

Algorithms for Interviews

Raymond Xu

raymond@adicu.com

raymondxu.io



This Talk

Sequel to [Data Structures for Interviews](#)

- This talk is more challenging
- Assumes data structures proficiency

for each:

- Basic Principles
- Example Problems
- Study Guide

Outline

Sorting

Recursion

Greedy

Dynamic Programming

Outline

Sorting

Recursion

Greedy

Dynamic Programming

Sorting

Given a **collection** of comparable elements, sort them.

Collection: Array, ArrayList, LinkedList, Stack, Queue

(Relevant) Sorting Algorithms

Slowest

$O(n^2)$

Selection Sort, Insertion Sort

$O(n \log n)$

Quicksort, Mergesort, Heapsort

$O(n)$

Bucket Sort, Radix Sort

Fastest

Lightning Review of Sorts!

Selection Sort

Repeatedly select the smallest unsorted element and place it right after the sorted elements.

Insertion Sort

Repeatedly slide each element left until it is in the proper relative place.

Bucket Sort

Scatter elements into buckets, sort within each bucket, and combine the buckets.

Radix Sort

Sort within significant positions for all significant positions.

Heapsort

Build a heap and repeatedly extract the root.

Mergesort

Repeatedly divide lists into two sublists, repeatedly merge the sublists together in sorted order.

> Recursion Tree Breakdown

Quicksort

Sort elements only with respect to a pivot such that the pivot is in its final location,
Recur on left and right sublists.

> Recursion Tree Breakdown

Study Guide

Implement the $n \log n$ sorts.

What are the best and worst case inputs for each sort?

- Runtimes?

How do you sort a Linked List? How about a stack or queue?

- Runtimes?

- Space complexities?

Outline

Sorting

Recursion

Greedy

Dynamic Programming

Recursion

Use recursion when the solution to the problem depends on solutions to smaller instances of the same problem.

> Fibonacci Recursion Tree Breakdown

$$\text{fib}(0) = 1$$

$$\text{fib}(1) = 1$$

$$\text{fib}(n) = \text{fib}(n-2) + \text{fib}(n-1) \quad \textit{for } n > 1$$

Divide and Conquer

Dividing a problem into subproblems that are solved recursively and then combined to solve the original problem.

Divide and Conquer

Dividing a problem into subproblems that are solved recursively and then combined to solve the original problem.

Examples:

Binary search

Quicksort

Mergesort

Fast Integer Multiplication

Recursion

BST Sum

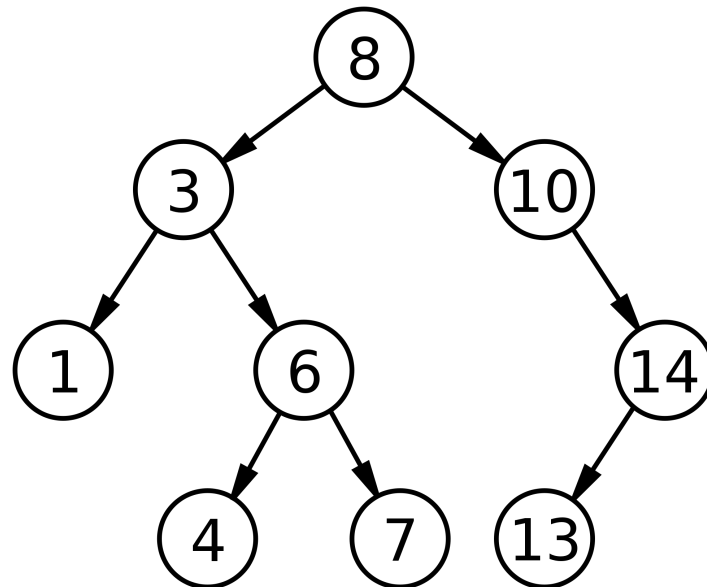
- Find the sum of a BST where each node has an integer

Linked List Merge

- Merge two sorted Linked Lists in place

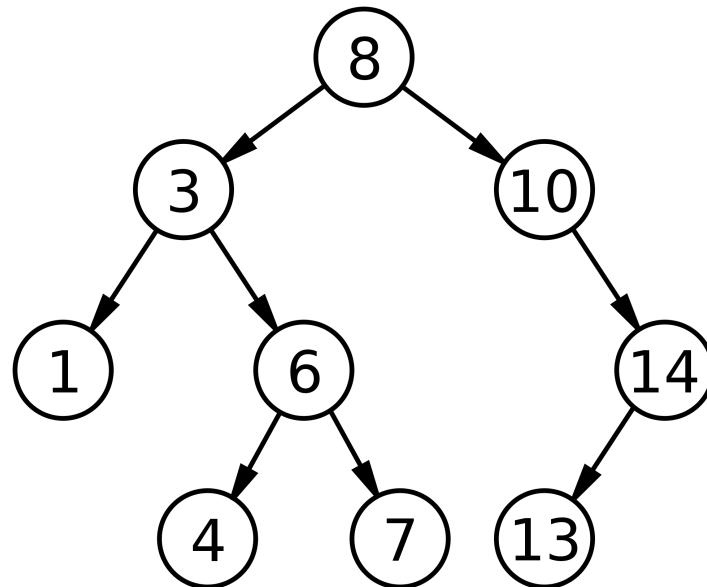
BST Sum

Problem: Find the sum of all the nodes in a BST where each node has an integer.



BST Sum

Solution: Pass the values of the each node from the leaves to the root and sum them off of the recursive stack.



BST Sum

```
int bstSum(Node n) {  
    if (n == null)  
        return 0;  
    return n.value + bstSum(n.left) +  
           bstSum(n.right);  
}
```

Linked List Merge

Problem: Merge two sorted Linked Lists in place.

Linked List Merge

Problem: Merge two sorted Linked Lists in place.

Solution: Use recursion to pass back the appropriate “next” node to the previous nodes.

Linked List Merge

```
Node merge(Node list1, Node list2) {
    if (list1 == null) { return list2; }
    if (list2 == null) { return list1; }

    if (list1.val < list2.val) {
        list1.next = merge(list1.next, list2);
        return list1;
    }
    else {
        list2.next = merge(list1, list2.next);
        return list2;
    }
}
```

Study Guide

Practice a lot of recursion problems:

- Develop base case instinct**
- Learn data passing themes**
- Analyze runtime**

Trees, sorting, searching

Outline

Sorting

Recursion

Greedy

Dynamic Programming

Greedy

Greedy algorithms take the optimal choice at each local step, which produces an optimal/almost-optimal global result.

Greedy

Coin change

- Minimum number of coins needed to represent n cents

Kruskal's Algorithm

- Minimum Spanning Tree

Coin Change

Problem: Find the minimum number of coins needed to represent n cents.

Coin Change

Problem: Find the minimum number of coins needed to represent n cents.

Solution: Starting from the largest denomination, use as many coins as you can until you have to move to a smaller denomination.

Coin Change

```
int coinChange(int n) {
    int numCoins = 0;

    while (n >= 25) {
        n -= 25;
        numCoins++;
    }
    while (n >= 10) {
        n -= 10;
        numCoins++;
    }
    ...
    return numCoins;
}
```

Kruskal's Algorithm

Problem: Find a Minimum Spanning Tree of a graph.

Kruskal's Algorithm

Problem: Find a Minimum Spanning Tree of a graph.

Solution: Repeatedly select the smallest edge that does not form a cycle with the selected edges.

Kruskal's Algorithm

```
function kruskal(set of edges) {  
  -init a set of edges to represent the MST edges  
  -init a set for each vertex (to detect cycles)  
  -init a min heap and add all graph edges into it  
  -while heap is not empty:  
    -pop the min edge  
    -if the min edge does not form a cycle with the  
    MST edges:  
      -add the edge to the MST edges set  
      -union the vertex sets  
  -return the MST edges set  
}
```

Study Guide

Study common greedy problems:

- Activity Scheduling**
- Coin Change**
- MST**
- Graph Bipartition**

Build intuition on whether a greedy strategy could be applicable to a problem

Outline

Sorting

Recursion

Greedy

Dynamic Programming

Dynamic Programming

Building up to an optimal solution to a problem using the optimal solutions to subproblems.

DP

DP

- bottom-up
- optimal substructure
- overlapping, repeating subproblems
- tabulation vs memoization

DP vs Recursion

DP

- bottom-up
- optimal substructure
- overlapping, repeating subproblems
- tabulation vs memoization

Recursion

- top-down
- distinct subproblems

Dynamic Programming

Rod Cutting

- Cut a rod into discrete pieces, each length has a value, maximize value

Longest Increasing Subsequence

- Find the length of the longest subsequence in an array of integers

Rod Cutting

Problem: Given a rod of length n , a table of lengths and values, and unlimited cuts, determine the maximum value obtainable.

Rod Cutting

Problem: Given a rod of length n , a table of lengths and values, and unlimited cuts, determine the maximum value obtainable.

value	1	5	8	9	10	17	18	20
length	1	2	3	4	5	6	7	8

For $n = 8$, the maximum value is 22 by cutting the rod into two rods of lengths 2 and 6.

Rod Cutting

Solution:

$dp[i]$ stores the optimal value attainable from a rod of length i

Compute $dp[i]$ by considering all indices j less than i
find the maximum $(value[j] + dp[i - j])$ and set $dp[i]$ to this value

The solution is in $dp[n]$

Rod Cutting

```
int cutRod(int[] value, int n) {
    int[] dp = new int[n + 1];

    for (int i = 1; i <= n; i++) {
        int max = Integer.MIN_VALUE;
        for (int j = 1; j < i; j++) {
            max = Math.max(max, value[j]
                            + dp[i - j]);
        }
        dp[i] = max;
    }
    return dp[n];
}
```

Rod Cutting

Time Complexity: $O(n^2)$

Space Complexity: $O(n)$

Classic recursive solution has a time complexity of $O(2^N)$

Longest Increasing Subsequence

Problem: Find the length of the longest increasing subsequence in an array of integers.

Longest Increasing Subsequence

Problem: Find the length of the longest increasing subsequence in an array of integers.

```
arr = [8, 2, 5, 3, 10, 1, 30, 76]  
lis = [2, 5, 10, 30, 76]
```

Longest Increasing Subsequence

Solution:

$dp[i]$ stores the length of the LIS that ends at the element at index i

Compute $dp[i]$ by considering all indices j less than i
if $(dp[j] + 1 > dp[i])$ and $(arr[j] < arr[i])$
then we can update $dp[i]$

The solution is the maximum value in the dp array

Longest Increasing Subsequence

```
int lis(int[] arr) {
    // Initialize dp array and set all entries to 1
    int dp[] = new int[arr.length];
    for (int x = 0; x < n; x++) dp[x] = 1;

    // Fill in dp array
    for (int i = 0; i < n; i++)
        for (int j = 0; j < i; j++)
            if (arr[j] < arr[i] && dp[j] + 1 > dp[i])
                dp[i] = dp[j] + 1;

    // Find lis length
    int max = 0;
    for (x = 0; x < n; x++)
        max = Math.max(max, dp[x]);

    return max;
}
```

Longest Increasing Subsequence

Time Complexity: $O(n^2)$

Space Complexity: $O(n)$

There exist more efficient algorithms for

LIS: [O\(n log n\) solution](#)

Study Guide

Focus on 1D DP problems:

- Base case (initialize array)**
- Recurrence (build the array)**
- Solution (where in the array is it?)**

The hardest part is figuring out how to build the recurrence

Extra-credit: Practice some 2D DP problems

Outline

Sorting

Recursion

Greedy

Dynamic Programming

Definitely know

Sorting

Recursion

Good-to-know

Greedy

Dynamic Programming

Outline

Sorting

Recursion

Greedy

Dynamic Programming

Resources

Most interviews don't demand much formal algorithms knowledge.

Problems

- HackerRank
- GeeksForGeeks
- Leetcode
- CTCI

Theory

- Analysis of Algorithms (CSOR 4231)
- CLRS

Algorithms for Interviews

Raymond Xu

raymond@adicu.com

raymondxu.io

