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 complecity 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
    - classical 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)
    - approach: express the problem recursively; store (cache) partial solutions in a table.
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
  - Thinking about MST we talked about a general union-find data structure, that 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 directed acyclic graphs, SSSP can be computed in $O(E + V)$ time with dynamic programming
  - Dijkstra's SSSP algorithm
    - Runs in $O(E \lg V)$ time
    - Works correctly only if G has non-negative weights
  - If graphs has negative edge weights, SSSP can be computed but takes longer than Dijkstra's algorithm (Bellman-Ford's algorithm, $O(EV)$)

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 speaking, NPC is the hard core of 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).
- The $1M questions: Is $P = NP$? Not known. Most people believe the answer is NO.

Interview tips

The following tips were taken from: *Hacking a Google InterviewMIT, Instructors: Bill Jacobs and Curtis Fonger http://courses.csail.mit.edu/iap/interview.*

When asked a question, open a dialog with the interviewer. Let them know what you are thinking. You might, for example, suggest a slow or partial solution (let them know that the solution is not ideal), mention some observations about the problem, or say any ideas you have that might lead to a solution. Often, interviewers will give hints if you appear to be stuck.

Often, you will be asked to write a program during an interview. For some reason, interviewers usually have people write programs on a blackboard or on a sheet of paper rather than on a computer. It is good to get practice with writing code on the board in order to be prepared for this.

Here is a list of "do's" and "don't's" when doing a programming interview:

Do's:

- Ask for clarification on a problem if you didn't understand something or if there is any ambiguity
- Let the interviewer know what you are thinking
- Suggest multiple approaches to the problem
- Bounce ideas off the interviewer (such as ideas for data structures or algorithms)
- If you get stuck, don't be afraid to let them know and politely ask for a hint

Don't's:

- Never give up! This says nothing good about your problem solving skills.
- Don't just sit in silence while thinking. The interviewer has limited time to find out as much as possible about you, and not talking with them tells them nothing, except that you can sit there silently.
- If you already know the answer, don't just blurt it out! They will suspect that you already knew the answer and didn't tell them you've seen the question before. At least pretend to be thinking though the problem before you give the answer!

Sample interview questions

Taken from Hacking a Google Interview Practice Questions MIT
`http://courses.csail.mit.edu/iap/interview/`.

1. Question: Searching through an array.

   Given a sorted array of integers, how can you find the location of a particular integer x?

2. Question: Target Sum.

   Given an integer x and an unsorted array of integers, describe an algorithm to determine whether two of the numbers add up to x. (In this case, say that the interviewer hates hash tables.)

3. Question: Nearest Neighbor. Say you have an array containing information regarding n people. Each person is described using a string (their name) and a number (their position along a number line). Each person has three friends, which are the three people whose number is nearest their own. Describe an algorithm to identify each person's three friends.

4. Question: Given an array of integer, find the number of un-ordered pairs in that array, say given 1, 3, 2, the answer is 1 because 3, 2 is un-ordered, and for array 3, 2, 1, the answer is 3 because 3, 2, 3, 1, 2, 1.

5. Question: Obstacle Avoidance.

   Given an n x n grid with a person and obstacles, how would you find a path for the person to a particular destination? The person is permitted to move left, right, up, and down.

6. Question: Axis Aligned Rectangles

   Describe an algorithm that takes an unsorted array of axis aligned rectangles and returns any pair of rectangles that overlaps, if there is such a pair.

   Axis aligned means that all the rectangle sides are either parallel or perpendicular to the x and yaxis.

   You can assume that each rectangle object has two variables in it: the xy coordinates of the upperleft corner and the bottomright corner.

More interview problems

Some recent interview questions from Bowdoin students:

1. (2016) Given a NxN matrix count the number of islands.. A 1 denotes part of island 0 denotes water (blob search).

2. (2016) Max contiguous subarry sum: return the largest sum of a contiguous sub-array in a given array

   Given [2 3 4 4 -4 5], should return 2+3+4+4 = 13

   Given [4 -4 5 6 -2 8 -10], should return 5+6+ -2 + 8 = 17

3. (2016) Given sorted array with pairs and a single unique value, i.e. [1, 1, 2, 2, 3, 4, 4, 5, 5], return the value of the unique int.

4. (2016) Given a String remove all duplicate letters

5. Given a binary tree, write a method to verify if the tree is a binary search tree in linear time in linear time and constant space.

6. Write a function to find the kth biggest number in an unsorted array in linear time (expected).

7. Given an unsorted array and a number K. Find a subset of the array whose sum is K.

   Note: Normally the interviewer would start off asking about the 2-Sum problem, then 3-Sum and then they'd ask to generalize to find any subset. Again, it emphasizes how important the thought process is.

8. Given k sorted arrays, write a method to merge the into one sorted array. (note: often used as an intermediate step for harder problems)

9. Write a routine in pseudocode to reverse the bits in a word passed as input (using bitwise operations!).

10. You are in charge of planning a party for a company. The company has a hierarchical structure, which forms a tree rooted at the president. Each person has a supervisor, and possibly some people whom they supervise. Assume everyone is listed at the highest possible position in the tree and there are no double affiliations (everybody has only one supervisor). The company also has a (secret?) database which ranks each person with a conviviality rating (a real number, which can be negative if the person is really difficult or boring). In order to make the party fun for everybody, the president does not want both a person and his/her immediate "supervisor" to attend.

    You are given a tree that describes the strucure of a company. Each node also holds a name and a conviviality ranking. Describe an algorithm to make up a guest list that maximizes the sum of the conviviality rankings of the guests. Analyze the running time of your algorithm.

ANSWERS

1. Question: Searching through an array. Given a sorted array of integers, how can you find the location of a particular integer x?

   Good answer: Use binary search. Compare the number in the middle of the array with x. If it is equal, we are done. If the number is greater, we know to look in the second half of the array. If it is smaller, we know to look in the first half. We can repeat the search on the appropriate half of the array by comparing the middle element of that array with x, once again narrowing our search by a factor of 2. We repeat this process until we find x. This algorithm takes O(log n) time.

   Notsogood answer: Go through each number in order and compare it to x. This algorithm takes O(n) time.

2. Question: Target Sum. Given an integer x and an unsorted array of integers, describe an algorithm to determine whether two of the numbers add up to x. (In this case, say that the interviewer hates hash tables.)

   Good Answer: Sort the array. For each number a in the array, binary searchfor $x - a$. This solution takes O(n log n) time because we sort the numbers and then perform n binary searches.

3. Question: Nearest Neighbor. Say you have an array containing information regarding n people. Each person is described using a string (their name) and a number (their position along a number line). Each person has three friends, which are the three people whose number is nearest their own. Describe an algorithm to identify each person's three friends.

   Good answer: Sort the array in ascending order of the people's number. For each person, check the three people immediately before and after them. Their three friends will be among these six people. This algorithm takes O(n log n) time, since sorting the people takes that much time.

4. Given an array of integer, find the number of un-ordered pairs in that array, say given 1, 3, 2, the answer is 1 because 3, 2 is un-ordered, and for array 3, 2, 1, the answer is 3 because 3, 2, 3, 1, 2, 1.

   Good answer: Obviously, this can be solved by brute force with $O(n^2)$ running time, or permute all possible pairs then eliminate those invalid pairs. BVut can be done better in $O(n \lg n)$ via divide and conquer.

5. Question: Obstacle Avoidance. Given an n x n grid with a person and obstacles, how would you find a path for the person to a particular destination? The person is permitted to move left, right, up, and down.

Good Answer: Use BFS, or any shortest path finding algorithm.

6. Question: Axis Aligned Rectangles

   Describe an algorithm that takes an unsorted array of axis aligned rectangles and returns any pair of rectangles that overlaps, if there is such a pair.

   Axis aligned means that all the rectangle sides are either parallel or perpendicular to the x and yaxis.

   You can assume that each rectangle object has two variables in it: the xy coordinates of the upperleft corner and the bottomright corner.

   Good Answer: Create a sorted array of the x coordinates of the left and right edges of the rectangles. Then, use a "scanline" to move from left to right through the rectangles. Keep a binary search tree containing the y coordinates of the top and bottom edges of the rectangles that overlap the scanline. For each element of the array, check whether it is a left or right edge. If it is a right edge, remove the corresponding top and bottom edges from the BST. If it is a left edge, search the BST for rectangles that overlap the current rectangle; if there is one, return the overlap. Then, add the y coordinates of the top and bottom edges of the rectangle to the BST. The search takes O(n log n) time, since it takes O(n log n) time to sort the rectangles and each of the 2n iterations takes O(log n) time.