Algorithm Engineering Lecture 5: Workshop on Customized Data Structures

CSE 431
Spring 2024
Kevin Liu, Ph.D.
Michigan State University

Optional reading:
Skiena 2012
chapter 3

Includes material adapted from Stony Brook 373, U.Toronto BCH441H, and previous editions of CSE431

A Problem-Solving Toolbox

	Search	Insert	Delete	Index	Min	Max	Pred	Succ
Unsorted Array	O(n)	O(1)	O(1)	O(1)	O(n)	O(n)	O(n)	O(n)
Sorted Array	O(log n)	O(n)	O(n)	O(1)	O(1)	O(1)	O(1)	O(1)
Unsorted List	O(n)	O(1)	O(1)	O(n)	O(n)	O(n)	O(n)	O(n)
Sorted List	O(n)	O(n)	O(1)	O(n)	O(1)	O(1)	O(1)	O(1)
Binary Search Tree	O(n)	O(n)	O(n)	-	O(n)	O(n)	O(n)	O(n)
AVL Tree	O(log n)	O(log n)	O(log n)	-	O(log n)	O(log n)	O(log n)	O(log n)
Hash Table (good hash!)	O(1)	O(1)	O(1)	-	O(n)	O(n)	O(n)	O(n)
Heap	O(n)	O(log n)	O(log n)	-	O(n)	O(1)	O(n)	O(n)

Priority Queue With Removal

Design a data structure that supports find_maximum, insert, and remove last inserted element as fast as possible.

Min/Max Priority Queue With Removal

Design a data structure that supports extract_maximum, extract_minimum, and insert as fast as possible.

Min/Max Priority Queue With Removal

Design a data structure that supports extract_maximum, extract_minimum, and insert as fast as possible.

How would the data structure be different if one of the operations was much less frequent than the other two?

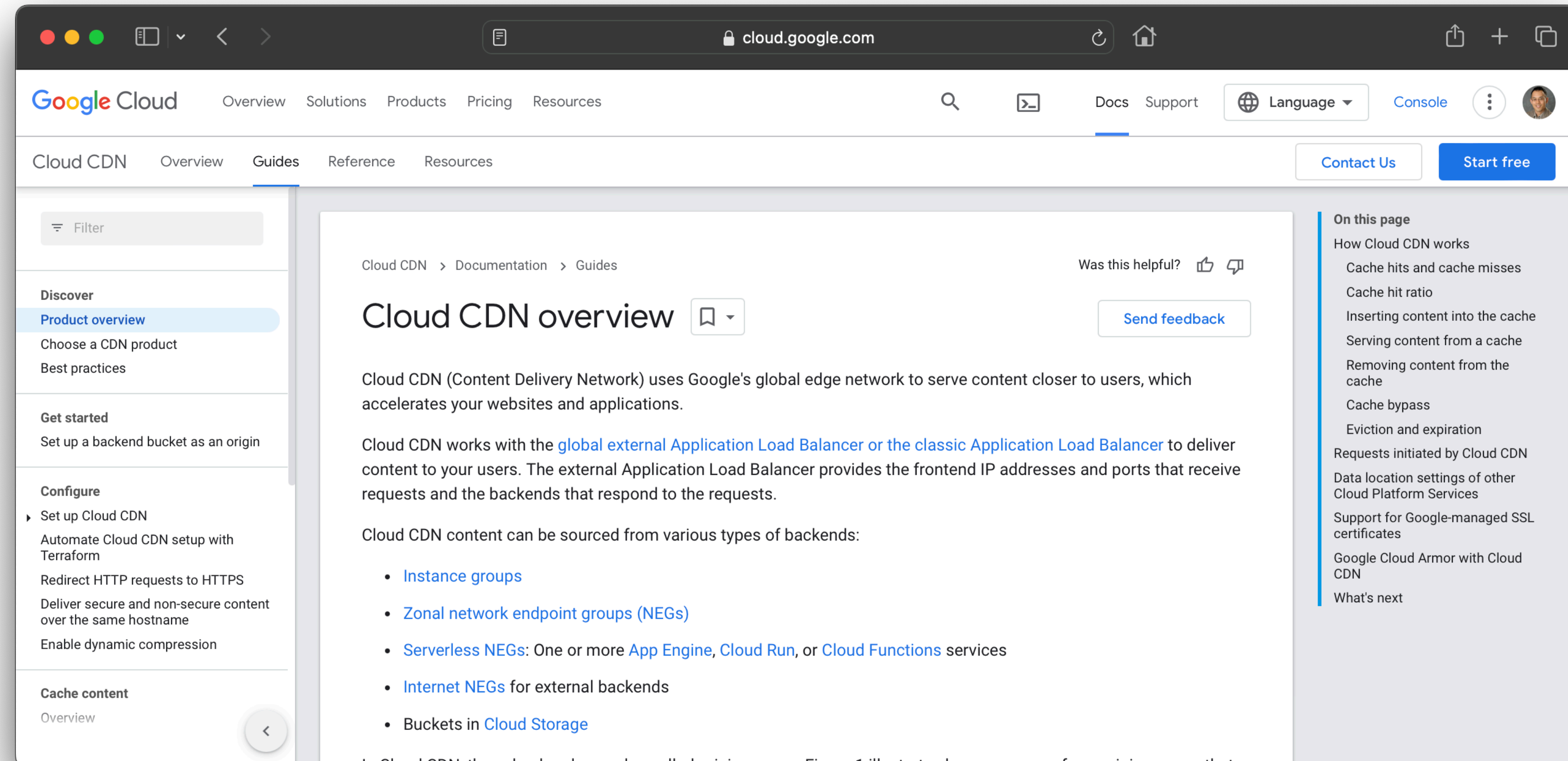
Weighted Random Numbers

Given a random() function that returns values uniformly distributed between 0 and 1, come up with an efficient algorithm that can randomly choose between outcomes that each have a specified weight.

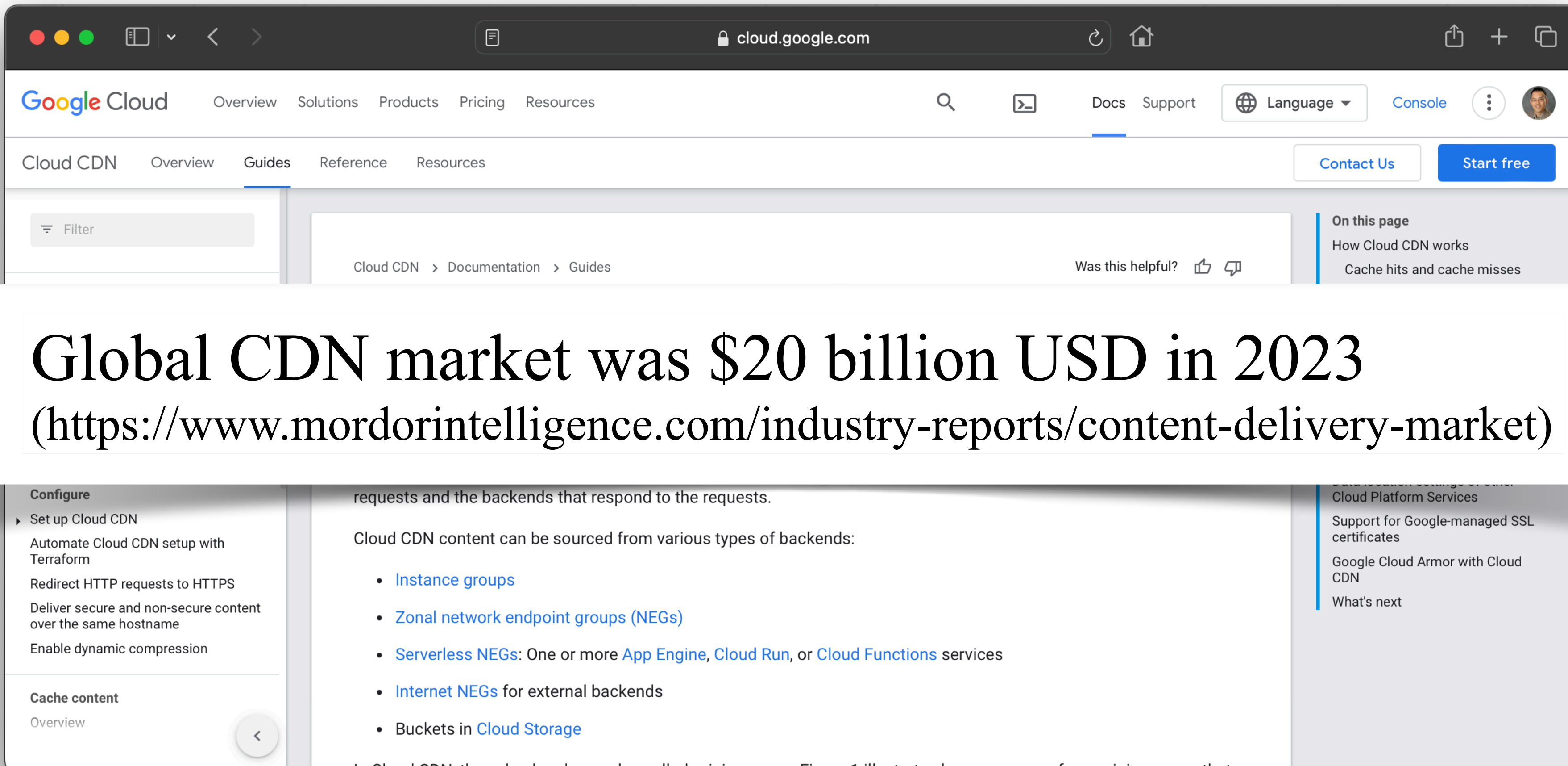
Example: A = 30%; B = 10%; C = 5%; …
Assume that there are many outcomes.

REAL-WORLD APPLICATION 1: WEB CACHES

Content Delivery Networks (CDNs) and Web Caching



Content Delivery Networks (CDNs) and Web Caching



Global CDN market was $20 billion USD in 2023

(https://www.mordorintelligence.com/industry-reports/content-delivery-market)

Web Cache Filtering

- URL requests in high-throughput and online manner
- Well over half of URLs are requested once only and never again requested
- We want to cache URLs (and URL content) that are accessed more than once
- The set of URLs is many orders of magnitude larger than we can store on our computing infrastructure, and growing all the time

Web Cache Filtering

- Propose an algorithmic solution that keeps track of all URL requests that have been accessed at least once before
- Memory usage is capped to a fixed size *b*
- We really don't want false negatives: check if a URL request is brand-new with 100% certainty
  - Note: this is by FAR the most common case
  - Recall discussion about abstracting away less important cases and focusing on the one/few cases that really matter
- But false positives are ok: occasionally caching some URLs that have been accessed once (but not two or more times) only adds a bit of overhead

Web Cache Filtering

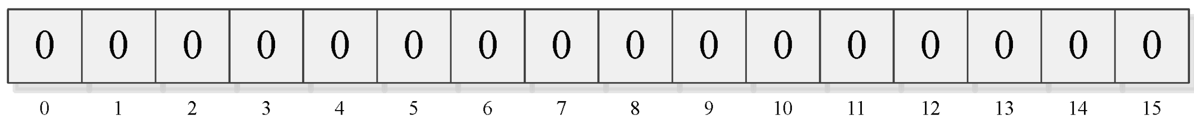
Bloom Filters

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

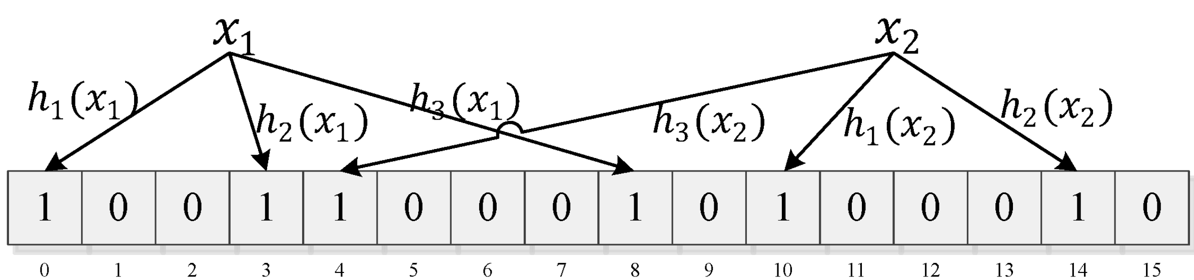
(a) A Bloom filter initialized with 0s

Reproduced from He et al. 2021

Bloom Filters



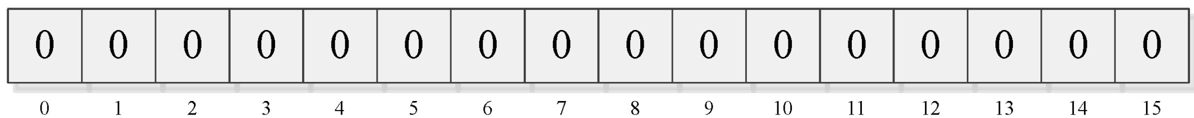
(a) A Bloom filter initialized with 0s



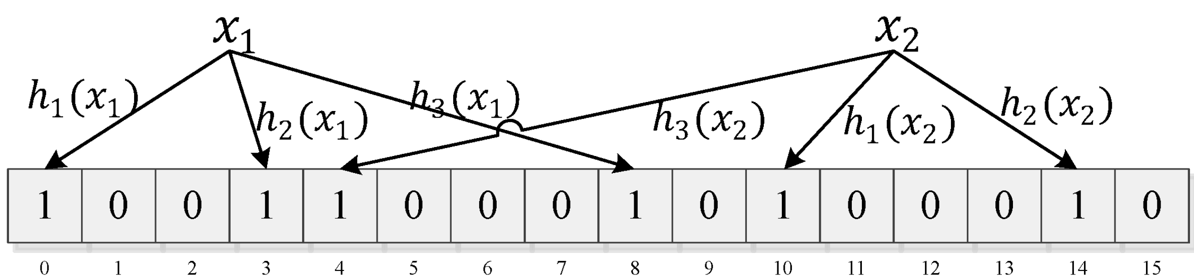
(b) Insert operation of Bloom filter

Reproduced from He et al. 2021

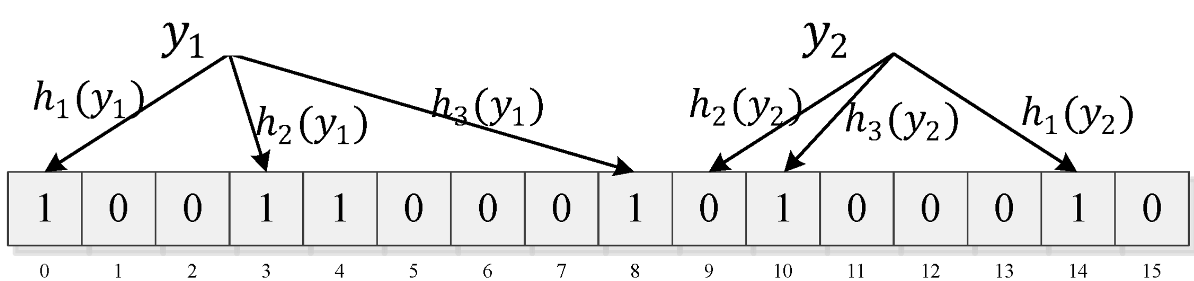
Bloom Filters



(a) A Bloom filter initialized with 0s



(b) Insert operation of Bloom filter

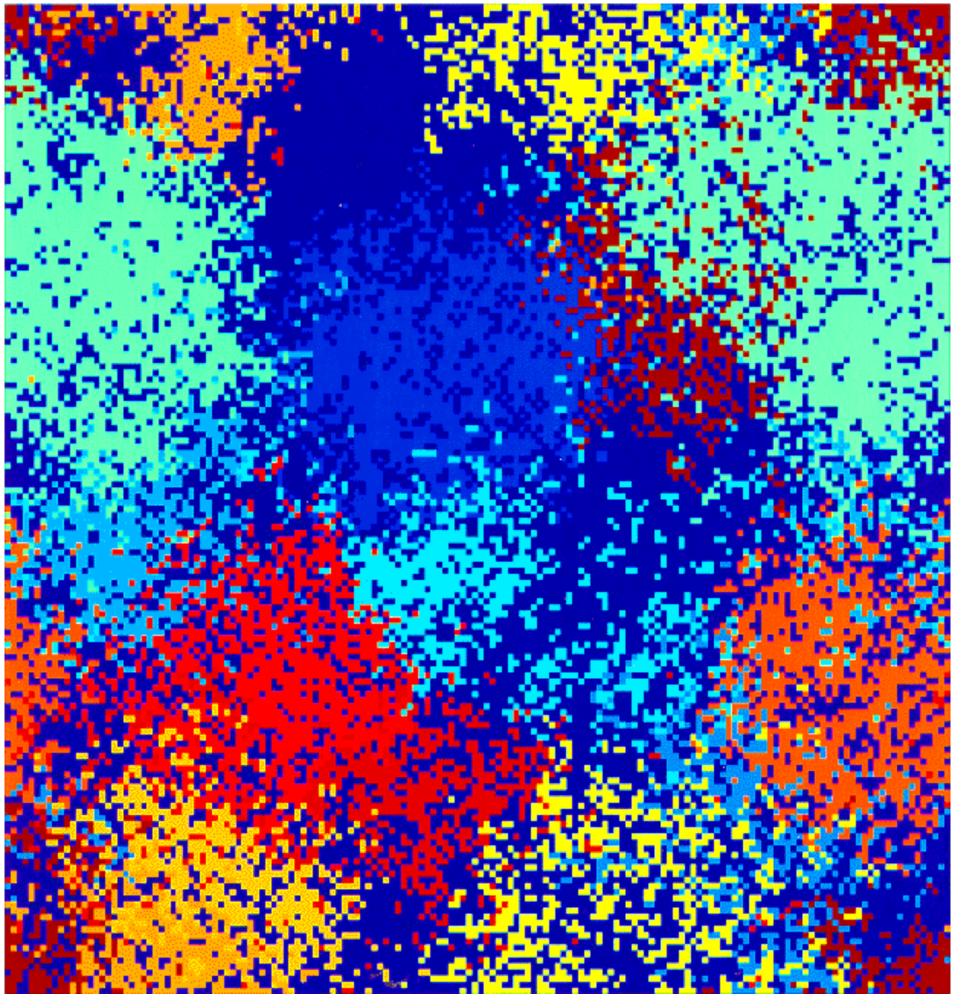


(c) Query operation of Bloom filter

Reproduced from He et al. 2021

REAL-WORLD APPLICATION 2: SCIENTIFIC SIMULATION SOFTWARE

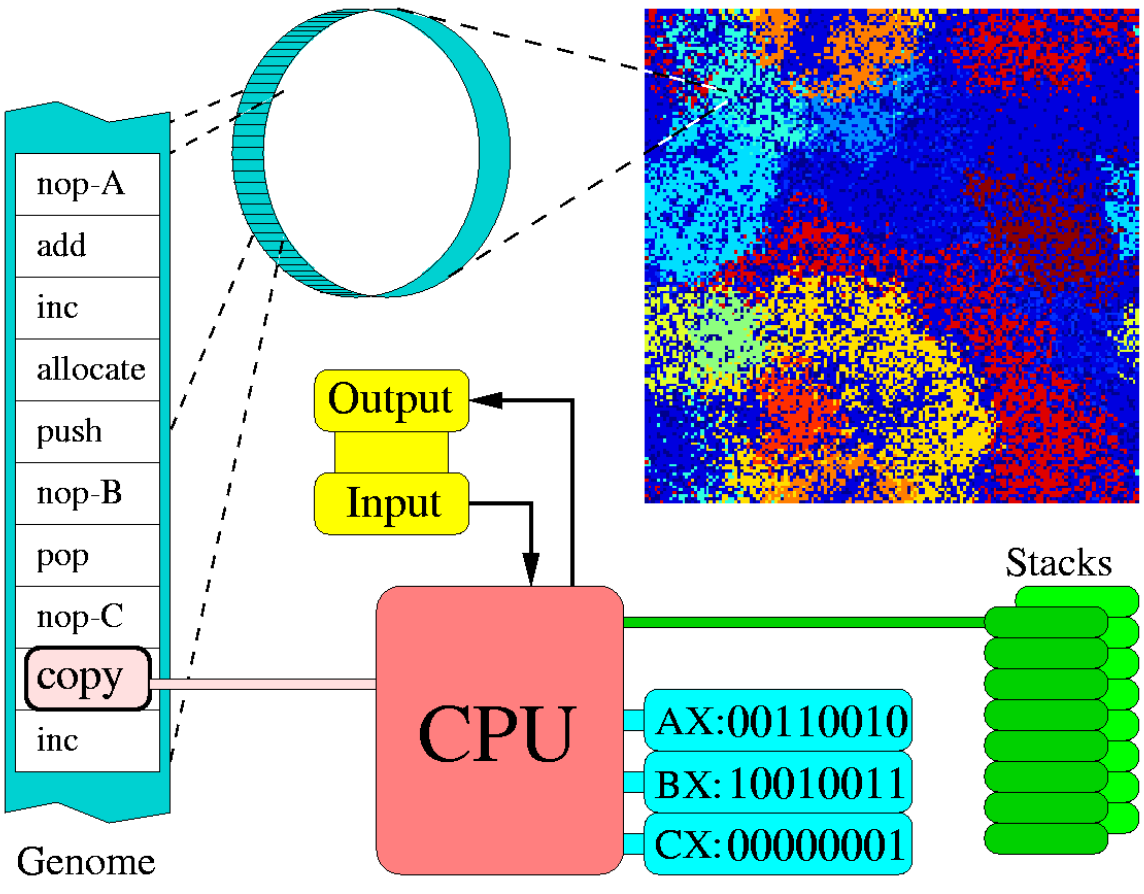
A Research Platform: Avida



Avida is a research platform for experimental studies of evolution and ecology with digital organisms.

Researchers can control most aspects of the system and record any measures of their choosing.

Physical World in Avida



Adaptation in Avida

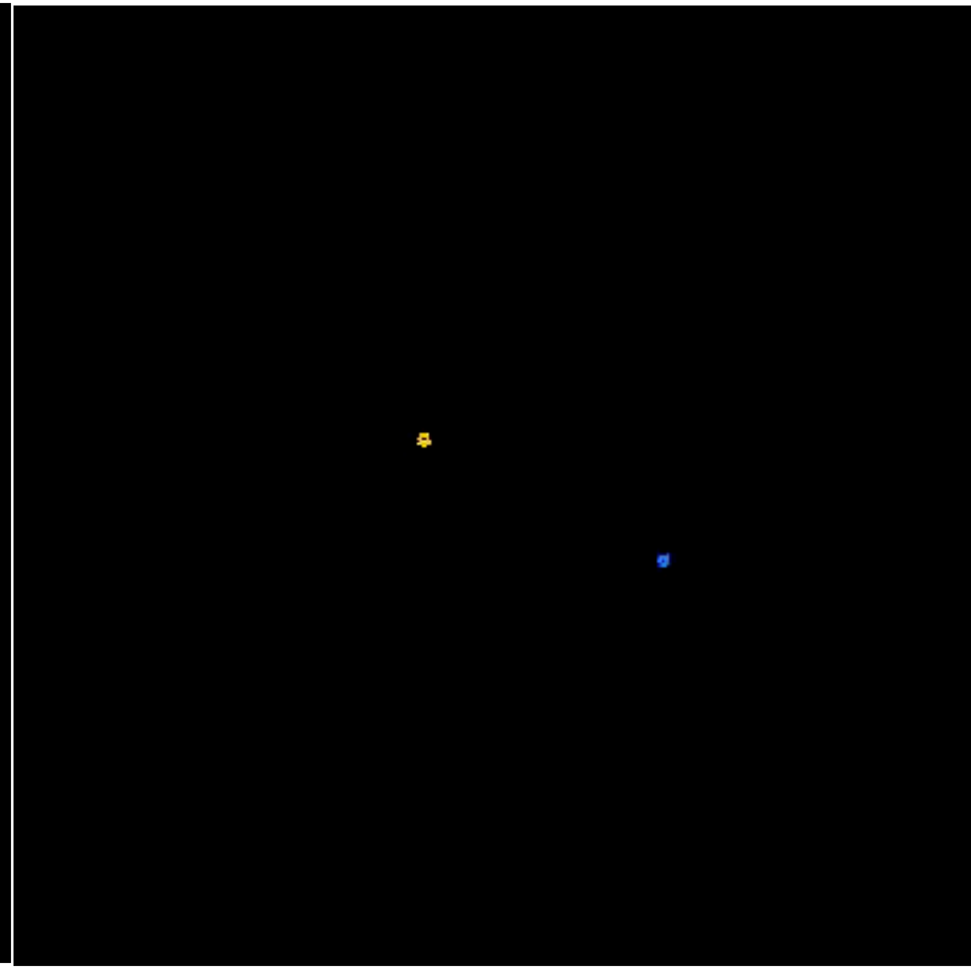
CPU cycles = Energy

Simple environments: all organisms receive the same amount of energy.

Complex environments: organisms compute math functions to metabolize a resource and gain energy.

Evolution *minimizes* time to produce offspring and adjusts priorities to *maximize* net energy absorbed.

Competition Experiments



The Situation:

We have n virtual CPUs each of which have a processing speed P_i. We want to each virtual CPU to be called with a probability proportional to its processing speed.

Example: $P_A = 2$, $P_B = 5$, $P_C = 3$

Sequence: B C A B B C B C A B B C A B B C B C A …

The Problem:

Given *n* objects, each of which have a processing speed, devise an algorithm that will produce a sequence in which the objects should be executed.

Analyze for speed of

> **Select**() – Pick a random CPU
>
> **AdjustSpeed**(ID, speed) – Change speed of a CPU

Assume that both of the functions we are analyzing will be run O(n) times. No other functions are needed.

Random returns a uniform random value in constant time.

Updated Problem:

What if we want to make sure that our output is as evenly distributed as possible? In other words:

Example: $P_A = 2, \quad P_B = 4$

GOOD Sequence: B B A B B A B B A B B A B B A B B A

POOR Sequence: A A B B B B A A B B B B A A B B B B

BAD Sequence: B B B A B B B B B B B B A A A B A A

Next time:
Everything You Ever Wanted to Know About Sorting (In 80 Minutes or Less!)