Analysis of Algorithms, I
CSOR W4231

Eleni Drinea
*Computer Science Department*

Columbia University

Quicksort, randomized quicksort, occupancy problems
Outline

1. Quicksort
2. Randomized Quicksort
3. Random variables and linearity of expectation
4. Analysis of randomized Quicksort
5. Occupancy problems
1. Quicksort

2. Randomized Quicksort

3. Random variables and linearity of expectation

4. Analysis of randomized Quicksort

5. Occupancy problems
Quicksort facts

- Quicksort is a **divide and conquer** algorithm
- It is the standard algorithm used for sorting
- It is an **in-place** algorithm
- Its worst-case running time is $\Theta(n^2)$ but its average-case running time is $\Theta(n \log n)$
- We will use it to introduce **randomized** algorithms
QuickSort: main idea

- Pick an input item, call it $pivot$, and place it in its final location in the sorted array by re-organizing the array so that:
  - all items $\leq pivot$ are placed before $pivot$
  - all items $> pivot$ are placed after $pivot$

- Recursively sort the subarray to the left of $pivot$.
- Recursively sort the subarray to the right of $pivot$. 
Quicksort pseudocode

Quicksort\((A, left, right)\)

\[
\text{if } |A| = 0 \text{ then return } \quad //A \text{ is empty}
\]
end if

\[
split = \text{Partition}(A, left, right)
\]

Quicksort\((A, left, split - 1)\)

Quicksort\((A, split + 1, right)\)

Initial call: \text{Quicksort}(A, 1, n)
Subroutine $\text{Partition}(A, left, right)$

**Notation:** $A[i, j]$ denotes the portion of $A$ starting at position $i$ and ending at position $j$.

$\text{Partition}(A, left, right)$

1. picks a pivot item
2. re-organizes $A[left, right]$ so that
   - all items before pivot are $\leq$ pivot
   - all items after pivot are $> pivot$
3. returns split, the index of pivot in the re-organized array

After Partition, $A[left, right]$ looks as follows:
1. Pick a *pivot* item: for simplicity, always pick the **last item** of the array as *pivot*, i.e., $pivot = A[right]$.
   - Thus $A[right]$ will be placed in its final location in the sorted output when *Partition* returns; it **will never be used** (or moved) again until the algorithm terminates.

2. Re-organize the input array $A$ in place. *How?*

*(What if we didn’t care to implement *Partition* in place?)*
Implementing *Partition* in place

*Partition* examines the items in $A[left, right]$ one by one and maintains three regions in $A$. Specifically, after examining the $j$-th item for $j \in [left, right - 1]$, the regions are:

1. **Left region**: starts at $left$ and ends at $split$; $A[left, split]$ contains all items $\leq pivot$ examined so far.
2. **Middle region**: starts at $split + 1$ and ends at $j$; $A[split + 1, j]$ contains all items $> pivot$ examined so far.
3. **Right region**: starts at $j + 1$ and ends at $right - 1$; $A[j + 1, right - 1]$ contains all unexamined items.

![Diagram of partition algorithm](image)
At the **beginning** of iteration $j$, $A[j]$ is compared with $pivot$.

If $A[j] \leq pivot$:

1. swap $A[j]$ with $A[split + 1]$, the first element of the **middle region** (items $> pivot$): since $A[split + 1] > pivot$, it is “safe” to move it to the end of the middle region.

2. increment $split$ to include $A[j]$ in the **left region** (items $> pivot$).
Iteration $j$: when $A[j] \leq pivot$

Beginning of iteration $j$ (assume $A[j] \leq pivot$)

End of iteration $j$: $A[j]$ got swapped with $A[\text{split+1}]$, split got updated to split+1

Beginning of iteration $j$ (assume $A[j] \leq pivot$)

End of iteration $j$: $A[j]$ got swapped with $A[\text{split+1}]$, split got updated to split+1
Example: $A = \{1, 3, 7, 2, 6, 4, 5\}$, Partition($A, 1, 7$)
Pseudocode for Partition

**Partition**\((A, left, right)\)

- \(pivot = A[right]\)
- \(split = left - 1\)

**for** \(j = left\) to \(right - 1\) **do**
  - **if** \(A[j] \leq pivot\) **then**
    - swap\((A[j], A[split + 1])\)
    - \(split = split + 1\)
  - **end if**

**end for**

swap\((pivot, A[split + 1])\) //place pivot after \(A[split]\) (why?)

return \(split + 1\) //the final position of pivot
Analysis of Partition: correctness

**Notation:** $A[i, j]$ denotes the portion of $A$ that starts at position $i$ and ends at position $j$.

**Claim 1.**

For $\text{left} \leq j \leq \text{right} - 1$, at the end of loop $j$,

1. all items in $A[\text{left}, \text{split}]$ are $\leq \text{pivot}$; and
2. all items in $A[\text{split} + 1, j]$ are $> \text{pivot}$

**Remark:** If the claim is true, correctness of Partition follows (why?).
Proof of Claim 1

By induction on $j$.

1. **Base case:** For $j = left$ (that is, during the first execution of the for loop), there are two possibilities:
   - if $A[left] \leq pivot$, then $A[left]$ is swapped with itself and $split$ is incremented to equal $left$;
   - otherwise, nothing happens.

   In both cases, the claim holds for $j = left$.

2. **Hypothesis:** Assume that the claim is true for some $left \leq j < right - 1$.
   - That is, at the end of loop $j$, all items in $A[left, split]$ are $\leq pivot$ and all items in $A[split + 1, j]$ are $> pivot$. 
3. **Step:** We will show the claim for $j + 1$. That is, we will show that after loop $j + 1$, all items in $A[left, split]$ are $\leq pivot$ and all items in $A[split + 1, j + 1]$ are $> pivot$.

- At the beginning of loop $j + 1$, by the hypothesis, items in $A[left, split]$ are $\leq pivot$ and items in $A[split + 1, j]$ are $> pivot$.

- Inside loop $j + 1$, there are two possibilities:

This completes the proof of the inductive step.
Analysis of *Partition*: running time and space

- **Running time**: on input size $n$, *Partition* goes through each of the $n - 1$ leftmost elements once and performs constant amount of work per element.
  
  ⇒ *Partition* requires $\Theta(n)$ time.

- **Space**: in-place algorithm
Analysis of Quicksort: correctness

- Quicksort is a recursive algorithm; we will prove correctness by induction on the input size $n$.
- We will use strong induction: the induction step at $n$ requires that the inductive hypothesis holds at all steps $1, 2, \ldots, n - 1$ and not just at step $n - 1$, as with simple induction.
- Strong induction is most useful when several instances of the hypothesis are required to show the inductive step.
Analysis of Quicksort: correctness

- **Base case:** for $n = 0$, Quicksort sorts correctly.

- **Hypothesis:** for all $0 \leq m < n$, Quicksort correctly sorts on input size $m$.

- **Step:** show that Quicksort correctly sorts on input size $n$.
  
  - **Partition**$(A, 1, n)$ re-organizes $A$ so that all items
    - in $A[1, \ldots, \text{split} - 1]$ are $\leq A[\text{split}]$;
    - in $A[\text{split} + 1, \ldots, n]$ are $> A[\text{split}]$.

  - Next, Quicksort$(A, 1, \text{split} - 1)$, Quicksort$(A, \text{split} + 1, n)$ will correctly sort their inputs (by the hypothesis). Hence

  $A[1] \leq \ldots \leq A[\text{split} - 1]$ and $A[\text{split} + 1] \leq \ldots \leq A[n]$.

At this point, Quicksort terminates and $A$ is sorted.
Analysis of Quicksort: space and running time

- **Space**: in-place algorithm

- **Running time** $T(n)$: depends on the arrangement of the input elements
  - the sizes of the inputs to the two recursive calls—hence the form of the recurrence—depend on how pivot compares to the rest of the input items
Suppose that in every call to **Partition** the pivot item is the **median** of the input.

Then every **Partition** splits its input into two lists of almost equal sizes, thus

\[ T(n) = 2T(n/2) + \Theta(n) = O(n \log n). \]

This is a “balanced” partitioning.

- **Example of best case:** \( A = [1 \ 3 \ 2 \ 5 \ 7 \ 6 \ 4] \)

**Remark 1.**

*You can show that \( T(n) = O(n \log n) \) for any splitting where the two subarrays have sizes \( \alpha n, \ (1 - \alpha)n \) respectively, for constant \( 0 < \alpha < 1 \).*
Running time of Quicksort: Worst Case

- Upper bound for worst-case running time: \( T(n) = O(n^2) \)
  - at most \( n \) calls to Partition (one for each item as pivot)
  - Partition requires \( O(n) \) time

- This worst-case upper bound is tight:
  - If every time Partition is called pivot is greater (or smaller) than every other item, then its input is split into two lists, one of which has size 0.
    - This partitioning is very “unbalanced”: let \( c, d > 0 \) be constants, where \( T(0) = d \); then
      \[
      T(n) = T(n - 1) + T(0) + cn = \Theta(n^2).
      \]

△ A worst-case input is the sorted input!
Running time: average case analysis

**Average case:** what is an “average” input to sorting?

- Depends on the application.
- Intuition why average-case analysis for uniformly distributed inputs to **Quicksort** is $O(n \log n)$ appears in your textbook.
- We will use **randomness** within the algorithm to provide **Quicksort** with a uniform at random input.
1. Quicksort

2. Randomized Quicksort

3. Random variables and linearity of expectation

4. Analysis of randomized Quicksort

5. Occupancy problems
1. **Deterministic** algorithm, randomness over the inputs

- On the same input, the algorithm always produces the same output using the same time.
  - So far, we have only encountered such algorithms.
- The input is randomly generated according to some underlying distribution.
- **Average case analysis**: analysis of the running time of the algorithm on an average input.
2. **Randomized** algorithm, worst-case (deterministic) input

- On the same input, the algorithm produces the same output but different executions may require different running times.
  - The latter depend on the random choices of the algorithm (e.g., coin flips, random numbers).
  - Random samples are assumed **independent** of each other.

- **Worst-case input**

- **Expected running time analysis**: analysis of the running time of the randomized algorithm on a worst-case input.
1. Deterministic algorithms are a special case of randomized algorithms.

2. Even when equally efficient deterministic algorithms exist, randomized algorithms may be simpler, require less memory of the past or be useful for symmetry-breaking.
Randomized Quicksort

Can we use randomization so that Quicksort works with an “average” input even when it receives a worst-case input?

1. Explicitly permute the input.


Idea 1 (intuition behind random sampling).

No matter how the input is organized, we won’t often pick the largest or smallest item as pivot (unless we are really, really unlucky). Thus most often the partitioning will be “balanced”.
Randomized-Quicksort\((A, left, right)\)

\[
\begin{align*}
\text{if } |A| == 0 & \text{ then return } // A \text{ is empty} \\
\text{end if} \\
\text{split} = \text{Randomized-Partition}\((A, left, right)\) \\
\text{Randomized-Quicksort}\((A, left, \text{split} - 1)\) \\
\text{Randomized-Quicksort}\((A, \text{split} + 1, right)\)
\end{align*}
\]

Randomized-Partition\((A, left, right)\)

\[
\begin{align*}
b = \text{random}\((left, right)\) \\
\text{swap}(A[b], A[right]) \\
\text{return Partition}\((A, left, right)\)
\end{align*}
\]

Subroutine \text{random}\((i, j)\) returns a random number between \(i\) and \(j\) inclusive.
1. Quicksort

2. Randomized Quicksort

3. Random variables and linearity of expectation

4. Analysis of randomized Quicksort

5. Occupancy problems
To analyze the expected running time of a randomized algorithm we keep track of certain parameters and their expected size over the random choices of the algorithm.

To this end, we use random variables.

A discrete random variable $X$ takes on a finite number of values, each with some probability. We’re interested in its expectation

$$E[X] = \sum_j j \cdot \Pr[X = j].$$
Example 1: Bernoulli trial

**Experiment 1**: flip a biased coin which comes up

- *heads* with probability $p$
- *tails* with probability $1 - p$

**Question**: what is the expected number of *heads*?
Experiment 1: flip a biased coin which comes up
  - heads with probability $p$
  - tails with probability $1 - p$

Question: what is the expected number of heads?

Let $X$ be a random variable such that

$$X = \begin{cases} 
1 & \text{, if coin flip comes heads} \\
0 & \text{, if coin flip comes tails}
\end{cases}$$
**Example 1: Bernoulli trial**

**Experiment 1:** flip a biased coin which comes up

- *heads* with probability $p$
- *tails* with probability $1 - p$

**Question:** what is the expected number of *heads*?

Let $X$ be a random variable such that

\[
X = \begin{cases} 
1 & \text{, if coin flip comes *heads*} \\
0 & \text{, if coin flip comes *tails*} 
\end{cases}
\]

Then

\[
\Pr[X = 1] = p \\
\Pr[X = 0] = 1 - p \\
E[X] = 1 \cdot \Pr[X = 1] + 0 \cdot \Pr[X = 0] = p
\]
Indicator random variables

- **Indicator random variable:** a discrete random variable that only takes on values 0 and 1.

- Indicator random variables are used to denote occurrence (or not) of an event.

  Example: in the biased coin flip example, $X$ is an indicator random variable that denotes the occurrence of *heads*.

**Fact 1.**

*If $X$ is an indicator random variable, then $E[X] = \Pr[X = 1]$.***
Experiment 2: flip the biased coin $n$ times

Question: what is the expected number of heads?
**Example 2: Bernoulli trials**

**Experiment 2:** flip the biased coin $n$ times

**Question:** what is the expected number of *heads*?

**Answer 1:** Let $X$ be the random variable counting the number of times *heads* appears.

$$E[X] = \sum_{j=0}^{n} j \cdot \Pr[X = j].$$

$\Pr[X = j]$?
Example 2: Bernoulli trials

Experiment 2: flip the biased coin $n$ times

Question: what is the expected number of heads?

Answer 1: Let $X$ be the random variable counting the number of times heads appears.

$$E[X] = \sum_{j=0}^{n} j \cdot \Pr[X = j].$$

$\Pr[X = j]$?

$X$ follows the binomial distribution $B(n, p)$, thus

$$\Pr[X = j] = \binom{n}{j} p^j (1 - p)^{n-j}.$$
Example 2: Bernoulli trials

A different way to think about $X$:

**Answer 2:** for $1 \leq i \leq n$, let $X_i$ be an indicator random variable such that

$$X_i = \begin{cases} 1 , & \text{if } i\text{-th coin flip comes } \text{heads} \\ 0 , & \text{if } i\text{-th coin flip comes } \text{tails} \end{cases}$$
Example 2: Bernoulli trials

A different way to think about $X$:

**Answer 2:** for $1 \leq i \leq n$, let $X_i$ be an indicator random variable such that

$$X_i = \begin{cases} 1, & \text{if } i\text{-th coin flip comes heads} \\ 0, & \text{if } i\text{-th coin flip comes tails} \end{cases}$$

Define the random variable

$$X = \sum_{i=1}^{n} X_i$$

We want $E[X]$. By Fact 1, $E[X_i] = p$, for all $i$. 
Linearity of expectation

\[ X = \sum_{i=1}^{n} X_i, \quad E[X_i] = p, \quad E[X] = ? \]
Linearity of expectation

\[ X = \sum_{i=1}^{n} X_i, \quad E[X_i] = p, \quad E[X] = ? \]

**Remark 1:** \( X \) is a complicated random variable defined as the sum of simpler random variables whose expectation is known.
Linearity of expectation

\[ X = \sum_{i=1}^{n} X_i, \quad E[X_i] = p, \quad E[X] =? \]

**Remark 1:** $X$ is a complicated random variable defined as the sum of simpler random variables whose expectation is known.

**Proposition 1 (Linearity of expectation).**

Let $X_1, \ldots, X_k$ be arbitrary random variables. Then

\[ E[X_1 + X_2 + \ldots + X_k] = E[X_1] + E[X_2] + \ldots + E[X_k] \]
Linearity of expectation

\[ X = \sum_{i=1}^{n} X_i, \quad E[X_i] = p, \quad E[X] = ? \]

**Remark 1:** \( X \) is a complicated random variable defined as the sum of simpler random variables whose expectation is known.

**Proposition 1 (Linearity of expectation).**

Let \( X_1, \ldots, X_k \) be arbitrary random variables. Then

\[ E[X_1 + X_2 + \ldots + X_k] = E[X_1] + E[X_2] + \ldots + E[X_k] \]

**Remark 2:** We made no assumptions on the random variables. For example, they do not need be independent.
Answer 2: for $1 \leq i \leq n$, let $X_i$ be an indicator random variable such that

$$X_i = \begin{cases} 1, & \text{if } i\text{-th coin flip comes } heads \\ 0, & \text{if } i\text{-th coin flip comes } tails \end{cases}$$

Define the random variable

$$X = \sum_{i=1}^{n} X_i$$

By Fact 1, $E[X_i] = p$, for all $i$. By linearity of expectation,

$$E[X] = E\left[ \sum_{i=1}^{n} X_i \right] = \sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} p = np.$$
Today

1. Quicksort
2. Randomized Quicksort
3. Random variables and linearity of expectation
4. Analysis of randomized Quicksort
5. Occupancy problems
Let $T(n)$ be the expected running time of Randomized-Quicksort.

We want to bound $T(n)$.

Randomized-Quicksort differs from Quicksort only in how they select their pivot elements.

We will analyze Randomized-Quicksort based on Quicksort and Partition.
Pseudocode for Partition

\[
\text{Partition}(A, \text{left}, \text{right}) \\
pivot = A[\text{right}] \quad \text{line 1} \\
\text{split} = \text{left} - 1 \quad \text{line 2} \\
\text{for } j = \text{left} \text{ to } \text{right} - 1 \text{ do} \quad \text{line 3} \\
\quad \text{if } A[j] \leq \text{pivot} \text{ then} \quad \text{line 4} \\
\quad \quad \text{swap}(A[j], A[\text{split} + 1]) \quad \text{line 5} \\
\quad \quad \text{split} = \text{split} + 1 \quad \text{line 6} \\
\quad \text{end if} \quad \text{line 7} \\
\text{end for} \quad \text{line 8} \\
\text{swap}(\text{pivot}, A[\text{split} + 1]) \quad \text{line 9} \\
\text{return } \text{split} + 1 \quad \text{line 10}
\]
Few observations

1. *How many times is Partition called?*
Few observations

1. *How many times is* Partition *called?*
   At most $n$.

2. Further, each Partition call spends some work
   1. *outside* the for loop

   2. *inside* the for loop
Few observations

1. *How many times is Partition called?*
   At most $n$.

2. Further, each Partition call spends some work
   1. outside the for loop
      - every Partition spends constant work outside the for loop
      - at most $n$ calls to Partition
      $\Rightarrow$ total work outside the for loop in all calls to Partition is $O(n)$
   2. inside the for loop
**Few observations**

1. *How many times is Partition called?*
   At most $n$.

2. Further, each *Partition* call spends some work
   1. **outside** the for loop
      - every *Partition* spends **constant** work outside the for loop
      - at most $n$ calls to *Partition*
      ⇒ total work **outside** the for loop in all calls to *Partition* is $O(n)$
   2. **inside** the for loop
      - let $X$ be the total number of comparisons performed at line 4 in all calls to *Partition*
      - each comparison may require some further **constant** work (lines 5 and 6)
      ⇒ total work **inside** the for loop in all calls to *Partition* is $O(X)$
Towards a bound for \( T(n) \)

\( X = \) the total number of comparisons in all \texttt{Partition} calls.

The running time of \texttt{Randomized-Quicksort} is

\[ O(n + X). \]

Since \( X \) is a random variable, we need \( E[X] \) to bound \( T(n) \).
Towards a bound for $T(n)$

$X = \text{the total number of comparisons in all } \text{Partition calls.}$

The running time of $\text{Randomized-Quicksort}$ is

$$O(n + X).$$

Since $X$ is a random variable, we need $E[X]$ to bound $T(n)$.

**Fact 2.**

*Fix any two input items. During the execution of the algorithm, they may be compared at most once.*
Towards a bound for $T(n)$

$X = \text{the total number of comparisons in all } \text{Partition calls.}$

The running time of \text{Randomized-Quicksort} is

$$O(n + X).$$

Since $X$ is a random variable, we need $E[X]$ to bound $T(n)$.

\textbf{Fact 2.}

\textit{Fix any two input items. During the execution of the algorithm, they may be compared at most once.}

\textbf{Proof.}

Comparisons are only performed with the \textit{pivot} of each \text{Partition} call. After \text{Partition} returns, \textit{pivot} is in its final location in the output and will not be part of the input to any future recursive call.
There are $n$ numbers in the input, hence $\binom{n}{2} = \frac{n(n-1)}{2}$ distinct (unordered) pairs of input numbers.

From Fact 1, the algorithm will perform at most $\binom{n}{2}$ comparisons.

What is the expected number of comparisons?
Simplifying the analysis

- There are $n$ numbers in the input, hence $\binom{n}{2} = \frac{n(n-1)}{2}$ distinct (unordered) pairs of input numbers.
- From Fact 1, the algorithm will perform at most $\binom{n}{2}$ comparisons.
- What is the expected number of comparisons?

To simplify the analysis

- relabel the input as $z_1, z_2, \ldots, z_n$, where $z_i$ is the $i$-th smallest number.
- assume that all input numbers are distinct; thus $z_i < z_j$, for $i < j$. 

Let $X_{ij}$ be an indicator random variable such that

$$X_{ij} = \begin{cases} 
1, & \text{if } z_i \text{ and } z_j \text{ are ever compared} \\
0, & \text{otherwise}
\end{cases}$$
Let $X_{ij}$ be an indicator random variable such that

$$X_{ij} = \begin{cases} 
1, & \text{if } z_i \text{ and } z_j \text{ are ever compared} \\
0, & \text{otherwise}
\end{cases}$$

The total number of comparisons is given by

$$X = \sum_{1 \leq i < j \leq n} X_{ij}.$$
Let $X_{ij}$ be an indicator random variable such that

$$X_{ij} = \begin{cases} 1, & \text{if } z_i \text{ and } z_j \text{ are ever compared} \\ 0, & \text{otherwise} \end{cases}$$

The total number of comparisons is given by $X = \sum_{1 \leq i < j \leq n} X_{ij}$. 

$E[X] =$?
Writing $X$ as the sum of indicator random variables

Let $X_{ij}$ be an indicator random variable such that

$$X_{ij} = \begin{cases} 
1, & \text{if } z_i \text{ and } z_j \text{ are ever compared} \\
0, & \text{otherwise}
\end{cases}$$

The total number of comparisons is given by

$$X = \sum_{1 \leq i < j \leq n} X_{ij}.$$ 

By linearity of expectation

$$E[X] = E \left[ \sum_{1 \leq i < j \leq n} X_{ij} \right] = \sum_{1 \leq i < j \leq n} E[X_{ij}] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr[X_{ij} = 1]$$
Writing $X$ as the sum of indicator random variables

Let $X_{ij}$ be an indicator random variable such that

$$X_{ij} = \begin{cases} 
1, & \text{if } z_i \text{ and } z_j \text{ are ever compared} \\
0, & \text{otherwise}
\end{cases}$$

The total number of comparisons is given by $X = \sum_{1 \leq i < j \leq n} X_{ij}$.

By linearity of expectation

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

**Goal:** compute $\Pr[X_{ij} = 1]$, that is, the probability that two fixed items $z_i$ and $z_j$ are ever compared.
Fix two items $z_i$ and $z_j$. When are they compared?

**Notation:** let $Z_{ij} = \{z_i, z_{i+1}, \ldots, z_j\}$

Consider the initial call `Partition(A, 1, n)`. Assume it picks $z_k$ **outside** $Z_{ij}$ as pivot (see figure below).

1. $z_i$ and $z_j$ are **not** compared in this call *(why?)*.
2. All items in $Z_{ij}$ will be greater (or smaller) than $z_k$, so they will **all be input to the same subproblem** after `Partition(A, 1, n)` returns.
In the first Partition with \( \text{pivot} \in \mathcal{Z}_{ij} = \{z_i, \ldots, z_j\} \)

The first Partition call that picks its pivot from \( \mathcal{Z}_{ij} \) determines if \( z_i, z_j \) are ever compared. Three possibilities:

1. \( \text{pivot} = z_i \)

2. \( \text{pivot} = z_j \)

3. \( \text{pivot} = z_\ell, \) for some \( i < \ell < j \)
In the first **Partition** with \( pivot \in Z_{ij} = \{z_i, \ldots, z_j\} \)

The first **Partition** call that picks its \( pivot \) from \( Z_{ij} \) determines if \( z_i, z_j \) are ever compared. Three possibilities:

1. \( pivot = z_i \)

   \( z_i \) is compared with every element in \( Z_{ij} - \{z_i\} \), thus with \( z_j \) too. \( z_i \) is placed in its final location in the output and will not appear in any future calls to **Partition**.

2. \( pivot = z_j \)

   \( z_j \) is compared with every element in \( Z_{ij} - \{z_j\} \), thus with \( z_i \) too. \( z_j \) is placed in its final location in the output and will not appear in any future recursive calls.

3. \( pivot = z_\ell \), for some \( i < \ell < j \)

   \( z_i \) and \( z_j \) are never compared (why?)
So $z_i$ and $z_j$ are compared when . . .

. . . either of them is chosen as *pivot* in that *first* Partition call that chooses its *pivot* element from $Z_{ij}$.

Now we can compute $\Pr[X_{ij} = 1]$:

$$
\Pr[X_{ij} = 1] = \Pr[z_i \text{ is chosen as } \textit{pivot} \text{ by the first } \text{Partition}
\text{ that picks its } \textit{pivot} \text{ from } Z_{ij}, \text{ or}
\text{ } z_j \text{ is chosen as } \textit{pivot} \text{ by the first } \text{Partition}
\text{ that picks its } \textit{pivot} \text{ from } Z_{ij}] \quad (1)
$$
Suppose we are given a set of events $\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n$, and we are interested in the probability that any of them happens.

**Union bound:** Given events $\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n$, we have

$$\Pr \left[ \bigcup_{i=1}^{n} \varepsilon_i \right] \leq \sum_{i=1}^{n} \Pr[\varepsilon_i].$$

**Union bound for mutually exclusive events:** Suppose that $\varepsilon_i \cap \varepsilon_j = \emptyset$ for each pair of events. Then

$$\Pr \left[ \bigcup_{i=1}^{n} \varepsilon_i \right] = \sum_{i=1}^{n} \Pr[\varepsilon_i].$$
Computing the probability that $z_i$ and $z_j$ are compared

Since the two events in equation (1) are mutually exclusive, we obtain

$$\Pr[X_{ij} = 1] = \Pr[z_i \text{ is chosen as } pivot \text{ by the first } \text{Partition} \text{ call that picks its } pivot \text{ from } Z_{ij}]$$

$$+ \Pr[z_j \text{ is chosen as } pivot \text{ by the first } \text{Partition} \text{ call that picks its } pivot \text{ from } Z_{ij}]$$

$$= \frac{1}{j-i+1} + \frac{1}{j-i+1} = \frac{2}{j-i+1},$$

(2)

since the set $Z_{ij}$ contains $j - i + 1$ elements.
From $\Pr[X_{ij} = 1]$ to $E[X]$

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

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

(3)

Note that $\sum_{\ell=1}^{k} \frac{1}{\ell} = H_k$ is the $k$-th harmonic number, such that

$$\ln k \leq H_k \leq \ln k + 1$$

(4)

Hence $\sum_{\ell=2}^{n-i+1} \frac{1}{\ell} \leq \ln (n - i + 1)$. Substituting in (3), we get

$$E[X] \leq 2 \sum_{i=1}^{n-1} \ln (n - i + 1) \leq 2 \sum_{i=1}^{n-1} \ln n = O(n \ln n)$$
Equations (3), (4) also yield a lower bound of $\Omega(n \ln n)$ for $E[X]$ (show this!).

Hence $E[X] = \Theta(n \ln n)$. Then the expected running time of Randomized-Quicksort is

$$T(n) = \Theta(n \ln n)$$
Today

1 Quicksort

2 Randomized Quicksort

3 Random variables and linearity of expectation

4 Analysis of randomized Quicksort

5 Occupancy problems
Balls in bins problems

**Occupancy problems:** find the distribution of balls into bins when \( m \) balls are thrown independently and uniformly at random into \( n \) bins.

- Applications: analysis of randomized algorithms and data structures (e.g., hash table)

**Q1:** How many balls can we throw before it is more likely than not that some bin contains at least two balls?

In symbols: find \( k \) such that

\[
\Pr[\exists \text{ bin with } \geq 2 \text{ balls after } k \text{ balls thrown}] > \frac{1}{2}
\]
Easier to analyze the complement of this event

Easier to think about the probability of the complementary event.

\textit{Q1 (rephrased):} Find $k$ such that

$$\Pr[\text{every bin has } \leq 1 \text{ ball after } k \text{ balls thrown}] \leq 1/2$$
Analysis: one ball at a time

- The 1st ball falls into some bin.
- The 2nd ball falls into a new bin w. prob. $1 - \frac{1}{n}$.
- The 3rd ball falls into a new bin (given that the first two balls fell into different bins) w. prob. $1 - \frac{2}{n}$.
- The $m$-th ball falls into a new bin (given that the first $k - 1$ balls fell into different bins) w. prob. $1 - \frac{k-1}{n}$.

By the chain rule of conditional probability, the probability that the $k$-th ball falls into a new bin is given by

$$\prod_{i=1}^{k-1} \left(1 - \frac{i}{n}\right)$$

(5)
Application: the birthday paradox

Use $1 + x \leq e^x$ for all real $x$ to upper bound (5)

\[
\prod_{i=1}^{k-1} e^{-i/n} = e^{-\sum_{i=1}^{k-1} i/n} = e^{-\frac{k(k-1)}{(2\cdot n)}} \approx e^{-\frac{k^2}{2n}} \tag{6}
\]

Requiring $e^{-\frac{k^2}{2n}} < 1/2$ yields $k > \sqrt{n \cdot 2 \ln 2} = \Omega(\sqrt{n})$.

Application: birthday paradox

Assumption: For $n = 365$, each person has an independent and uniform at random birthday from among the 365 days of the year.

Once 23 people are in a room, it is more likely than not that two of them share a birthday.
More balls-in-bins questions

- **Q2**: What is the expected load of a bin after $m$ balls are thrown?

- **Q3**: What is the expected number of empty bins after $m$ balls are thrown?

- **Q4**: What is the load of the fullest bin with high probability?

- **Q5**: What is the expected number of balls until every bin has at least one ball (Coupon Collector’s Problem)?
Expected load of a bin

Suppose that $m$ balls are thrown independently and uniformly at random into $n$ bins. Fix a bin.

- Let $X_i$ be an indicator r.v. such that $X_i = 1$ if and only if ball $i$ falls in the fixed bin. Then

$$E[X_i] = \Pr[X_i = 1] = \frac{1}{n}.$$  

The total #balls in the bin is given by $X = \sum_{i=1}^{m} X_i$. By linearity of expectation,

$$E[X] = \sum_{i=1}^{m} E[X_i] = m/n.$$  

Since bins are symmetric, the expected load of any bin is $m/n$. 
Expected \# empty bins

Suppose that $m$ balls are thrown independently and uniformly at random into $n$ bins. Fix a bin $j$.

- Let $Y_j$ be an indicator r.v. such that $Y_j = 1$ if and only if bin $j$ is empty.
- $\Pr[\text{ball } i \text{ does not fall in bin } j] = 1 - 1/n$
- $\Pr[\text{for all } i, \text{ ball } i \text{ does not fall in bin } j] = (1 - 1/n)^m$
- Hence $\Pr[Y_j = 1] = (1 - 1/n)^m$.

The number of empty bins is given by the random variable $Y = \sum_{j=1}^n Y_j$. By linearity of expectation

$$E[Y] = \sum_{j=1}^n E[Y_j] = n \left(1 - \frac{1}{n}\right)^m \approx ne^{-m/n}$$
Proposition 2.

When throwing \( n \) balls into \( n \) bins uniformly and independently at random, the maximum load in any bin is \( \Theta(\ln n / \ln \ln n) \) with probability close to 1 as \( n \) grows large.

Two-sentence sketch of the proof.

1. Upper bound the probability that any bin contains more than \( k \) balls by a union bound: 
\[
\sum_{j=1}^{n} \sum_{\ell=k}^{n} \binom{n}{\ell} \left( \frac{1}{n} \right)^{\ell} \left( 1 - \frac{1}{n} \right)^{n-\ell}.
\]

2. Compute the smallest possible \( k^* \) such that the probability above is less than \( 1/n \); the latter becomes negligible as \( n \) grows large.
Expected #balls until no empty bins

Suppose that we throw balls independently and uniformly at random into $n$ bins, one at a time (the first ball falls at time $t = 1$).

- We call a throw a **success** if it lands in an empty bin.
- We call the sequence of balls starting after the $(j-1)$-st success and ending with the $j$-th success, the $j$-th **epoch**.
- To understand the process terminates, we need analyze the duration of each epoch.
- To this end, let $\eta_j$ be the #balls thrown in epoch $j$.

Clearly the first ball is a **success**, hence $\eta_1 = 1$.

Let $\eta_2$ be the #balls thrown in epoch 2.

$$\forall t \in \text{epoch } 2, \Pr[\text{ball } t \text{ in epoch } 2 \text{ is a success}] = \frac{n-1}{n}$$

- Similarly, let $\eta_j$ be the #balls thrown in epoch $j$.

$$\forall t \in \text{epoch } j, \Pr[\text{ball } t \text{ in epoch } j \text{ is a success}] = \frac{n-j+1}{n}$$

At the end of the $n$-th epoch, each of the $n$ bins has at least one ball.
Let $\eta = \sum_{j=1}^{n} \eta_j$. We want

$$E[\eta] = E \left[ \sum_{j=1}^{n} \eta_j \right] = \sum_{j=1}^{n} E[\eta_j]$$

- Each epoch is geometrically distributed with success probability $p_j = \frac{n-j+1}{n}$.
- Recall that the expectation of a geometrically distributed variable with success probability $p$ is given by $1/p$.
- Thus $E[\eta_j] = \frac{1}{p_j} = \frac{n}{n-j+1}$.

Then

$$E[\eta] = \sum_{j=1}^{n} \frac{n}{n-j+1} = n \sum_{j=1}^{n} \frac{1}{j} = n(\ln n + O(1))$$
A sample space $\Omega$ consists of the possible outcomes of an experiment.

Each point $x$ in the sample space has an associated probability mass $p(x) \geq 0$, such that $\sum_{x \in \Omega} p(x) = 1$.

Example experiment: flip a fair coin; $\Omega = \{\text{heads, tails}\}$; $\Pr[\text{heads}] = \Pr[\text{tails}] = 1/2$.

We define an event $\mathcal{E}$ to be any subset of $\Omega$, that is, a collection of points in the sample space.

We define the probability of the event to be the sum of the probability masses of all the points in $\mathcal{E}$. That is,

$$\Pr[\mathcal{E}] = \sum_{x \in \mathcal{E}} p(x)$$