Sp18 CS 61B Discussion 13

Welcome!

Wayne Li

wli2@berkeley.edu

https://wayne-li2.github.io/

Announcements

- Project 3 is due 4/18!
- HW 5 released 4/20 and due 4/25

Quiz Instructions

- If you haven’t yet, please also **neatly** put your email address **outside the name box** if you want to be emailed!
- Bubble number **41**.

Forms

- Want to meet with me specifically?
  - https://goo.gl/forms/3HyKxt0ZAWHKRmij2
- Want to meet with **any** TA (not me)? @3258
- Survey:
  - https://goo.gl/forms/qxZ8qzS47HB62wiE3

Aside

External Merge-sort (CS 186)

- **Question**: How to sort 1 billion numbers?
  - Cannot fit in one machine's memory, but can fit on disk.
  - What's the algorithm?

External Merge-sort (CS 186)

- Split the input into **memory-sized** chunks.
- For each chunk, **quicksort** in memory, then **write to disk**.
- Okay, now we have **N** chunks of sorted numbers. What next?

External Merge-sort (CS 186)

- We do an **n-way** streaming merge-sort!
- Example: Memory can hold 4 blocks

External Merge-sort (CS 186)

- Output: [ ]
  [1, 4, 9, ...]   [0, 1, 1, ...]   [4, 5, 8, ...]

External Merge-sort (CS 186)

- Output: [ ]

  [1, 4, 9, ...] [**0**, 1, 1, ...] [4, 5, 8, ...]

External Merge-sort (CS 186)

- Output: [0]
  [1, 4, 9, ...]   [1, 1, ...]   [4, 5, 8, ...]

External Merge-sort (CS 186)

- Output: [0]
  [**1**, 4, 9, ...] [1, 1, ...] [4, 5, 8, ...]

External Merge-sort (CS 186)

- Output: [0, 1]
  [4, 9, ...]   [1, 1, ...]   [4, 5, 8, ...]

External Merge-sort (CS 186)

- Output: [0, 1]
  [4, 9, ...]   [**1**, 1, ...]   [4, 5, 8, ...]

External Merge-sort (CS 186)

- Output: [0, 1, 1]
  [4, 9, ...]   [1, ...]   [4, 5, 8, ...]

External Merge-sort (CS 186)

- Output: [0, 1, 1] -> Write out!
  [4, 9, ...]   [1, ...]   [4, 5, 8, ...]

External Merge-sort (CS 186)

- Output: [ ]
  [4, 9, ...] [1, ...] [4, 5, 8, ...]

External Merge-sort (CS 186)

- Output: [ ]

  [4, 9, …] [**1**, …] [4, 5, 8, …]

External Merge-sort (CS 186)

- Output: [1]
  [4, 9, ...]   [...]   [4, 5, 8, ...]

External Merge-sort (CS 186)

- Output: [1]

  [4, 9, ...]   [...]   [4, 5, 8, ...]

  Read in!

External Merge-sort (CS 186)

- Output: [1]

  [4, 9, ...]   [8, 9, 9, ...]   [4, 5, 8, ...]

  Read in!

External Merge-sort (CS 186)

- Output: [1]
  [**4**, 9, ...] [8, 9, 9, ...] [4, 5, 8, ...]

External Merge-sort (CS 186)

- Output: [1, 4]
  [9, ...]   [8, 9, 9, ...]   [4, 5, 8, ...]

External Sorting (CS 186)

- You’ll learn a more optimized version in CS 186!

External Sorting (CS 186)

- One quick optimization: Use N machines to quicksort each chunk!

External Sorting (CS 186)

- One quick optimization: Use N machines to quicksort each chunk!

Inversions

Definition of Inversion

- Given list A[1...N]:
    - For every i:
        - For every j such that i < j:
            - Exist inversion if A[i] > A[j]

Heapify

Quick Visualization

- Visualization
- Build heap bottom-up.
- At each sub-heap, find the smaller child, push up. Take the new child, and compare with its children.
- Runtime proof

Runtime Proof

$\boxed{3.4}$ *Extra*: Show that the running time of bottom-up heapify is $\Theta(n)$.

Some useful facts:

$$\sum_{i=0}^{\infty} x^i = \frac{1}{1-x}$$

Taking the derivative of both sides:

$$\sum_{i=0}^{\infty} ix^i = \frac{x}{(1-x)^2}$$

Running time of heapify is:

$$\begin{aligned}
\sum_{i=0}^{\log n} i\frac{n}{2^{i+1}} &= \frac{n}{2}\left(\sum_{i=0}^{\log n} i\left(\frac{1}{2}\right)^i\right) \\
&\leq \frac{n}{2}\left(\sum_{i=0}^{\infty} i\left(\frac{1}{2}\right)^i\right) \\
&= \frac{n}{2}\frac{\frac{1}{2}}{(\frac{1}{2})^2} \\
&= \Theta(n)
\end{aligned}$$

Essentially, the idea is just that each level roughly doubles the work, so the total runtime dependency on n is linear.

Extra: Is Heapsort Good?

- Is heapsort good?
  - Theta(N log N)!

Extra: Is Heapsort Good?

- In practice, it is bad for full sorts.
  - Quicksort is on average faster!
  - Unlike merge-sort, it is not stable!
  - Bad caching
  - **Cannot parallelize heapsort easily to multiple machines.**

Quicksort Optimization

Minimize Memory

- Hoare Partitioning: **in-place partition**
  - v.s. Creating three arrays, smaller, equal, and bigger.

Minimize Runtime

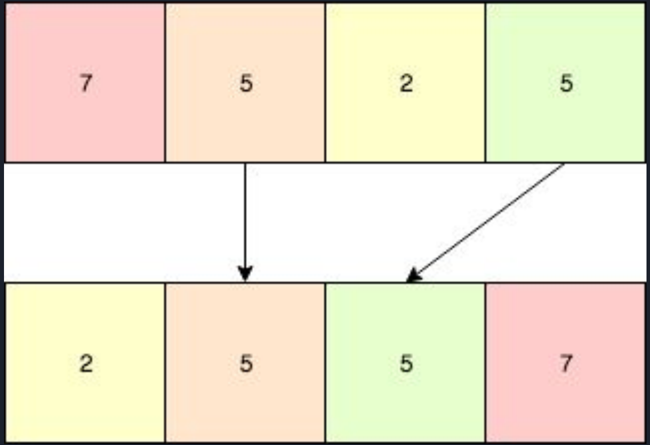
- Median pivots or approximate medians
  - Avoid worst case
  - **Quickselect**: Linear time median picking.
  - **Randomized approximate medians**: Simply pick k random pivots, and run an O(k) median selection (k << n)

Stability

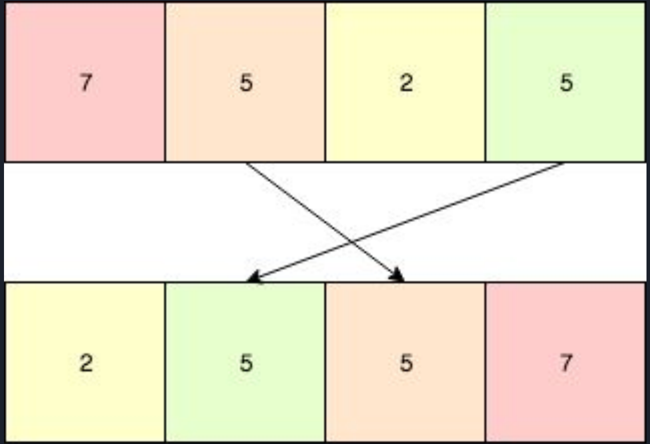
Sorting Stability

- Stable: Objects with the same value appear in the same order in the sorted output.
- We will see its power during radix sort!

Stable



Unstable



Onto Discussion