Algorithms Crash Review

Laura Toma, csci2200, Bowdoin College

Here is a list of fundamental topics studied in algorithms:

1. Worst case, best case running times.
   - Binary search runs in logarithmic time worst case and constant time best case.
2. Asymptotic growth of functions (O, Ω, Θ). Used to talk about complexity of algorithms.
   - This algorithm runs in $O(n^2)$ worst case.
   - This algorithm runs in $\Theta(n \lg n)$ worst case.
3. Recurrences and summations. Used to analyze complexity of algorithms and in particular recursive algorithms.
   - Example: mergesort recurrence: $T(n) = 2T(n/2) + \Theta(n)$ solves to $\Theta(n \lg n)$.
4. Comparison-based Sorting
   - Quadratic sorts: insertion sort, bubble sort, selection sort
   - Mergesort
   - Quicksort, Randomized quicksort
   - Heapsort
   - Comparison-based sorting lower bound: Any sorting algorithm that uses only comparisons to order the elements must take $\Omega(n \lg n)$ in the worst case.
5. Linear-time sorting
   - Not a general purpose sort; need additional assumptions, usually that the keys to be sorted are integers in a small range.
   - Example: sort n keys that are all integers in the range $\{-10, ..., 10\}$.
   - Counting sort, Radix sort, Bucket sort
   - Generally run in $O(n + k)$ where k is the range
6. Selection: given a set of n elements, find the ith smallest.
   - Via sorting, in $O(n \lg n)$ time.
   - Direct algorithm in $O(n)$ time
7. Data structures:

- Priority queue: supports FIND-MIN, DELETE-MIN, INSERT, DELETE, CHANGE-KEY. Max priority queue, min priority queue.
- Dictionary: supports INSERT, DELETE, SEARCH
- Union-Find: supports MAKE-SET, FIND-SET, UNION-SET

8. Data structure implementations
   - Heap as priority queue
   - Balanced binary search trees for dictionaries.
9. Binary search trees
   - binary search tree property
   - tree walks
   - supports INSERT, DELETE, SEARCH, MIN, MAX, PRED, SUCC in $O(h)$ time
10. Red-black trees
    - height is manintained $O(\lg n)$ via rotations
    - all operations run in $O(\lg n)$ time
11. Dynamic programming
    - technique used for solving optimization problems that have optimal substructure, and overlapping subproblems
    - approach: express the problem recursively; store (cache) partial solutions in a table.
    - Examples: 0-1 knapsack, rod cutting, longest common subsequence, maximum sub-array, weighted interval scheduling, subset sum, weighted subset sum (0-1 knapsack), sequence alignment, longest common subsequence; also shortest paths (Bellman Ford)
12. Greedy algorithms
    - Solves optimization problems by "quickly" finding the choice that looks best "locally", taking it, and recursing on what's left. The best choice is determined by examing only local information, without going into recursion to see how good the choice is "globally".
    - Classical examples: fractional knapsack, interval scheduling; also Dijkstra SSSP, Prim and Kruskal's algorithm for MST are all greedy.
    - Greedy heuristics are used a lot in AI and are good starting points for optimization problems.
13. Graph algorithms
    - Graph representation (adjacency list, adjacency matrix)
    - Graph traversal: BFS
      - G can be directed or undirected
      - BFS runs in linear time $O(V + E)$

  - BFS used to: find connected components (G undirected), check bipartiteness (G undirected), compute shortest paths (G un-weighted, all edges have weight 1)
- Graph traversal: DFS
  - G can be directed or undirected
  - DFS runs in linear time $O(V + E)$
  - DFS used for: find connected components, reachability, find cycles (back edges), topological sort (G directed, acyclic)

- Acyclic directed graphs (DAGs): topological ordering
  - Ordering of the vertices such that all edges are "forward".
  - Exists if and only iff the graph has no cycles
  - Can be computed in linear time $O(V + E)$ eitehr directly or via DFS
  - Used for dynamic programming on DAGs.
  - Many problems have easier/faster solution on DAGs and tehse solutions exploit that the DAG has a topological order. Example: SSSP on DAGs in linear time; Longest paths on DAGs in linear time (note: longest path is an NPC problem).

- Minimum spanning tree (MST)
  - G: connected, undirected, weighted
  - MST can be computed in $O(E \lg V)$ time with Prim's or Kruskal's algorithms
  - Union-find data structure: supports Find and Join/Union operations.

- Single source shortest paths (SSSP): Find shortest paths from a vertex to all other vertices
  - G: directed, weighted
  - On directed unweighted graphs SSSP can be computed by BFS in $O(E + V)$ time
  - On DAGs, SSSP can be computed in $O(E + V)$ time
  - Dijkstra: Runs in $O(E \lg V)$ time; Works correctly only if G has non-negative weights
  - If the graph has negative edge weights: Bellman-Ford's algorithm runs in $O(EV)$ and can also detect negative-weight cycles

14. Complexity:

- P: problems that can be solved in polynomial time (on a deterministic Turing machine)
- NP: problems that can be verified in polynomial time
- NPC: a problem is in NPC if it is in NP and all problems in NP reduce to it in polynomial time. Intuitively, NPC is the core of hard problems in NP.
- Some NPC problems: SAT (satisfiability: is there an assignment that makes a given formula true), traveling salesman (TSP: find minimum-cost tour), longest path (find the longest simple path in a graph); find the largest clique subgraph (a clique is a complete graph).
- All NPC problems reduce to each other therefore if any of them is shown to be in P, it follows they all are and $P = NP$
- The \$1M questions: Is $P = NP$? Not known. Most people believe the answer is NO.