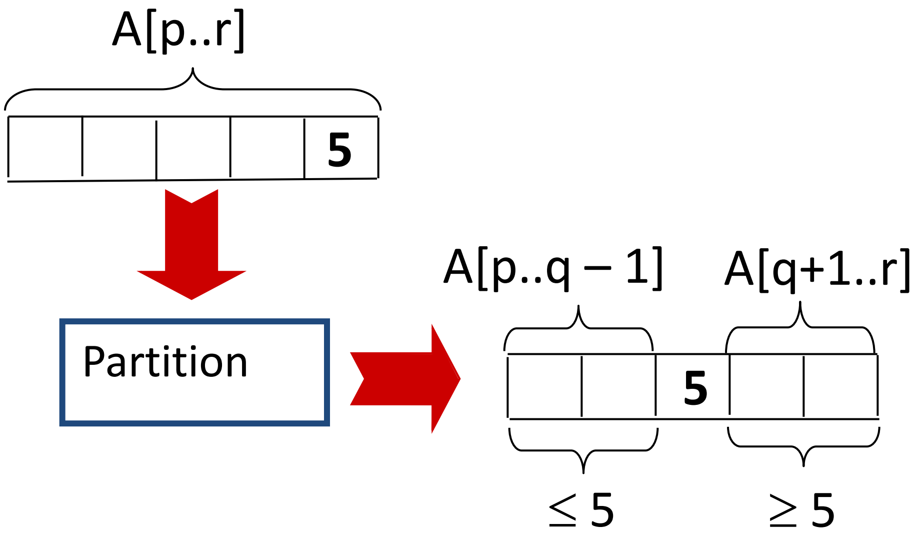
Based on slides from Lin & Devi (UNC)

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
```



A[p..r]

5

Partition

A[p..q – 1]

A[q+1..r]

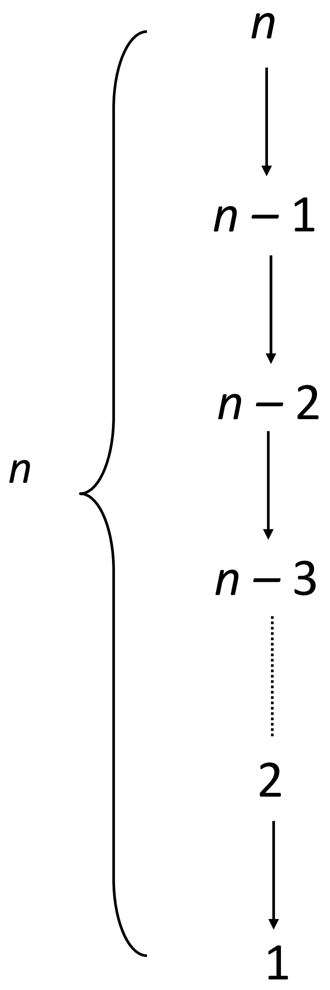
5

≤ 5

≥ 5

Worst-case Partition Analysis

Recursion tree for worst-case partition



n

$n-1$

$n-2$

$n-3$

2

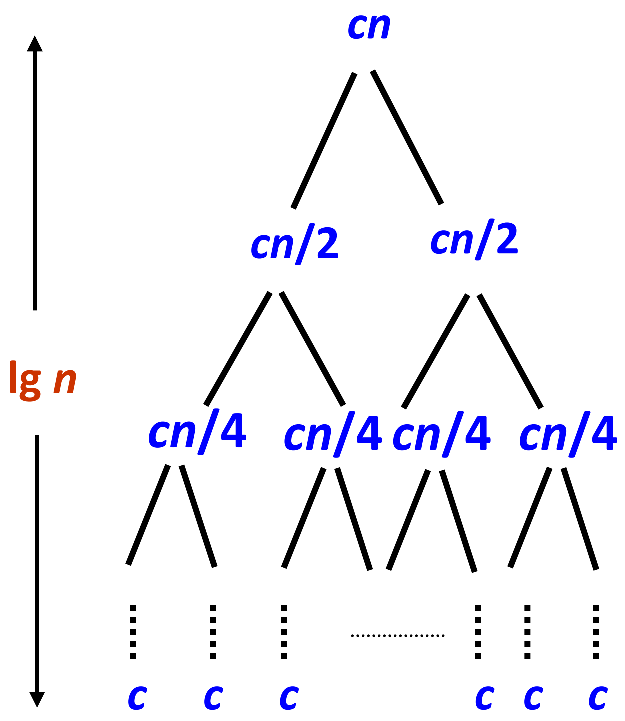
1

n

Split off a single element at each level:

$$
\begin{aligned}
T(n) &= T(n-1) + T(0) + \text{PartitionTime}(n) \\
&= T(n-1) + \Theta(n) \\
&= \textstyle\sum_{k=1 \text{ to } n} \Theta(k) \\
&= \Theta(\textstyle\sum_{k=1 \text{ to } n} k) \\
&= \Theta(n^2)
\end{aligned}
$$

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.
  - **Note:** Insertion Sort is very fast on almost-sorted lists.

Average case analysis

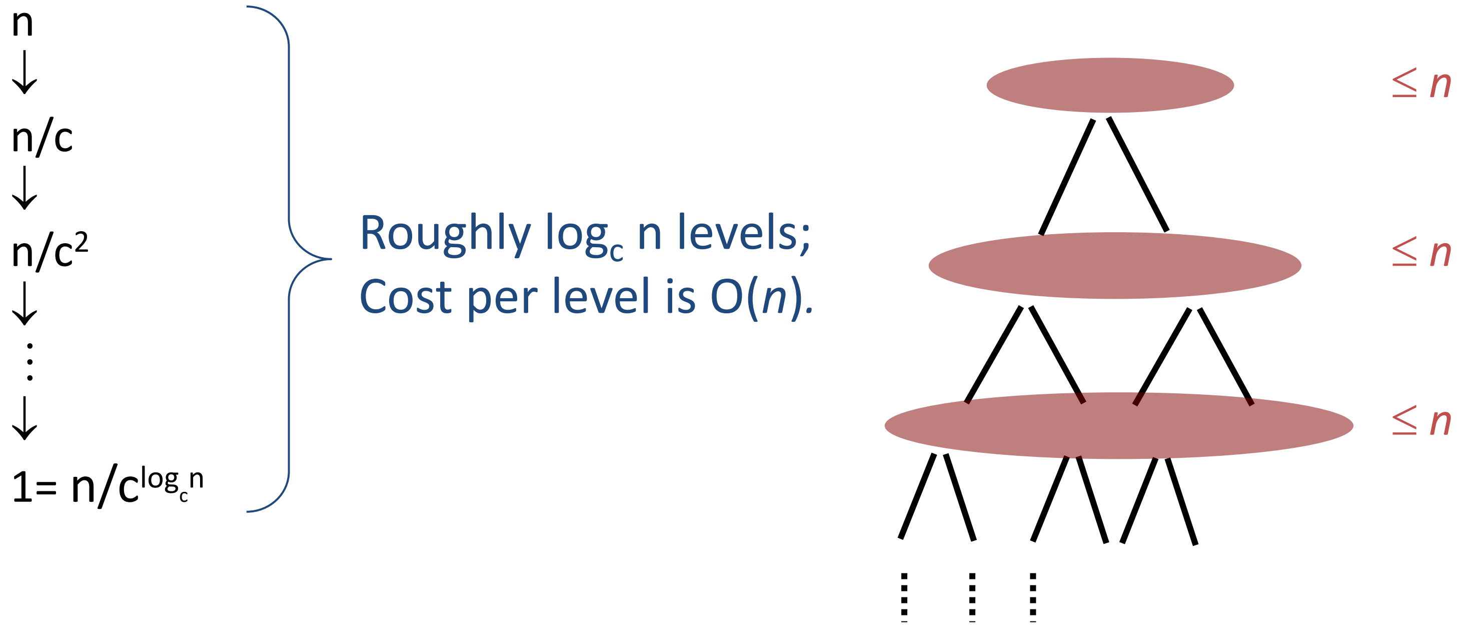
Unbalanced Partition Analysis

What happens if we get poorly-balanced partitions,

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

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

↓

n/c

↓

n/c^2

↓

⋮

↓

$1 = n/c^{\log_c n}$

Roughly $\log_c$ n levels;

Cost per level is $O(n)$.

$\le n$

$\le n$

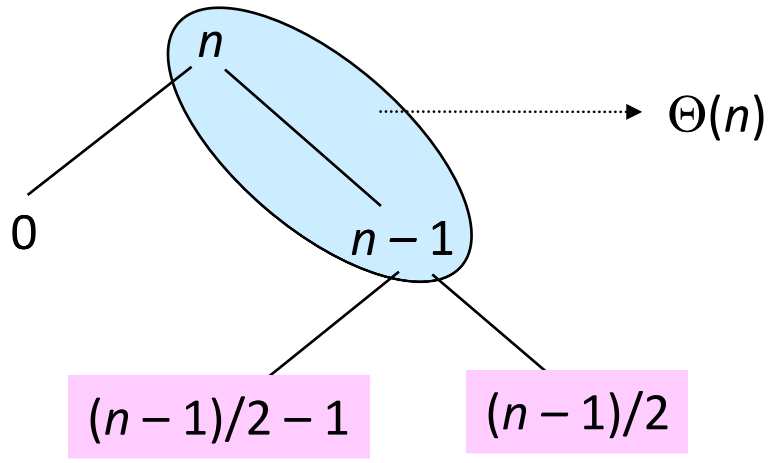
$\le n$

(**Remember:** 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

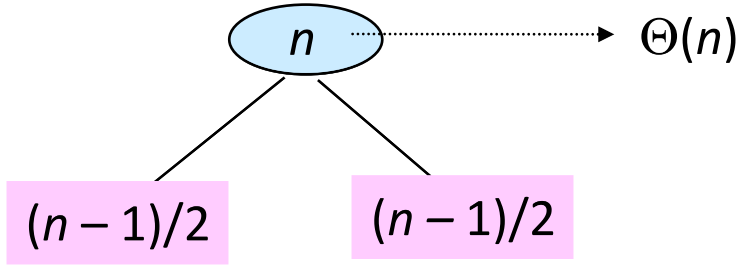


Bad split followed by a good split:

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, 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' ll 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];
    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
```

Technical intermission

Expectation & Indicators

Expectation

- Average or mean
- The expected value of a discrete random variable X is $E[X] = \sum_x x \Pr\{X=x\}$
- Linearity of Expectation
  - $E[X+Y] = E[X]+E[Y]$, for all X, Y
  - $E[aX+Y] = a\,E[X] + E[Y]$, for constant a and all X, Y
- For mutually independent random variables $X_1, \ldots, X_n$
  - $E[X_1 X_2 \ldots X_n] = E[X_1] \cdot E[X_2] \cdot \ldots \cdot E[X_n]$

Expectation – Example

- Let X be the RV denoting the value obtained when a fair die is thrown. What will be the mean of X, when the die is thrown n times.
  - Let $X_1, X_2, \ldots, X_n$ denote the values obtained during the n throws.
  - The mean of the values is $(X_1+X_2+\ldots+X_n)/n$.
  - Since the probability of getting values 1 thru 6 is (1/6), on an average we can expect each of the 6 values to show up $(1/6)n$ times.
  - So, the numerator in the expression for mean can be written as $(1/6)n\cdot1+(1/6)n\cdot2+\ldots+(1/6)n\cdot6$
  - The mean, hence, reduces to $(1/6)\cdot1+(1/6)\cdot2+\ldots(1/6)\cdot6$, which is what we get if we apply the definition of expectation.

Indicator Random Variables

- A simple yet powerful technique for computing the expected value of a random variable.
- Convenient method for converting between probabilities and expectations.
- Helpful in situations in which there may be dependence.
- Takes only 2 values, 1 and 0.
- Indicator Random Variable for an event A of a sample space is defined as:

$$I\{A\} = \begin{cases} 1 & \text{if } A \text{ occurs,} \\ 0 & \text{if } A \text{ does not occur.} \end{cases}$$

Indicator Random Variable

Lemma 5.1

Given a sample space S and an event A in the sample space S, let $X_A = \text{I}\{A\}$. Then $\text{E}[X_A] = \Pr\{\text{A}\}$.

Proof:

Let $\bar{A} = S - A$ (Complement of A)

Then,

$$\begin{aligned} \text{E}[X_A] &= \text{E}[\text{I}\{A\}] \\ &= 1 \cdot \Pr\{A\} + 0 \cdot \Pr\{\bar{A}\} \\ &= \Pr\{A\} \end{aligned}$$

Indicator RV – Example

Problem: Determine the expected number of heads in n coin flips.

Method 1: Without indicator random variables.

Let X be the random variable for the number of heads in n flips.

Then, $E[X] = \sum_{k=0..n} k \cdot \Pr\{X=k\}$

We can solve this with a lot of math.

Indicator RV – Example

- **Method 2 :** Use Indicator Random Variables
- Define n indicator random variables, X_i, $1 \leq i \leq n$.
- Let X_i be the indicator random variable for the event that the i^{th} flip results in a Head.
- X_i = I{the i^{th} flip results in H}
- Then $X = X_1 + X_2 + \ldots + X_n = \sum_{i=1..n} X_i$.
- By Lemma 5.1, $E[X_i] = \Pr\{H\} = ½$, $1 \leq i \leq n$.
- Expected number of heads is $E[X] = E[\sum_{i=1..n} X_i]$.
- By linearity of expectation, $E[\sum_{i=1..n} X_i] = \sum_{i=1..n} E[X_i]$.
- $E[X] = \sum_{i=1..n} E[X_i] = \sum_{i=1..n} ½ = n/2$.

Back to business

Average case analysis

Average Case Analysis of **Randomized Quicksort**

Let RV X = number of comparisons over all calls to Partition.

Q: Why is it a good measure?

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)

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.

Exercise: Prove this.

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

Analysis (Continued)

Therefore,

$$
\begin{aligned}
E[X] &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \\
&= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} \\
&< \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k} \\
&= \sum_{i=1}^{n-1} O(\lg n) \\
&= \boxed{O(n \lg n).}
\end{aligned}
$$

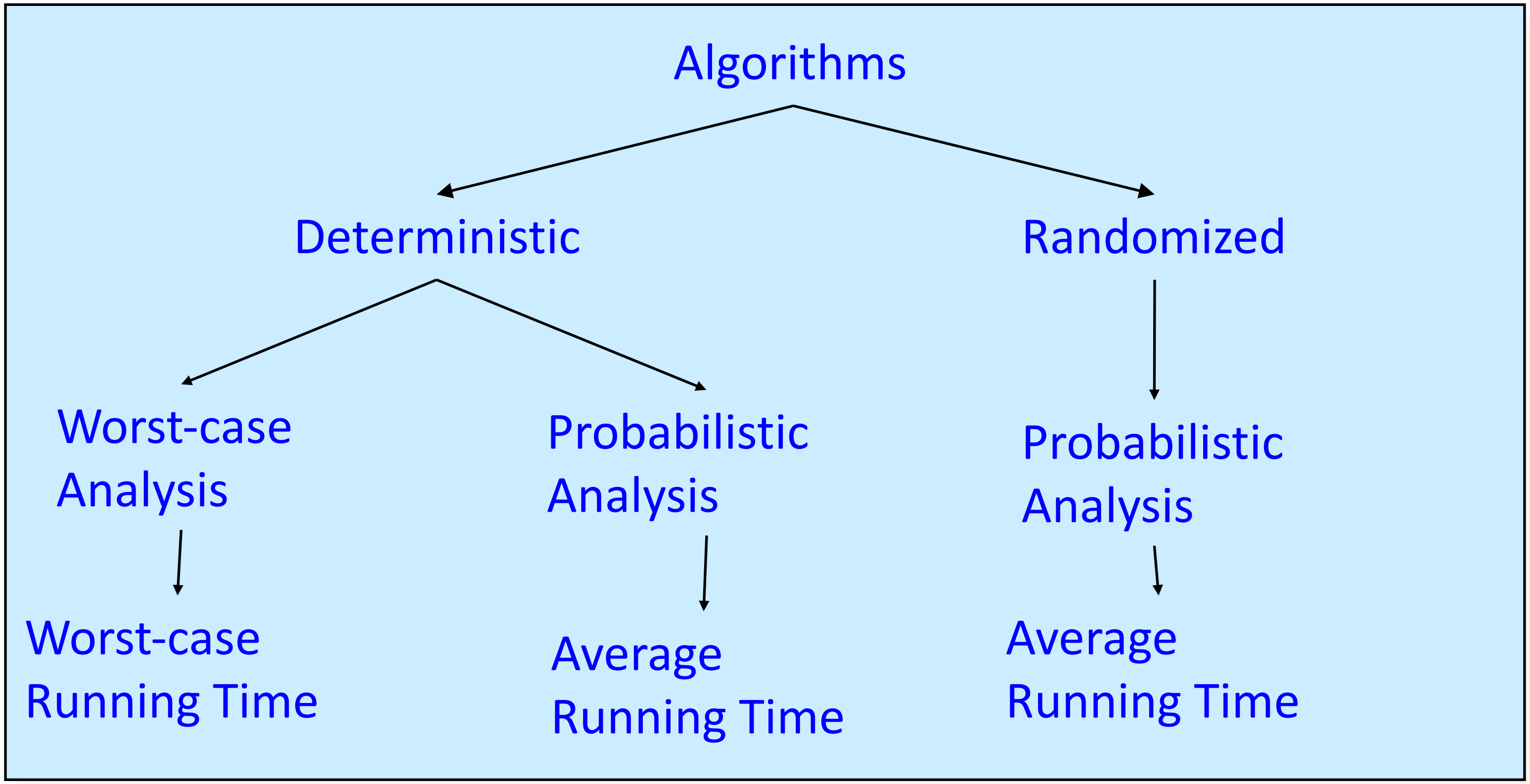
Substitute $k = j - i$.

$$\sum_{k=1}^{n} \frac{1}{k} = H_n \quad (n^{\text{th}} \text{ Harmonic number})$$

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

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.


Algorithms
Deterministic
Randomized
Worst-case Analysis
Probabilistic Analysis
Probabilistic Analysis
Worst-case Running Time
Average Running Time
Average Running Time