Lecture 1: Randomness, Probability, and Algorithms

Take a standard deck of 52 cards, with 13 ♠, 13 ♡, 13 ♢, and 13 ♣. Shuffle it (well), so that the order is completely (uniformly)[1] random. *How many consecutive pairs of the same suit do you expect?*

[1] People usually say “random” when they mean *uniformly* random, and that can be quite ambiguous. I’ll try not to.

For instance,

4♡, 3♡, 8♣, 2♣, 3♠, 10♡, 8♢, 7♠, K♡, 5♢, 8♡, J♡, 9♣,
5♣, J♠, 2♡, Q♠, 2♠, 10♠, 6♠, 6♣, 5♡, 4♣, 9♠, Q♢, 8♠,
6♢, 10♢, 7♣, J♣, K♣, 4♢, K♢, K♠, A♢, A♠, A♣, 4♠, A♡,
3♣, 9♢, 3♢, J♢, 9♡, Q♡, Q♣, 2♢, 10♣, 5♠, 7♢, 6♡, 7♡

has 15 such consecutive pairs.

Please check.

So. . . what’s the expected number of consecutive same-suit pairs in a shuffled deck?

Let’s try to estimate this:

```python
import numpy as np
import random
deck = 13*['S', 'H', 'D', 'C']
consecutives = []
for _ in range(50000):
    shuffled_deck = random.sample(deck, len(deck));
    consecutives += [np.sum([shuffled_deck[i] == shuffled_deck[i+1] for i
     in range(len(deck)-1)])]
print("Empirical mean: %f" % np.mean(consecutives))
```

I ran it: this gave 11.98176. I ran it again: 12.0022. And these 50,000 attempts look roughly like this (Fig. 1):

```python
import matplotlib.pyplot as plt
plt.hist(consecutives, density = True, bins=25, edgecolor='k');
plt.axvline(np.mean(consecutives), color='r', linestyle='dashed',
     linewidth=2)
plt.show()
```

Looks like this expected number is something like 12. How do we explain this?

Try to see what changes if you change the number of cards in the deck (for instance, 34 cards from each suit instead of 13). Could you have predicted it?

Theorem 1. *The expected number of consecutive same-suit pairs is 12.*

Proof. “Linearity of expectation.” ☐

$\forall i, \Pr[\, X_i = X_{i+1} \,] = \frac{13-1}{52-1}$

One for the rainy days.

It’s raining, and all n students of the class come to the lecture with an umbrella. They all leave it in front of the lecture hall. At the end of the lecture, they leave the room in (uniformly) random order, and crossing the door each takes the closest remaining umbrella. In expectation, *how many leave with their own umbrella*?

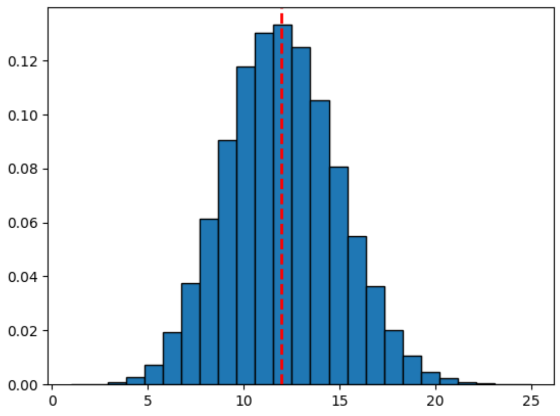

Figure 1: (Normalized) histogram of the number of consecutive pairs of the same suit, over 50,000 trials. The red dashed line is the empirical mean.

Theorem 2. *The expected number of fixed points in a uniformly random permutation of* $\{1, 2 \dots, n\}$ *is one.*

What's a randomized algorithm?

Randomized algorithms are algorithms whose behaviour *does not depend solely on the input.* It also depends (in part) on random choices or the values of a number of *random bits.*

So we can think of the algorithm A as taking an input x and a string of uniformly random bits $R \in \{0,1\}^*$. Now, since A is randomised, this could mean several things:

1. The time $\tau_A(x; R)$ that A takes on input x is itself random, and depends on the random bits R
2. The output $A(x; R)$ of A on input x is random, and can be different based on the random bits R
3. Something else (*e.g.*, the amount of memory used by A could depend on the random bits)
4. all, or any combination of the above

A randomized algorithm can be viewed as a probability distribution over deterministic ones.

One equivalent way to view randomized algorithms, which proves very useful in many cases, is as a *probability distribution over deterministic algorithms*. That is, one can see a randomized algorithm A taking input x and using random bits R as a *family* of deterministic algorithms

$$\{A_r\}_{r\in\{0,1\}^*}$$

and the randomized algorithm, on input x, starts by (1) choosing a random R; (2) running the deterministic algorithm A_R on x.

Typically, what we want is then analyze A on *worst-case* input x and *uniformly random* R. The two things to keep in mind: (1) is the output of A always correct? Is it correct with high probability (over the choice of random bits R)? (2) is the running time of A always bounded? Is it bounded with high probability, or in expectation (over the choice of random bits R)?

($\star\star$) For those interested, for decision problems this is related to complexity classes ZPP, RP, and BPP. Think of that last one as "randomized P."

Las Vegas algorithms: the algorithm is *always correct*, but the running time is only bounded *in expectation*.

Monte-Carlo algorithms: the algorithm is only *correct with high probability*, but the running time is bounded *with probability one*.

This is very abstract right now, so before moving to the subtantial example of QuickSort (soon), here is an example:

> Given an (arbitrary) array of size n containing all numbers from 1 to n, output the index of an even number.

Claim 2.1 (Bad news). *Any deterministic algorithm for this task must have worst-case time complexity $\Omega(n)$.*

Claim 2.2 (Good news). *There is a Las Vegas algorithm for this task with* expected *time complexity $O(1)$.*

Claim 2.3 (Good news). *There is a Monte Carlo algorithm for this task with* worst-case *time complexity $O(1)$, and probability of success* 0.99.

Relation to other notions of analysis.

What you have focused so far in algorithms classes is *worst-case analysis*: this quantifies the worst possible behaviour of an algorithm (its time complexity, or space complexity, usually), that is, how badly it will do *on the worst possible input you can give it*. This is very useful to know, since once you have figured that out then you know that, no matter what, your algorithm cannot do worse than that, even on the most adversarial situation. But there are other types of analysis, less frequent and usually much harder to interpret, that can be used: *expected time analysis* (when the algorithm is randomized: **this class**), *average time analysis* (when the input itself comes from some known probability distribution: **not now**), *amortized analysis* (when we use the algorithm repeatedly on a sequence of inputs, and look at the worst-case sequence of input for the algorithm divided by the length of the sequence).

Summary: if $\tau_A(x)$ is the time taken by A on input x of size $|x|$ and $A_k(x_1, \dots, x_k)$ corresponds to running the algorithm successively on inputs $x_1, \dots, x_k$, then the time analyses discussed above correspond to:

$$T(n) = \max_{x:|x|=n} \tau_A(x) \qquad \text{(Worst-case)}$$

$$T_{\text{expected}}(n) = \max_{x:|x|=n} \mathbb{E}_R[\tau_A(x; R)] \qquad \text{(Expected: } A \text{ is randomized)}$$

$$T_{\text{average}}(n) = \mathbb{E}_x[\tau_A(x)] \qquad \text{(Average-case: } x \text{ is random)}$$

$$T_{\text{amortized}}(n) = \lim_{k\to\infty} \frac{1}{k} \max_{|x_1|=\cdots=|x_k|=n} \tau_{A_k}(x_1, \dots, x_k) \qquad \text{(Amortized)}$$

Again, you probably focused on the first in previous studies, and in this unit we will also consider the second.

But why?

Some of the reasons for using randomization:

- Quickly finding representative or relevant parts of the input (*e.g.*, sampling data from a large dataset)
- Avoid pathological corner cases
- Avoid predictable outcomes
- Allow for simpler or more efficient algorithms
- ...

Can you think of anything else?

Some drawbacks:

- The behaviour of the algorithm is, well, random. The output might be, or the running time, or something else. Are you happy with this non-deterministic behaviour?
- Where do you find these "random bits" the algorithm needs?

Any idea?

Analyzing Randomized Quicksort

Is this a Las Vegas or a Monte Carlo algorithm?

Remember QuickSort from your previous classes? It's a very nice comparison-based sorting algorithm, which works as follows: This

Algorithm 1: QuickSort

```
Input: Input array $A$ of size $n$
1: if $n \leq 1$ then return $A$
2: Select an index $1 \leq i \leq n$, and let $p \leftarrow A[i]$ be the pivot
3: Partition $A$ into 3 subarrays: $A_1$ (elements smaller than $p$), $A_2$
   (equal to $p$), and $A_3$ (greater than $p$)          ▷ $O(n)$ time
4: Recursively call QuickSort on $A_1$ and $A_3$ to sort them
5: Merge the (sorted) $A_1$, $A_2$, $A_3$ into $A$              ▷ $O(n)$ time
6: return $A$
```

is the prototypical example of a divide-and-conquer algorithm: the only thing unspecified above is *how to choose the pivot*. And this is very important: the time complexity of the whole algorithm depends crucially on it!

The naive way to choose the pivot deterministically (just pick, say, $i = \lceil n/2 \rceil$) is quite terrible, leading to a worst-case time complexity of $O(n^2)$. Not so "quick." A much more involved way to do so, getting the *median* as pivot using linear-time selection does give sorting in worst-case $O(n \log n)$ time: but now the algorithm is very complicated, and not so fast in practice anymore.

If you don't remember what it is, that's alright – but it's worth looking it up.

But this is a class on randomized algorithm, so let's do the obvious randomized thing, and pick the pivot uniformly at random: in Line 2, choose i uniformly at random in $\{1, 2, \ldots, n\}$. This gives us *Randomized QuickSort*. The proof of correctness is the same as usual QuickSort, but what about the (expected) time complexity? *How fast is it?*

Well, the expected time complexity $T(n)$ satisfies the recurrence:

$$T(n) = \mathbb{E}[T(|A_1|) + T(|A_2|)] + O(n) \tag{1}$$

where the expectation is over the random choice of pivot in Line 2; and $T(1) = O(1)$.

Suppose for simplicity that all elements are distinct. Then, if we pick as pivot the k-th largest element, $n_L = k-1$ and $n_R = n-k$. What is the probability to pick the k-th largest element as pivot? We select the pivot uniformly at random, so that's $1/n$.

Curious? See how to adapt the proof below to the general case.

So we can rewrite (the cn is for the $O(n)$ in (1), which comes from the Divide and the Conquer steps):

$$\begin{aligned} T(n) &= cn + \frac{1}{n}\sum_{k=1}^{n}(T(k-1) + T(n-k)) \\ &= cn + \frac{1}{n}\sum_{k=0}^{n-1} T(k) + \frac{1}{n}\sum_{\ell=0}^{n-1} T(\ell) \end{aligned}$$

That is,

$$T(n) = cn + \frac{2}{n}\sum_{k=0}^{n-1} T(k) \tag{2}$$

Now, *how do we solve this?*

Any idea?

First method: guess, and prove inductively. You know the drill. Magically guess $T(n) \leq an\log n$, try to prove it by induction, see it doesn't quite work depending on which bound you use for $\sum_{k=1}^{n} k\log k$, maybe change your "magic guess" to $T(n) \leq an\log n - bn$ to make it work (or get a better bound for the sum).

Second method: integrals are nicer than sums. Instead of solving Eq. (2) directly, let's instead compare this discrete relation to a (much nicer to solve) differential equation. The idea is that often, "sums and integrals are basically the same thing,"

Fact 2.1. *Let f be a non-decreasing function. Then, for all $n \geq 0$,*

$$\int_0^n f(x)dx \leq \sum_{k=0}^{n} f(k) \leq \int_1^{n+1} f(x)dx$$

Let's apply that here, and solve the functional equation

We make a few implicit (reasonable) assumptions on T here: which ones?

$$T(x) = cx + \frac{2}{x}\int_0^x T(u)du, \qquad x > 0$$

Introducing the antiderivative $F(x) = \int_0^x T(u)du$, we can rewrite this as

$$F'(x) = cx + \frac{2}{x}F(x) \tag{3}$$

which is "easier" to solve,[2] and will lead to $T(x) = O(x\log x)$ and so $T(n) = O(n\log n)$. The point is that differential equations are often much easier to solve than discrete recurrence relations.

[2] Check with an automated solver like Mathematica first: `https://www.wolframalpha.com/input?i=solve+F%27%28x%29+%3D+c+x+%2B+2%2Fx+F%28x%29+`.

($\star\star$) *How:* dividing everything by x^2, Eq. (3) becomes

$$\frac{F'(x)}{x^2} - \frac{2}{x^3}F(x) = \frac{c}{x}$$

but then, we can use that $\frac{d}{dx}\frac{F(x)}{x^2} = \frac{F'(x)}{x^2} - \frac{2F(x)}{x^3}$, so integrating we get

$$\frac{F(x)}{x^2} = c\ln x + C$$

for some constant $C \in \mathbb{R}$, and so $F(x) = cx^2 \ln x + Cx^2$. Then $f(x) = F'(x) = 2cx\ln x + (2C+1)x = O(x\log x)$, and we are done.

What we have shown is the following:

Theorem 3. *Randomized QuickSort has expected running time $O(n \log n)$.*

But still worst-case running time $O(n^2)$.

What about the number of *comparisons*? Clearly, what we just showed implies that the expected number of comparisons is also $O(n\log n)$, but if that's all we are interested in, could we have proven it in a nicer way?

Theorem 4. *The expected number of comparisons performed by Randomized QuickSort is $O(n\log n)$.*

Proof. When we run QuickSort, all the comparisons at one level of the recurrence are between the current pivot and all the other $n-1$ elements, and we never compare two elements twice. So we *could* try to solve the corresponding recurrence on the expected number of comparisons $C(n)$:

$$C(n) = \mathbb{E}[C(|A_1|) + C(|A_2|)] + (n-1) \tag{4}$$

We could, but we will not. Instead, here's a slightly nicer argument based on linearity of expectation.

This is the same as for $T(n)$, but with an explicit constant instead of c.

Suppose for simplicity that all n elements are distinct and let us denote them, in ranked order, by

Intuitively, duplicate elements can only make the expected number of comparisons smaller. Can you argue why?

$$a_1 < a_2 < a_3 < \cdots < a_n$$

(note that this is only for the analysis, and that a_i is not necessarily the element at index i of A: the array is generally not already sorted!) For any two indices $i < j$, let $X_{ij} \in \{0,1\}$ be the indicator variable of whether Randomized QuickSort ever compares a_i and a_j. Since the algorithm never compares twice the same two elements, we have that the total number of comparisons is

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

and $C(n) = \mathbb{E}[X]$. By linearity of expectation,

$$C(n) = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \mathbb{E}[X_{ij}] = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \Pr[\,a_i \text{ and } a_j \text{ are compared}\,]$$

So it boils down to understanding the probability Randomized QuickSort ever compares two fixed distinct elements of the array. Suppose we are at the ℓ-th recursive step of the algorithm, with a_i, a_j both in the current subarray of size n_ℓ, and we pick a pivot p:

- If $a_i < p < a_j$, then we will recurse on two disjoint subarrays, one containing a_i and the other a_j, so that they will never be compared (decision made!). This happens with probability $\frac{j-i-1}{n_\ell}$.
- If p is either a_i or a_j, then they will be compared – since elements are only compared to the pivot (decision made!). This happens with probability $\frac{2}{n_\ell}$.
- Otherwise, they are not compared at this stage, but they both end up in the same subarray the algorithm recurses on, so the comparison could happen later on. This "no decision either way yet" happens with probability $1 - \frac{j-i+1}{n_\ell}$.

From the above, we have

$$\Pr\left[\begin{array}{c} a_i \text{ and } a_j \\ \text{are compared} \\ \text{at stage } \ell \end{array} \,\middle|\, \begin{array}{c} \text{Decision made} \\ \text{at stage } \ell \end{array}\right] = \frac{\frac{2}{n_\ell}}{\frac{j-i-1}{n_\ell} + \frac{2}{n_\ell}} = \frac{2}{j-i+1}$$

Overall, we can write

$$\begin{aligned}
\Pr\left[\begin{array}{c} a_i \text{ and } a_j \\ \text{are compared} \end{array}\right] &= \sum_{\ell=0}^{\infty} \Pr\left[\begin{array}{c} a_i \text{ and } a_j \\ \text{are compared} \\ \text{at stage } \ell \end{array} \,\middle|\, \begin{array}{c} \text{Decision made} \\ \text{at stage } \ell \end{array}\right] \Pr\left[\begin{array}{c} \text{Decision made} \\ \text{at stage } \ell \end{array}\right] \\
&= \sum_{\ell=0}^{\infty} \frac{2}{j-i+1} \cdot \Pr\left[\begin{array}{c} \text{Decision made} \\ \text{at stage } \ell \end{array}\right] \\
&= \frac{2}{j-i+1},
\end{aligned}$$

the last line since probabilities sum to one. We are almost there: remember that we are interested in $C(n)$, which we now are able to express as

$$C(n) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} = 2 \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{1}{k+1}$$

This may not look so nice, but letting $H_k = \sum_{i=1}^{k} \frac{1}{k} \leq \ln k + 1$ denote the k-th Harmonic number, we get

$$C(n) = 2 \sum_{i=1}^{n-1} (H_{n-i+1} - 1) = 2 \sum_{i=2}^{n} (H_i - 1) \leq 2 \sum_{i=2}^{n} \ln i \leq 2n \ln n$$

(we even get an explicit upper bound, not just $O(n \log n)$). □

A few useful probabilistic facts

Let X be a random variable (r.v.) taking real values: for instance, in $\mathbb{R}$ or $\mathbb{N}$. We assume X has an expectation and a variance.[3] A few useful things:

[3] This is not necessarily always true! Some random variables do not even have a well-defined expectation. For instance, the random variable defined on $\mathbb{Z}$ by $\Pr[X = k] = \frac{1}{C} \cdot \frac{1}{1+k^2}$ with $C = 1 + \pi \coth \pi$ (so that the probabilities sum to 1) is well-defined, but does not have an expectation since $\sum_{k\in\mathbb{Z}} k \cdot \Pr[X = k]$ is not defined (does not converge).

Fact 4.1. *If X takes values in $\mathbb{N} = \{0, 1, 2, \dots, \}$,*

$$\mathbb{E}[X] = \sum_{n=0}^{\infty} n \Pr[X = n] = \sum_{n=1}^{\infty} \Pr[X \geq n]$$

To remember whether the sum in the last expression starts at $n = 0$ or $n = 1$: either reprove it (a bit time-consuming), or take X to be the "useless" random variable equal to 0 with probability 1. Then $\mathbb{E}[X] = 0$, but $\sum_{n=0}^{\infty} \Pr[\, X \geq n \,] = \Pr[\, X \geq 0 \,] = 1$. So we shouldn't have the term $n = 0$.

Fact 4.2. *If X has a finite variance,*

$$\operatorname{Var}[X] = \mathbb{E}\left[(X - \mathbb{E}[X])^2\right] = \mathbb{E}\left[X^2\right] - \mathbb{E}[X]^2$$

As a direct consequence, $\operatorname{Var}[X] \leq \mathbb{E}\left[X^2\right]$ (sometimes useful).

Lemma 4.1 (Jensen's Inequality). *If $f\colon \mathbb{R} \to \mathbb{R}$ is convex (and $\mathbb{E}[f(X)]$ is well-defined)*

$$f(\mathbb{E}[X]) \leq \mathbb{E}[f(X)]\,.$$

For f concave, the inequality is reversed.

To remember the direction: check with $f(x) = x^2$ (convex). The variance is non-negative, so $0 \leq \operatorname{Var}[X] = \mathbb{E}\left[X^2\right] - \mathbb{E}[X]^2$.

Fact 4.3 (Linearity of Expectation). *For any X, Y and $a, b \in \mathbb{R}$,*

$$\mathbb{E}[aX + bY] = a\mathbb{E}[X] + b\mathbb{E}[Y]$$

(We do not *need X, Y to be independent!)*

This extends to more random variables: for instance, $\mathbb{E}[\sum_{i=1}^n X_i] = \sum_{i=1}^n \mathbb{E}[X_i]$. (No independence needed!)

Fact 4.4 (Variance). *For any X and $a \in \mathbb{R}$,*

$$\operatorname{Var}[aX] = a^2 \operatorname{Var}[X]\,.$$

Moreover, if X, Y are independent*,*

$$\operatorname{Var}[aX + bY] = a^2 \operatorname{Var}[X] + b^2 \operatorname{Var}[Y]\,.$$

More generally, **if** $X_1, \dots, X_n$ are mutually independent (or, weaker condition, *pairwise independent*: any two X_i, X_j with $i \neq j$ are independent, but $X_1, \dots, X_n$ as a whole might not be mutually independent.), then

$$\operatorname{Var}\left[\sum_{i=1}^n X_i\right] = \sum_{i=1}^n \operatorname{Var}[X_i]. \tag{5}$$

The proof is not too hard: basically, since $\operatorname{Var}[X] = \mathbb{E}\left[(X - \mathbb{E}[X])^2\right]$, consider $\mathbb{E}\left[(\sum_{i=1}^n (X_i - \mathbb{E}[X_i]))^2\right]$ and expand the square, then use linearity of expectation:

$$\begin{aligned}
\operatorname{Var}\left[\sum_{i=1}^n X_i\right] &= \mathbb{E}\left[\sum_{i=1}^n \sum_{j=1}^n (X_i - \mathbb{E}[X_i])(X_j - \mathbb{E}[X_j])\right] \\
&= \mathbb{E}\left[\sum_{i=1}^n (X_i - \mathbb{E}[X_i])^2\right] + \mathbb{E}\left[\sum_{i \neq j} (X_i - \mathbb{E}[X_i])(X_j - \mathbb{E}[X_j])\right] \\
&= \sum_{i=1}^n \mathbb{E}\left[(X_i - \mathbb{E}[X_i])^2\right] + \sum_{i \neq j} \mathbb{E}\left[(X_i - \mathbb{E}[X_i])(X_j - \mathbb{E}[X_j])\right]
\end{aligned}$$

The first term is exactly $\sum_{i=1}^{n} \operatorname{Var}[X_i]$; the second, by pairwise independence, is 0, since $\mathbb{E}\left[(X_i - \mathbb{E}[X_i])(X_j - \mathbb{E}\left[X_j\right])\right] = \mathbb{E}[(X_i - \mathbb{E}[X_i])]\mathbb{E}\left[(X_j - \mathbb{E}\left[X_j\right])\right] = 0 \cdot 0$.

Now, a few very trivial-looking (but useful!) facts. Suppose X takes values in $\{0, 1\}$, with $\Pr[\, X = 1 \,] = p$ (this is a Bernoulli random variable). Then

- $X^2 = X$ (of course!), so $\mathbb{E}\left[X^2\right] = \mathbb{E}[X] = \Pr[\, X = 1 \,] = p$
- That implies $\operatorname{Var}[X] = \mathbb{E}\left[X^2\right] - \mathbb{E}[X]^2 = p - p^2 = p(1-p)$, which is at most $1/4$.
- That implies that for a Binomial $X \sim \operatorname{Bin}(n, p)$, which is just the sum of *n independent, and identically distributed* (i.i.d.) Bernoullis with parameter p,

$$\mathbb{E}[X] = np, \qquad \operatorname{Var}[X] = np(1-p)\,.$$

Check it! $x(1-x) \leq 1/4$ for $x \in [0, 1]$, and the maximum is at $x = 1/2$.

Finally, an *indicator* random variable (for some "event" E) is just a Bernoulli random variable which is equal to 1 if the event occurs, and 0 otherwise (so, Bernoulli with parameter $\Pr(E)$). Usually denoted $\mathbb{1}_E$.