Transform and Conquer

Solve problem by transforming into:

- a more convenient instance of the same problem (*instance simplification*)
  - presorting
  - Gaussian elimination

- a different representation of the same instance (*representation change*)
  - balanced search trees
  - heaps and heapsort
  - polynomial evaluation by Horner’s rule
  - Fast Fourier Transform

- a different problem altogether (*problem reduction*)
  - reductions to graph problems
  - linear programming

Reading: Chapter 5, Sections 5.5, 5.6;
Chapter 6, Sections 6.1, 6.3

Exercises:

- 5.5: 2, 3
- 5.6: 2, 3, 7, 8, 10
- 6.1: 1, 3, 5, 6
- 6.3: 1, 2, 3, 4, 5

Next time: Heapsort (6.4); begin space-time tradeoffs
Instance simplification - Presorting

Solve instance of problem by transforming into another simpler/easier instance of the same problem

Presorting:
Many problems involving lists are easier when list is sorted.
- searching
- computing the median (selection problem)
- computing the mode
- finding repeated elements

Selection Problem
Find the $k^{th}$ smallest element in $A[1], \ldots, A[n]$. Special cases:
- $minimum$: $k = 1$
- $maximum$: $k = n$
- $median$: $k = \lceil n/2 \rceil$

- Presorting-based algorithm
  - sort list
  - return $A[k]$

- Partition-based algorithm (Variable decrease & conquer):
  - pivot/split at $A[s]$ using partitioning algorithm from quicksort
  - if $s = k$ return $A[s]$
  - else if $s < k$ repeat with sublist $A[s+1], \ldots, A[n]$
  - else if $s > k$ repeat with sublist $A[1], \ldots, A[s-1]$.
Notes on Selection Problem

- Presorting-based algorithm: $\Omega(n \log n) + \Theta(1) = \Omega(n \log n)$

- Partition-based algorithm (Variable decrease & conquer):
  - worst case: $T(n) = T(n-1) + (n+1) \Rightarrow \Theta(n^2)$
  - best case: $\Theta(1)$
  - average case: $T(n) = T(n/2) + (n+1) \Rightarrow \Theta(n)$
  - Bonus: also identifies the $k$ smallest elements (not just the $k^{th}$)

- Special cases max, min: better, simpler linear algorithm (brute force)

- Conclusion: Presorting does not help in this case.

Finding repeated elements

- Presorting-based algorithm:
  - use mergesort (optimal): $\Theta(n \log n)$
  - scan array to find repeated adjacent elements: $\Theta(n)$
  \[ \Rightarrow \Theta(n \log n) \]

- Brute force algorithm: $\Theta(n^2)$

- Conclusion: Presorting yields significant improvement

- Similar improvement for mode

- What about searching?
Taxonomy of Searching Algorithms

γ Elementary searching algorithms
- sequential search
- binary search
- binary tree search

γ Balanced tree searching
- AVL trees
- red-black trees
- multiway balanced trees (2-3 trees, 2-3-4 trees, B trees)

γ Hashing
- separate chaining
- open addressing

Binary search trees

γ Arrange keys in a binary tree with the binary search tree property:

Example 1: 5, 10, 3, 1, 7, 12, 9

Example 2: 4, 5, 7, 2, 1, 3, 6

• What about repeated keys?
Searching, insertion, and deletion

- Searching must be considered in the context of:
  - file size (internal vs. external)
  - dynamics of data (static vs. dynamic)

- Dictionary operations:
  - find
  - insert
  - delete

Searching, insertion, and deletion in binary search trees

- Searching – straightforward
- Insertion – search for key, insert at leaf where search terminated
- Deletion – 3 cases:
  - deleting key at a leaf
  - deleting key at node with single child
  - deleting key at node with two children

- All operations: worst case # key comparisons = \( h + 1 \)
- \( \lfloor \log n \rfloor \leq h \leq n - 1 \) with average (random files) 1.41 \( \log n \)
- Thus all operations have:
  - worst case: \( \Theta(n) \)
  - average case: \( \Theta(\log n) \)
- **Bonus:** inorder traversal produces sorted list (treesort)
Balance trees: AVL trees

- For every node, difference in height between left and right subtree is at most 1
- AVL property is maintained through rotations, each time the tree becomes unbalanced

\[ \lg n \leq h \leq 1.4404 \lg (n + 2) - 1.3277 \]

average: \( 1.01 \lg n + 0.1 \) for large \( n \)

- Disadvantage: needs extra storage for maintaining node balance
- A similar idea: red-black trees (height of subtrees is allowed to differ by up to a factor of 2)

AVL tree rotations

- Small examples:
  - 1, 2, 3
  - 3, 2, 1
  - 1, 3, 2
  - 3, 1, 2

- Larger example: 4, 5, 7, 2, 1, 3, 6

- See figures 6.4, 6.5 for general cases of rotations;

- Algorithm maintains balance factor for each node - see figure 6.3