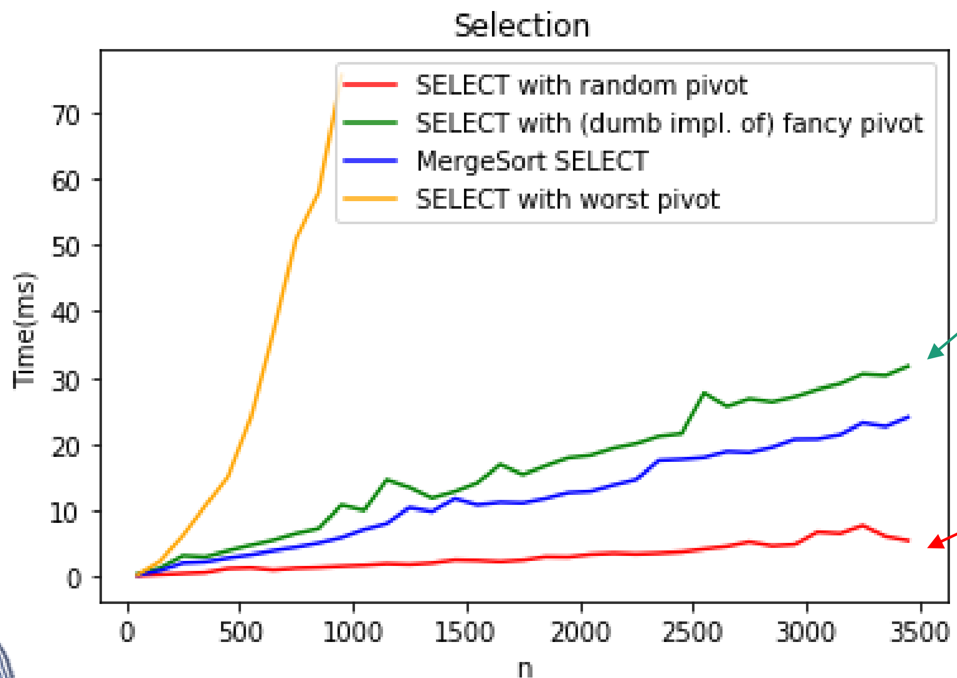
CSE100: Design and Analysis of Algorithms
Lecture 10 – Randomized Algorithms

Feb 17th 2022

Randomized Algorithms and QuickSort

A few Lectures ago…

- … in a galaxy far, far away!
- We saw a divide-and-conquer algorithm to solve the **`Select`** problem in time O(n) in the worst-case.
- It all came down to picking the pivot…



We choose a pivot **cleverly**

We choose a pivot **randomly.**

Randomized algorithms

- We make some random choices during the algorithm.
- We hope the algorithm works.
- We hope the algorithm is fast.

For today we will look at algorithms that always work and are probably fast.

e.g., **`Select`** with a random pivot is a randomized algorithm.

- Always works (aka, is correct).
- Probably fast.

Today

- How do we analyze randomized algorithms?
- A few randomized algorithms for sorting.
    - BogoSort
    - QuickSort

- BogoSort is a pedagogical tool.
- QuickSort is important to know. (in contrast with BogoSort…)

How do we measure the runtime of a randomized algorithm?

Scenario 1

1. You publish your algorithm.
2. Bad guy picks the input.
3. You run your randomized algorithm.

Scenario 2

1. You publish your algorithm.
2. Bad guy picks the input.
3. Bad guy chooses the randomness (fixes the dice) and runs your algorithm.

- In **Scenario 1**, the running time is a **random variable.**
  - It makes sense to talk about **expected running time**.
- In **Scenario 2**, the running time is **not random**.
  - We call this the **worst-case running time** of the randomized algorithm.

Today

- How do we analyze randomized algorithms?
- A few randomized algorithms for sorting.
  - BogoSort
  - QuickSort

- BogoSort is a pedagogical tool.
- QuickSort is important to know. (in contrast with BogoSort…)

Assume A has distinct entries

BogoSort

Suppose that you can draw a random integer in {1,…,n} in time O(1). How would you randomly permute an array in-place in time O(n)?

Ollie the over-achieving ostrich

- **BogoSort**(A)
  - **While** true:
    - Randomly permute A.
    - Check if A is sorted.
    - **If** A is sorted, **return** A.

- Let $X_i = \begin{cases} 1 \text{ if A is sorted after iteration i} \\ 0 \text{ otherwise} \end{cases}$
- $E[X_i] = \frac{1}{n!}$
- $E[\text{number of iterations until A is sorted}] = n!$