1 Probability Basics

1.1 Random Experiments

A **random experiment** is a process whose outcome cannot be predicted with certainty, even under identical conditions.

Examples:

- Tossing a coin
- Rolling a die
- Choosing a pivot uniformly at random from an array

1.2 Sample Space and Outcomes

The **sample space** Ω is the set of all possible outcomes of a random experiment.

An **outcome** ω is a single element of Ω.

Examples:

- Random index from $\{1, \ldots, n\}$: $\Omega = \{1, 2, \ldots, n\}$
- Two coin tosses: $\Omega = \{HH, HT, TH, TT\}$
- Rolling a die: $\Omega = \{1, 2, 3, 4, 5, 6\}$

1.3 Probability Distribution

A **probability distribution** assigns a probability $P(\omega)$ to each outcome $\omega \in \Omega$ such that:

- $P(\omega) \geq 0$ for all ω
- $\sum_{\omega \in \Omega} P(\omega) = 1$

Examples:

- Fair die: $P(i) = \frac{1}{6}$ for $i = 1, \ldots, 6$
- Two fair coins: $P(HH) = P(HT) = P(TH) = P(TT) = \frac{1}{4}$

1.4 Events

An **event** is a subset $E \subseteq \Omega$.

The event occurs if the outcome $\omega \in E$.

Example:

$$E = \{HH, HT, TH\}$$

represents getting at least one head when tossing two coins.

2 Random Variables

A **random variable** is a function:

$$X : \Omega \to \mathbb{R}$$

that assigns a numerical value to each outcome.

3 Expectation

3.1 Definition

For a discrete random variable X, the expected value is:

$$\mathbb{E}[X] = \sum_x x \cdot \Pr(X = x)$$

3.2 Linearity of Expectation

For any random variables X and Y:

$$\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y]$$

Important: Linearity holds *regardless of independence*. This property is crucial for analyzing randomized algorithms.

4 Examples of Expectation

4.1 Two Coin Tosses

Let X be the number of heads when tossing two coins.

$$\Pr(X = 0) = \frac{1}{4}, \quad \Pr(X = 1) = \frac{1}{2}, \quad \Pr(X = 2) = \frac{1}{4}$$

$$\mathbb{E}[X] = 0 \cdot \frac{1}{4} + 1 \cdot \frac{1}{2} + 2 \cdot \frac{1}{4} = 1$$

4.2 Fair Die

Let X be the outcome of a fair die.

$$\mathbb{E}[X] = \sum_{i=1}^{6} i \cdot \frac{1}{6} = \frac{21}{6} = 3.5$$

4.3 Sum of Two Dice

Let $S = D_1 + D_2$ be the sum of two independent fair dice.

Using linearity:

$$\mathbb{E}[S] = \mathbb{E}[D_1] + \mathbb{E}[D_2] = 3.5 + 3.5 = 7$$

Key Insight: No need to enumerate all 36 outcomes.

5 Geometric Distribution

A **geometric random variable** X with parameter p counts the number of independent trials until the first success, where each trial has success probability p.

$$\Pr(X = k) = (1-p)^{k-1}p \quad \text{for } k = 1, 2, \ldots$$

5.1 Expected Value

$$\mathbb{E}[X] = \frac{1}{p}$$

5.2 Example

Toss a fair coin until the first head appears.

Here, $p = \frac{1}{2}$, so:

$$\mathbb{E}[X] = \frac{1}{1/2} = 2$$

6 Randomized Selection

6.1 Algorithm

- Pick pivot g uniformly at random from the array
- Partition array into $L = \{x < g\}$ and $R = \{x > g\}$
- Recurse only on the side containing the kth smallest element

Worst-case time: $\Theta(n^2)$ Expected time: $\Theta(n)$

7 Good and Bad Pivots

A pivot g is **good** if:

$$\frac{n}{4} \leq \text{rank}(g) \leq \frac{3n}{4}$$

This guarantees recursion on at most $\frac{3n}{4}$ elements.

7.1 Probability of a Good Pivot

$$\Pr(\text{good pivot}) = \frac{n/2}{n} = \frac{1}{2}$$

Expected number of trials to find a good pivot:

$$\mathbb{E}[\text{trials}] = \frac{1}{1/2} = 2$$

8 Expected Recurrence

Context

We're analyzing the **randomized selection algorithm** (also known as QuickSelect), which finds the k-th smallest element in an unsorted array. The algorithm:

1. Picks a random pivot element
2. Partitions the array around the pivot
3. Recursively searches in one subarray (unlike QuickSort, which recurses on both)

Unlike the deterministic median-of-medians algorithm, we do **not reject bad pivots**. Instead, we analyze the expected running time over random pivot choices.

Setting Up the Recurrence

Let X be the random variable representing the size of the recursive subproblem after partitioning.

The recurrence relation is:

$$T(n) = T(X) + cn$$

where cn represents the cost of partitioning (scanning through all n elements).

Taking expectations on both sides:

$$\mathbb{E}[T(n)] = \mathbb{E}[T(X)] + cn$$

Understanding the Pivot Distribution

When we choose a random pivot, its **rank** (position in sorted order) is uniformly distributed:

$$\Pr(\text{rank of pivot} = i) = \frac{1}{n} \quad \text{for } i = 1, 2, \ldots, n$$

If the pivot has rank i:

- There are $i - 1$ elements smaller than the pivot (left partition)
- There are $n - i$ elements larger than the pivot (right partition)

The algorithm recurses on whichever partition contains the element we're searching for. In the worst case (for this pivot choice), we recurse on the **larger** partition:

$$\text{Recursive subproblem size} = \max(i - 1, n - i)$$

Identifying Good Pivots

A pivot is "good" if it splits the array reasonably balanced. Specifically, we want:

$$\frac{n}{4} \leq \max(i-1, n-i) \leq \frac{3n}{4}$$

When does this happen?
The maximum of the two partition sizes is at most $\frac{3n}{4}$ when:

$$i - 1 \leq \frac{3n}{4} \quad \text{AND} \quad n - i \leq \frac{3n}{4}$$

From the first inequality: $i \leq \frac{3n}{4} + 1$
From the second inequality: $i \geq n - \frac{3n}{4} = \frac{n}{4}$
So good pivots have rank in the range $\left[\frac{n}{4}, \frac{3n}{4}\right]$.
The number of good pivots is approximately $\frac{3n}{4} - \frac{n}{4} = \frac{n}{2}$.
Key insight: At least half of all possible pivots are "good"!

$$\Pr(\text{pivot is good}) \geq \frac{1}{2}$$

Bounding the Expected Running Time

We can split the expectation based on whether we get a good or bad pivot:

$$\begin{aligned}\mathbb{E}[T(n)] &= cn + \mathbb{E}[T(X)] \\ &= cn + \Pr(\text{good pivot}) \cdot \mathbb{E}[T(X) \mid \text{good pivot}] \\ &\quad + \Pr(\text{bad pivot}) \cdot \mathbb{E}[T(X) \mid \text{bad pivot}]\end{aligned}$$

Since good pivots occur with probability at least $\frac{1}{2}$:

$$\mathbb{E}[T(n)] \leq cn + \frac{1}{2} \cdot \mathbb{E}[T(3n/4)] + \frac{1}{2} \cdot \mathbb{E}[T(n)]$$

The first term uses the fact that good pivots give us a subproblem of size at most $\frac{3n}{4}$.

The second term is conservative: even if we get a bad pivot, the subproblem is at most size n (it could be smaller, but we're being pessimistic).

Calculation

Starting from:

$$\mathbb{E}[T(n)] \leq cn + \frac{1}{2}\mathbb{E}[T(n)] + \frac{1}{2}\mathbb{E}[T(3n/4)]$$

Subtract $\frac{1}{2}\mathbb{E}[T(n)]$ from both sides:

$$\frac{1}{2}\mathbb{E}[T(n)] \leq cn + \frac{1}{2}\mathbb{E}[T(3n/4)]$$

Multiply both sides by 2:

$$\mathbb{E}[T(n)] \leq 2cn + \mathbb{E}[T(3n/4)]$$

This is our key recurrence inequality.

Solving the Recurrence

We can solve by unfolding (expanding the recursion):

$$\begin{aligned}
\mathbb{E}[T(n)] &\leq 2cn + \mathbb{E}[T(3n/4)] \\
&\leq 2cn + 2c\left(\frac{3n}{4}\right) + \mathbb{E}[T((3/4)^2 n)] \\
&\leq 2cn + 2c\left(\frac{3n}{4}\right) + 2c\left(\frac{3}{4}\right)^2 n + \mathbb{E}[T((3/4)^3 n)] \\
&\vdots \\
&\leq 2cn\left[1 + \frac{3}{4} + \left(\frac{3}{4}\right)^2 + \left(\frac{3}{4}\right)^3 + \cdots\right]
\end{aligned}$$

This is a geometric series with ratio $r = \frac{3}{4} < 1$:

$$\sum_{k=0}^{\infty}\left(\frac{3}{4}\right)^k = \frac{1}{1-3/4} = \frac{1}{1/4} = 4$$

Therefore:

$$\mathbb{E}[T(n)] \leq 2cn \cdot 4 = 8cn$$

Final Result

$$\boxed{\mathbb{E}[T(n)] = O(n)}$$

Important notes please read

Remarkable fact: Even though we might occasionally pick terrible pivots that don't reduce the problem size at all, the algorithm still runs in expected linear time!

The key reasons:

1. **Half the pivots are good**: We only need to reduce the problem by a constant fraction, not by half.
2. **Geometric decay**: Each time we get a good pivot, the problem shrinks to at most $\frac{3}{4}$ of its size.
3. **Expected constant good pivots**: On average, we expect to get a good pivot every 2 tries, so we make constant expected progress per level.

This is much simpler and faster in practice than the deterministic $O(n)$ median-of-medians algorithm, though the latter has a better worst-case guarantee.

9 Divide & Conquer vs Prune & Search

9.1 Divide & Conquer

- Divide problem into subproblems
- Solve all subproblems recursively
- Combine results

Example: Merge Sort, QuickSort

9.2 Prune & Search

- Guess/select a candidate
- Eliminate a fixed fraction of elements
- Recurse on remaining subset

Example: Randomized Selection

10 QuickSort

10.1 Algorithm

- Pick a pivot
- Partition array into $L < p$ and $R > p$
- Recursively sort L and R

QuickSort is **Divide & Conquer** since both sides are kept.

11 QuickSort Recurrence

Let $T(n)$ be the time to sort n elements.
Partitioning takes $\Theta(n)$.

$$T(n) = T(k) + T(n-k-1) + cn$$

11.1 Average Case

Balanced splits: $k \approx n/2$

$$T(n) = 2T(n/2) + cn$$

Solving the Recurrence Relation

Given

$$T(n) = 2T(n/2) + cn$$

Recursion Tree Approach

Level 0:

- Work at root: cn

Level 1:

- Two subproblems of size $n/2$
- Work at this level: $2 \cdot c(n/2) = cn$

Level 2:

- Four subproblems of size $n/4$
- Work at this level: $4 \cdot c(n/4) = cn$

Level 3:

- Eight subproblems of size $n/8$
- Work at this level: $8 \cdot c(n/8) = cn$

Pattern Recognition

At level i:

- Number of subproblems: 2^i
- Size of each subproblem: $n/2^i$
- Work at this level: $2^i \cdot c(n/2^i) = cn$

Finding Tree Height

The recursion bottoms out when the subproblem size reaches 1:

$$\frac{n}{2^i} = 1 \implies 2^i = n \implies i = \log_2 n$$

So the tree has height $\log_2 n$.

Total Work

Since each level does cn work and there are $\log_2 n + 1$ levels (from 0 to $\log_2 n$):

$$T(n) = cn \cdot (\log_2 n + 1) = cn \log_2 n + cn$$

Therefore:

$$\boxed{T(n) = \Theta(n \log n)}$$

Key Insight

Even though the number of subproblems doubles at each level, their sizes halve, so the total work per level remains constant at cn.

11.2 Worst Case

Highly unbalanced split: $k = 0$ or $n - 1$

$$T(n) = T(n-1) + cn$$

Unrolling:

$$T(n) = cn + c(n-1) + \cdots + c = \Theta(n^2)$$

$$\boxed{\text{Worst-case QuickSort time} = \Theta(n^2)}$$