Middlebury

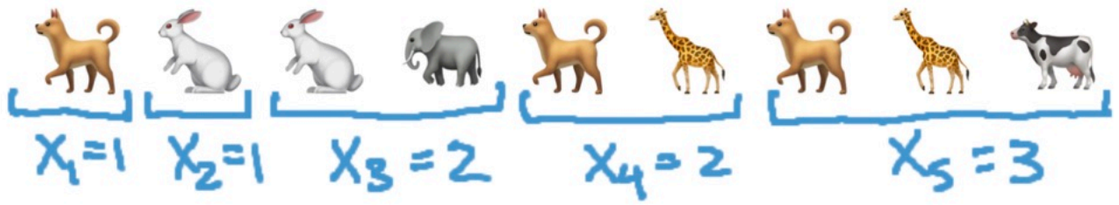
CSCI 200: Math Foundations of Computing

Spring 2026

Lecture 12W: Expected Complexity

From last class: collecting Happy Meal toys.

What is the expected number of happy meals (M) you need to buy to collect all n toys (🐕, 🐇, 🐘, 🦒, 🐄)? Example meal sequence to get all 5 toys:



Let X_i be the # of meals to buy to get the i-th toy. Within segment i (s_i):

- We have $(i-1)$ different toys so far. Probability of getting i-th toy is $p_i = (n+1-i)/n$.
- Probability of **needing** 1st meal of s_i? 1, 2nd meal of s_i? $(1-p_i)$, 3rd meal of s_i? $(1-p_i)^2$...

$n=5 \quad \approx 12$

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

what is $E[M]$?

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

$E[M] = E\left[\sum_{i=1}^{n} X_i\right] \overset{\text{by LoE}}{=} \sum_{i=1}^{n} E[X_i]$ what is this?

Let Y_{ij} be an IRV means $\begin{cases} 1 & \text{if need to buy HM}_j \text{ to get toy } i \\ 0 & \text{otherwise} \end{cases}$

$X_i = \sum_{j=0}^{\infty} Y_{ij}$ ← don't know when we'll get toy i

$E[M] = \sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} E\left[\sum_{j=0}^{\infty} Y_{ij}\right] \overset{\text{by LoE}}{=} \sum_{i=1}^{n} \sum_{j=0}^{\infty} E[Y_{ij}]$ prob.

$= \sum_{i=1}^{n} \sum_{j=0}^{\infty} (1-p_i)^j$

$\frac{1}{1-r}$ geom. series $\to \infty$

T/F

$= \sum_{i=1}^{n} \frac{1}{p_i}$

X_5 $(1-p_i)^0$ $(1-p_i)$ $(1-p_i)^2$

$\frac{1}{}$ need meal$_0$ in X_5?

$\frac{1}{}$ need meal$_1$ in X_5?

$\frac{1}{}$ need to buy meal$_2$ in X_5?

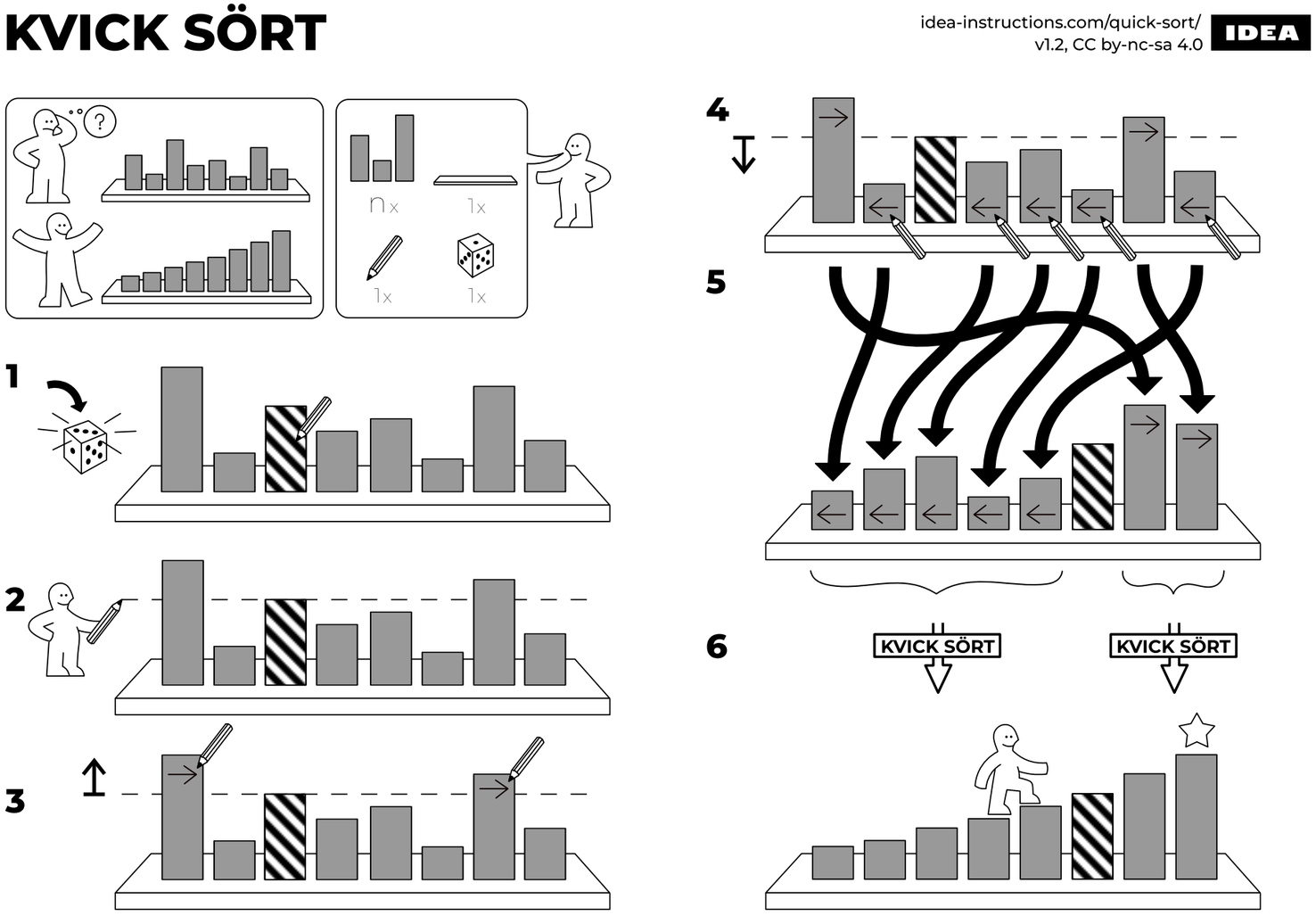
T/F

T/F

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

Goals for today:

- Use random variables and linearity of expectation to perform an average-case analysis of algorithms. Specifically, we'll analyze `Quick-Sort` with a random **pivot**.

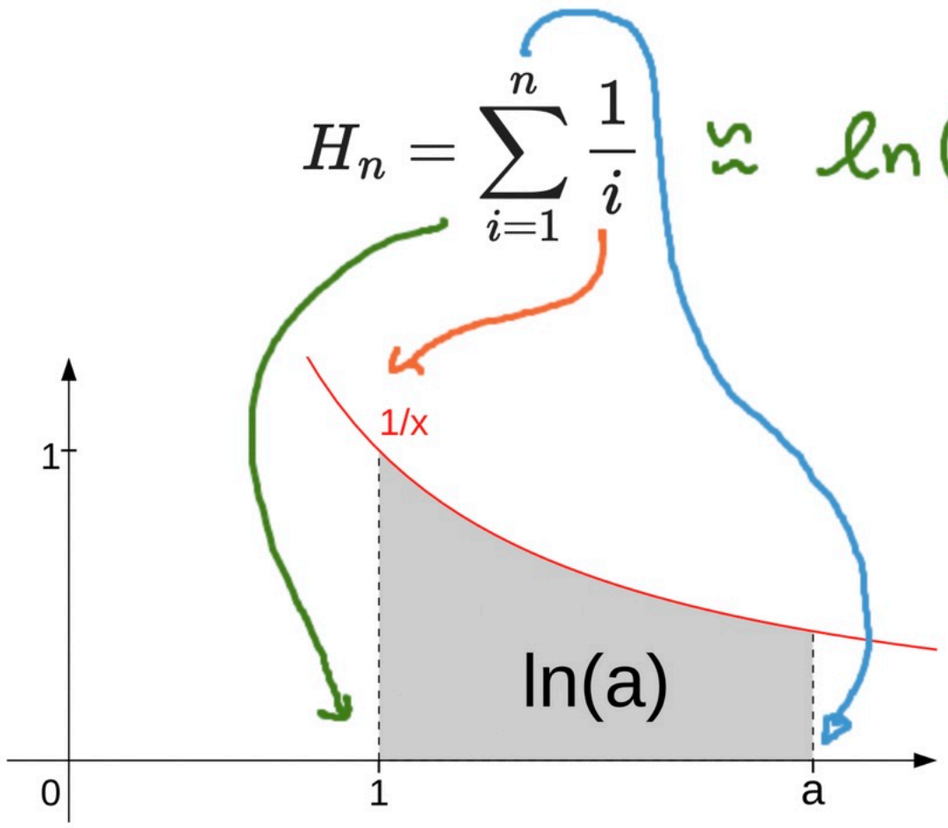


First a note about the sum from the Happy Meal problem.

$$n\sum_{i=1}^{n}\frac{1}{n+1-i} = n\left(\frac{1}{n}+\frac{1}{n-1}+\frac{1}{n-2}+\cdots+\frac{1}{2}+1\right) = n\underbrace{\sum_{i=1}^{n}\frac{1}{i}}_{H_n}$$

Harmonic numbers:

$$H_n = \sum_{i=1}^{n}\frac{1}{i} \approx \ln(n)$$



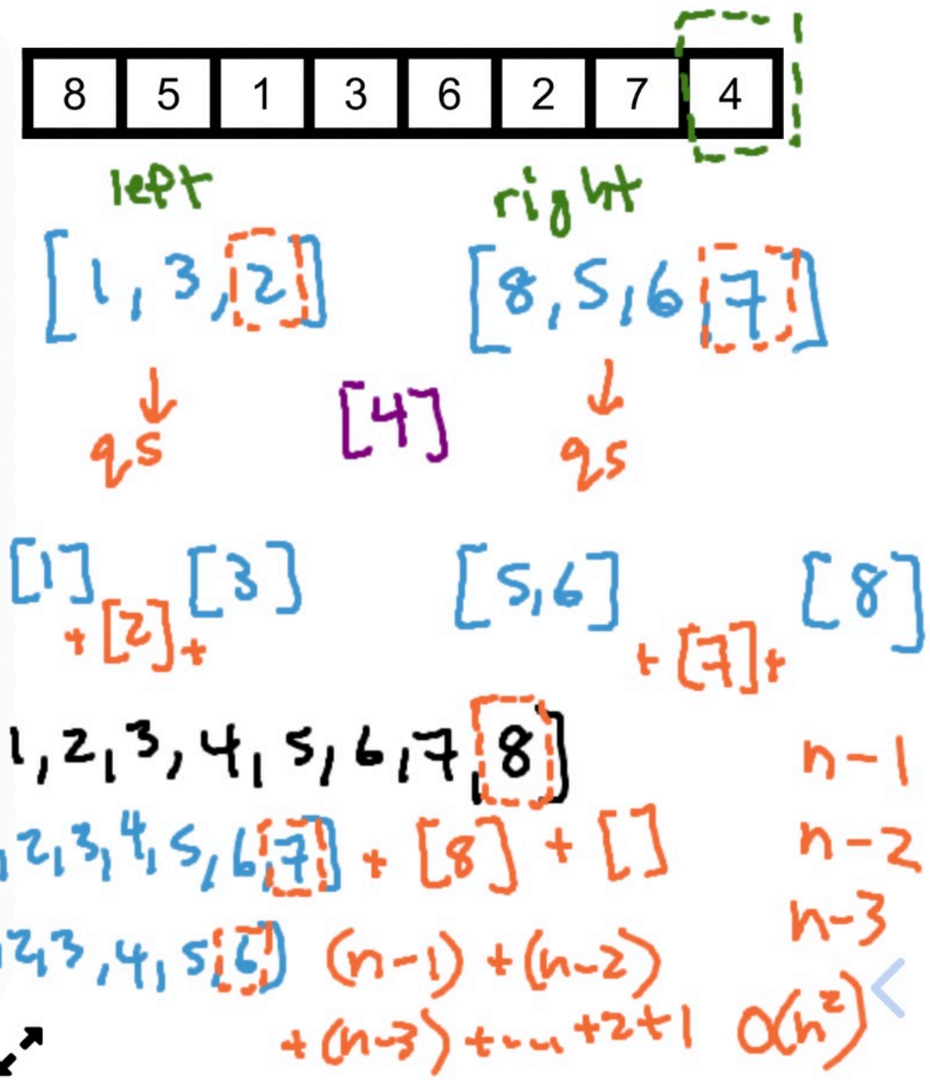
happy meals grows as $O(n \log n)$

The `Quick-Sort` algorithm (a very basic implementation, don't implement it this way). Count number of comparisons ($<$ or $>$).

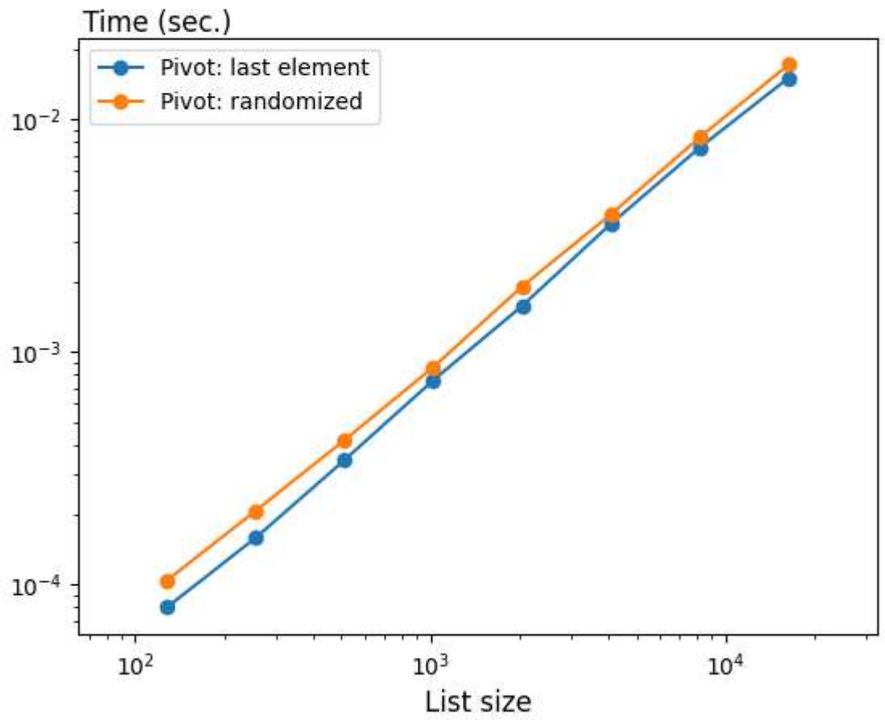
1. Pick a pivot.
2. Partition items so than any item < pivot is in left subarray and any item > pivot is in the right subarray.
3. Call `Quick-Sort` on each subarray.

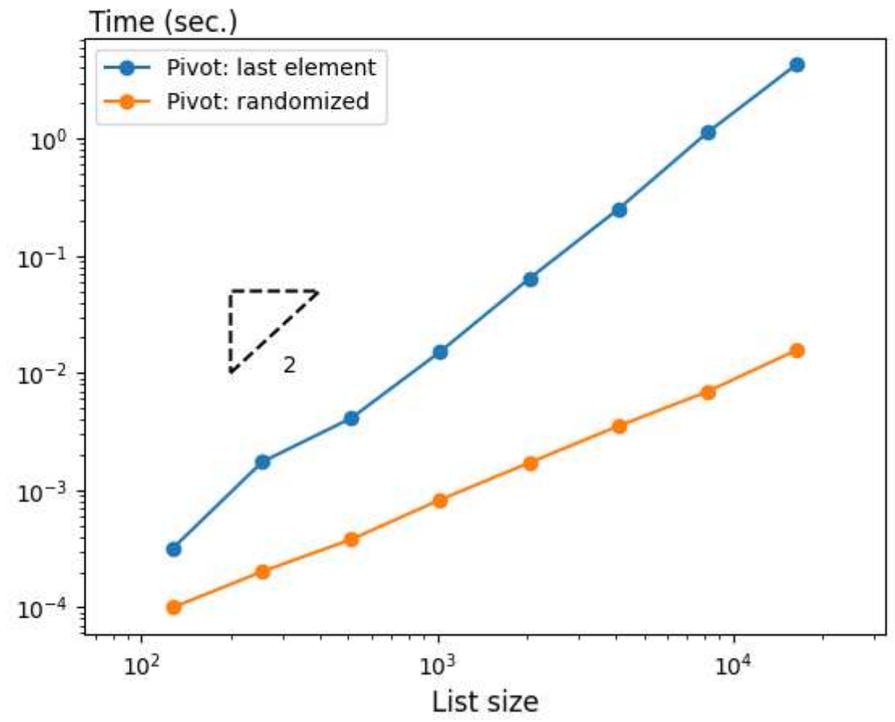
```python
def quicksort(array, randomized=True):

  # base case
  if len(array) <= 1:
    return array

  # select pivot
  pivot_index = len(array) - 1
  if randomized:
    pivot = random.randint(0, len(array) - 1)
  pivot = array[pivot_index]

  # partition
  left = [elem for elem in array if elem < pivot]
  right = [elem for elem in array if elem > pivot]

  # sort recursively
  left = quicksort(left, randomized)
  right = quicksort(right, randomized)

  return left + [pivot] + right
```

worst case:



Worst-case and average case complexity of `Quick-Sort` (experimentally).






```
>>> array = [i for i in range(n)]
>>> random.shuffle(array)
>>> quicksort(array, ... )
```

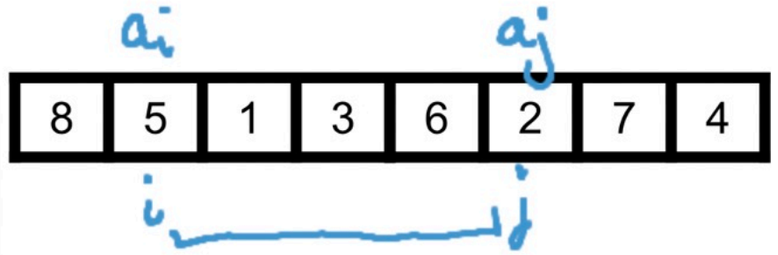
```
>>> array = [i for i in range(n)]
>>> quicksort(array, ... )
```

For (already) sorted arrays (and other special patterns too), complexity is $O(n^2)$.

Analyzing the total number of comparisons in Quick-Sort.

```python
def quicksort(array, randomized=True):

  # base case
  if len(array) <= 1:
    return array

  # select pivot
  pivot_index = len(array) - 1
  if randomized:
    pivot = random.randint(0, len(array) - 1)
  pivot = array[pivot_index]

  # partition
  left = [elem for elem in array if elem < pivot]
  right = [elem for elem in array if elem > pivot]

  # sort recursively
  left = quicksort(left, randomized)
  right = quicksort(right, randomized)

  return left + [pivot] + right
```

From $<$ $>$



- Let X_{ij} be an IRV which is 1 if a_i and a_j are ever compared.
- Let $M =$ # comparisons $= \sum_i \sum_j X_{ij}$.
- When are a_i and a_j compared? When either of them is the `pivot` (2 options) and they are both in the same subarray.
- Probability of being selected as pivot?

$$\text{prob}(a_i \text{ and } a_j \text{ compared}) = \frac{2}{j-i+1}$$

2 options

Sum over all possible pairs:

IRV!!!

$$E[M] = 2E\left[\sum_{i=1}^{n}\sum_{j=i+1}^{n} X_{ij}\right] \overset{\text{by LoE}}{=} 2\sum_{i=1}^{n}\sum_{j=i+1}^{n} E[X_{ij}] = 2\sum_{i=1}^{n}\sum_{j=i+1}^{n} \frac{2}{j-i+1}$$

Determining a closed-form expression for the sum.

Recall Harmonic numbers: $H_n = \sum_{i=1}^{n} \frac{1}{i}$. New identity: $\sum_{i=1}^{n} H_i = (n+1)H_n - n$

.

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

substitute $k = j-i+1$

$i=n-1$ @ $j=i+1$ $k = i+1-i+1 = 2$

↘ @ $j=n$ $k = n-i+1$

$i=1$ $i=2$ $i=3$

$$= 2\left(\sum_{k=2}^{n}\frac{2}{k} + \sum_{k=2}^{n-1}\frac{2}{k} + \sum_{k=2}^{n-2}\frac{2}{k} + \cdots + \sum_{k=2}^{3}\frac{2}{k} + \sum_{k=2}^{2}\frac{2}{k}\right)$$

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

$$= 4\left(H_n - 1 + H_{n-1} - 1 + H_{n-2} - 1 + \cdots + H_3 - 1 + H_2 - 1 + H_1 - H_1\right)$$

1

$$= 4\sum_{i=1}^{n}(H_i - 1) = 4\sum_{i=1}^{n} H_i - 4\sum_{i=1}^{n} 1$$

$\sum_{i=1}^{n} H_i$: $(n+1)H_n - n$ by identity *

$\sum_{i=1}^{n} 1$: n

$$= 4(n+1)H_n - 4n - 4n$$

$H_n \approx \log(n)$

$O(n \log n)$