Quiz ch1-3
Ch6: Priority Queue
Ch7: Quick-sort

Quiz: Open book,
open paper notes.
No elec devices
(phone, tablet, laptop)

1 Oct 2013
CMPT231
Dr. Sean Ho
Trinity Western University

Exam 1: 30pts

- [6] (Dis)prove: If $f(n) \in O(g(n))$ and $g(n) \in O(h(n))$, then $h(n) \in \Omega(f(n))$
- [6] (Dis)prove: If $f(n) \in O(g(n))$ and $g(n) \in O(f(n))$, then $f(n) = g(n)$
- [6] (Dis)prove: $f(n) \in \Theta(f(n/2))$
- The function uniq(A) should return a list of all the elements in A which are unique: e.g.,
  - uniq([5, 3, 4, 3, 6, 5]) → [4, 6] (or [6, 4])
  - Elements may be arbitrarily large, or even floats
  - [8] Implement uniq() as efficiently as you can
  - [4] Derive the algorithmic complexity

Exam 1 solutions: #1-3

- [6] (Dis)prove: If $f(n) \in O(g(n))$ and $g(n) \in O(h(n))$, then $h(n) \in \Omega(f(n))$
  - True: transitivity ⇒ $f \in O(h)$
  - Transpose symmetry ⇒ $h \in \Omega(f)$
- [6] (Dis)prove: If $f(n) \in O(g(n))$ and $g(n) \in O(f(n))$, then $f(n) = g(n)$
  - False: e.g., $f(n) = n$, $g(n) = 2n$
- [6] (Dis)prove: $f(n) \in \Theta(f(n/2))$
  - False: e.g., $f(n) = 2^n$:
    - $\lim_{n\to\infty}(f(n)/f(n/2)) = \lim_{n\to\infty}(2^n / 2^{n/2}) = \lim_{n\to\infty}(2^{n/2}) = \infty$
    - Hence $f \in \omega(f(n/2))$, so $f \notin \Theta(f(n/2))$

Exam 1 solutions: #4

- [8] Implement uniq() as efficiently as you can
  - function uniq(A):
    - MergeSort(A)
    - result = [A[1]]
    - for i in 2 .. length(A):
      - if (A[i] != A[i-1]) result.append(A[i])
    - return result
- [4] Derive the algorithmic complexity
  - MergeSort takes average $\Theta(n \lg n)$
  - Linear scan for uniques takes $\Theta(n)$
  - $\Rightarrow$ average $\Theta(n \lg n)$

Outline for today

- ch6: Binary max-heaps
  - Application: Priority Queue
- ch7: Quicksort
  - Partition & pivot
  - Randomised quicksort
  - Complexity analysis

Binary heap for priority queue

- Binary heaps can implement a priority queue:
  - Set of items with attached priorities
- Interface (set of operations):
  - insert(A, item, pri): add item to the queue A
  - find_max(A): return item with highest priority
  - pop_max(A): same but also delete item
  - set_pri(A, item, pri): set new priority for item (must be higher than old priority)
- Setup queue by building a max-heap
  - find_max() is easy: return A[1]
  - pop_max() also easy: remove A[1] and heapify

Inserting into priority queue

- set_pri(A, i, pri): starting from i, "bubble" item up until we find the right place:
  - A[i] = pri
  - while i>1 and A[i/2] < A[i]:
    - swap(A[i/2], A[i])
    - i = i/2
  - Complexity: # iterations = Θ(lg n)
- insert(A, pri): make a new node and set its priority
  - A.length++
  - set_pri(A, A.length, pri)
    - Typically, use pre-allocated fixed-length array, and use separate variable to track size of queue
  - Complexity: same as set_pri(): Θ(lg n)

Priority queue: summary

- Build priority queue using a max-heap: $\Theta(n)$
- Get highest priority item: $\Theta(1)$
- Get and delete highest priority item: $\Theta(\lg n)$
- Set new priority for an item: $\Theta(\lg n)$
- Insert new item into queue: $\Theta(\lg n)$

Outline for today

- ch6: Binary max-heaps
  - Application: Priority Queue
- ch7: Quicksort
  - Partition & pivot
  - Randomised quicksort
  - Complexity analysis

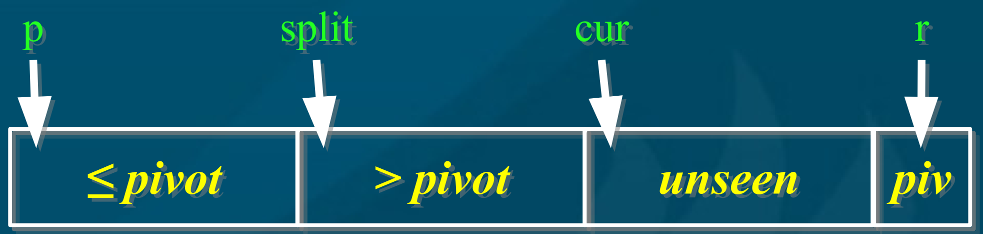
Quicksort

- Divide: partition array A[p .. r] such that:
  - max(A[p .. q-1]) ≤ A[q] ≤ min(A[q+1 .. r])
- Conquer: recurse on each part:
  - quicksort(A, p, q-1) and quicksort(A, q+1, r)
- No combine/merge step needed

- In-place sort
- Worst-case turns out to still be $\Theta(n^2)$, but average-case is $\Theta(n \lg(n))$, with small constants
- In practise, quicksort is one of the best algorithms when input values can be arbitrary

Quicksort: partition

- How to do the partitioning?
  - Pick last item as the pivot
  - Walk through array, partitioning array into items ≤ pivot and items > pivot
  - Lastly, swap pivot into place
    - partition(A, p, r):
      - pivot = A[r]
      - split = p
      - for cur = p .. r-1:
        - if A[cur] ≤ pivot:
          - swap(A[split], A[cur])
          - split++
      - swap(A[split], A[pivot])
      - return split



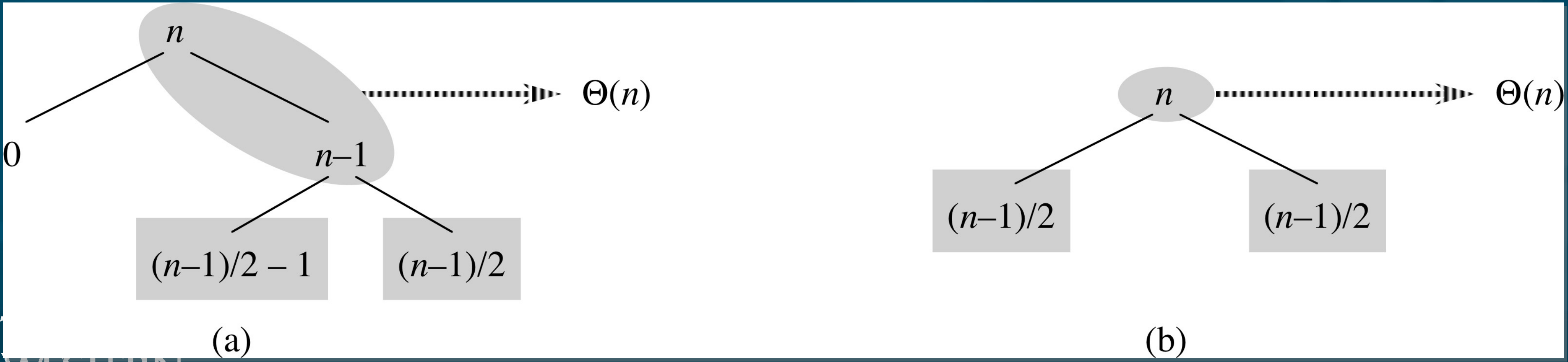
Complexity?

Quicksort: complexity

- Worst-case if every partition is the most uneven:
    - pivot (last item) is either largest or smallest item
    - $T(n) = T(n-1) + T(0) + \Theta(n)$
    - $\Rightarrow T(n) = \Theta(n^2)$
  - Example inputs that give worst case?
- Best-case if every partition is exactly in half:
    - $T(n) = 2T(n/2) + \Theta(n)$
    - $\Rightarrow T(n) = \Theta(n \lg(n))$
  - Example inputs that give best case?
- Average-case, assuming random input?

Quicksort: average case

- Not every partition will be best-case ½ – ½
  - On average, in between best and worst cases
  - Even if average split is, say, 9/10 – 1/10:
    - $T(n) = T((9/10)n) + T((1/10)n) + \Theta(n)$
    - $\Rightarrow T(n) = O(n \lg(n))$
- E.g., assume splits alternate between best+worst:
  - Only adds $O(n)$ work to each of $O(\lg n)$ levels
  - $\Rightarrow$ still $O(n \lg(n))$ (albeit w/higher constant)



Quicksort with constant splits

- p.178, #7.2-5: assume every split is α vs $1-\alpha$, with constant $0 < \alpha < ½$.
  - Min/max depth of a leaf in the recursion tree?
- Min depth: follow smaller side (α) of each split
  - How many splits until reach leaf (1 item)?
    - $\alpha^m n = 1 \implies m = -\lg(n) / \lg(\alpha)$
- Max depth: follow larger side ($1-\alpha$) of each split
  - How many splits until reach leaf (1 item)?
    - $(1-\alpha)^m n = 1 \implies m = -\lg(n) / \lg(1-\alpha)$
- Both are $\Theta(\lg n)$, so with constant-ratio splits, depth of recursion tree is $\Theta(\lg n)$, ⇒ total complexity is $\Theta(n \lg n)$

Outline for today

- ch6: Binary max-heaps
  - Application: Priority Queue
- ch7: Quicksort
  - Partition & pivot
  - Randomised quicksort
  - Complexity analysis

Randomised quicksort

- We saw how giving quicksort pre-sorted data results in worst-case behaviour
  - Always chose last element (r) as pivot
- We can alleviate this risk by randomising our choice of pivot:
  - rand_partition(A, p, r):
    - swap(A[r], A[rand(p, r)]) *# swap w/random item*
    - partition(A, p, r)
  - It is still possible our random pivot choices result in worst-case $\Theta(n^2)$ time – but unlikely!

Randomised quicksort: average

- Assume items are distinct, and name them in order: $\{z_1, z_2, \ldots, z_n\}$. How many comparisons?
  - Worst case: all pairs (z_i, z_j) compared $\Longrightarrow \Theta(n^2)$
  - A pair cannot be compared >1 time, because comparisons are only made against pivots, and once a pivot is used by partition(), it is not revisited
- When is a pair (z_i, z_j) compared?
  - Only if either z_i or z_j are chosen as a pivot before any other item inbetween $\{z_i, z_{i+1}, \ldots, z_j\}$
    - (If any other item is chosen first, then z_i, z_j will be on opposite sides of the split, and will not be compared)
  - $\Rightarrow$ probability is $2(1 / (j - i + 1))$

Randomised quicksort: average

- Summing over all pairs (z_i, z_j):

$$
\begin{aligned}
& \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} Pr\left(compare\ z_i\ with\ z_j\right) \\
= & \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \\
= & \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} \quad \left(let\ k=j-i\right) \\
< & \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k} \\
= & \sum_{i=1}^{n-1} O\left(\lg n\right) \qquad \textit{(e.g., by Riemann sums)} \\
= & O\left(n \lg n\right)
\end{aligned}
$$

Visualisations of Sorting algos

- **The Sound of Sorting** - Visualization and "Audibilization" of Sorting Algorithms