R3 Sort

3.22 https://github.com/GUMI-21/MIT6.006_note

Sorted Array

we can simply binary search to find keys and support Order operations!

Data Structure	Operations $O(\cdot)$				
	Container	Static	Dynamic	Order	
	`build(X)`	`find(k)`	`insert(x)` `delete(k)`	`find_min()` `find_max()`	`find_prev(k)` `find_next(k)`
Sorted Array	?	$\log n$	n	1	$\log n$

Sorting

- Selection sort maintains and grows a subset the largest i items in sorted order.
- Insertion sort maintains and grows a subset of the first i input items in sorted order.

Selection sort

Having already sorted the largest items into sub-array A[i+1:], the algorithm repeatedly scans the array for the largest item not yet sorted and swaps it with item A[i].
runtime: O(n^2)
`see code in r3.py`

Insertion Sort

Having already sorted sub-array A[:i], the algorithm repeatedly swaps item A[i] with the item to its left until the left item is no larger than A[i].
`see code in r3.py`

In-place and Stability

- in-place
  using at most a constant amount of additional space.
- stable
  Insertion sort is stable.
  meaning that items having the same value will appear in the sort in the same order as they appeared in the input array.

Merge Sort

o T(n) = Θ(n log n).
In particular, log n grows slower than any polynomial n^ε for ε > 0
`see code in r3.py`

Recurrences

- Substitution: Guess a solution and substitute to show the recurrence holds.
- Recursion Tree: Draw a tree representing the recurrence and sum computation at nodes. This is a very general method, and is the one we've used in lecture so far.
- Master Theorem: A general formula to solve a large class of recurrences. It is useful, but can also be hard to remember.

Master Theorem

case	solution	conditions
1	$T(n) = \Theta(n^{\log_b a})$	$f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$
2	$T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$	$f(n) = \Theta(n^{\log_b a} \log^k n)$ for some constant $k \geq 0$
3	$T(n) = \Theta(f(n))$	$f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$ and $af(n/b) < cf(n)$ for some constant $0 < c < 1$

case	solution	conditions	intuition
1	$T(n) = \Theta(n^{\log_b a})$	$c < \log_b a$	Work done at leaves dominates
2	$T(n) = \Theta(n^c \log n)$	$c = \log_b a$	Work balanced across the tree
3	$T(n) = \Theta(n^c)$	$c > \log_b a$	Work done at root dominates

Exercies