i206: Lecture 7:
Analysis of Algorithms, Sorting

Tapan Parikh

Spring 2013

Some slides courtesy Marti Hearst, John Chuang and others

Algorithms

- A clearly defined procedure for solving a problem.
- A clearly described procedure is made up of simple clearly defined steps.
- The procedure can be complex, but the steps must be simple and unambiguous.

Problem Solving Strategies

- Try to work the problem backwards
  - Reverse-engineer
  - Once you know it can be done, it is much easier to do
  - What are some examples?
- Stepwise Refinement
  - Break the problem into several sub-problems
  - Solve each sub-problem separately
  - Produces a modular structure
- Look for a related problem that has been solved before
  - Software design patterns

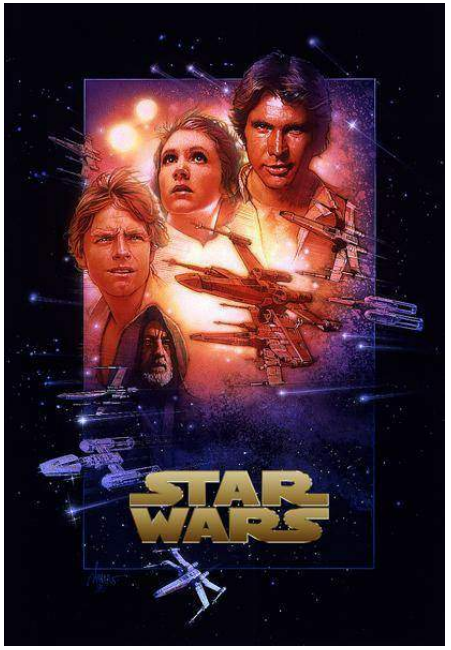
Example of Stepwise Refinement

Spiderman

- Peter Parker's goal: Make Mary Jane fall in love with him
- To accomplish this goal:
  - Get a cool car
  - To accomplish this goal:
    - Get $3000
    - To accomplish this goal:
      - Appear in a wrestling match …
- Each goal requires completing just one subgoal

Example of Stepwise Refinement

Star Wars Episode IV



- Luke Skywalker's goal: Make Princess Leia fall in love with him (they weren't siblings yet)
- To accomplish this goal:
  - Rescue her from the death star
  - To accomplish this goal:
    1. Land on Death Star
    2. Remove her from her Prison Cell
    3. Disable the Tractor Beam
    4. Get her onto the Millennium Falcon
  - To accomplish subgoal (2)
    - Obtain Storm Trooper uniforms
      - Have Wookie pose as arrested wild animal
    - Find Location of Cell
      - Have R2D2 communicate coordinates
  - To accomplish subgoal (3)
    - Have last living Jedi walk across catwalks
  - To accomplish subgoal (4)
    - Run down hall
    - Survive in garbage chute
      - Fight garbage monster
      - Have R2D2 stop compaction …

“Divide and Conquer” Algorithms

- Break the problem into smaller pieces
- Recursively solve the sub-problems
- Combine the results into a full solution

- If structured properly, this can lead to a much faster solution.

“Divide and Conquer” Algorithms

- Problem: Cut up the licorice into 64 pieces!.

“Divide and Conquer” Algorithms

- Problem: Cut up the licorice into 64 pieces!
- How do we do this in:
  - O(n)
  - O(log n)
  - O(sqrt n)

Which Algorithm is Better? What do we mean by “Better”?

- Sorting algorithms
  - Bubble sort
  - Insertion sort
  - Shell sort
  - Merge sort
  - Heapsort
  - Quicksort
  - Radix sort
  - …
- Search algorithms
  - Linear search
  - Binary search
  - Breadth first search
  - Depth first search
  - …

Search Example: Phonebook

- Given:
  - A physical phone book
    - Organized in alphabetical order
  - A name you want to look up
  - An algorithm in which you search through the book sequentially, from first page to last
  - What is the order of:
    - The best case running time?
    - The worst case running time?
    - The average case running time?
  - What is:
    - A better algorithm?
    - The worst case running time for this algorithm?

Analysis Example (Phonebook)

- This better algorithm is called Binary Search
- What is its running time?
  - First you look in the middle of *n* elements
  - Then you look in the middle of *n/2* = ½**n* elements
  - Then you look in the middle of ½ * ½**n* elements
  - …
  - Continue until there is only 1 element left
  - Say you did this *m* times: ½ * ½ * ½* …**n*
  - Then the number of repetitions is the smallest integer *m* such that $\frac{1}{2^m} n = 1$

Binary Search Pseudo-code

Brookshear Figure 5.14

```
procedure Search (List, TargetValue)
if (List empty)
  then
     (Report that the search failed.)
   else
     [Select the "middle" entry in List to be the TestEntry;
      Execute the block of instructions below that is
         associated with the appropriate case.
            case 1: TargetValue = TestEntry
                    (Report that the search succeeded.)
            case 2: TargetValue < TestEntry
                    (Apply the procedure Search to see if TargetValue
                          is in the portion of the List preceding TestEntry,
                          and report the result of that search.)
            case 3: TargetValue > TestEntry
                   (Apply the procedure Search to see if TargetValue
                         is in the portion of List following TestEntry,
                         and report the result of that search.)
     ] end if
```

Analyzing Binary Search

- In the worst case, the number of repetitions is the smallest integer *m* such that $\frac{1}{2^m} n = 1$

- We can rewrite this as follows:

$$\frac{1}{2^m} n = 1$$

$$n = 2^m$$ Multiply both sides by 2^m

$$\log n = m$$ Take the log of both sides

Since *m* is the worst case time, the algorithm is *O(logn)*

Substrings of a String

Say you have a long string, such as "supercalifragilisticexpialidocious".

Say you want to print out all the substrings of this string, starting from the left, and always retaining the substring you have printed out so far, e.g. "s", "su", "sup", "supe", and so on.

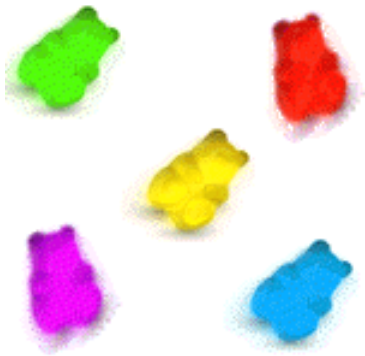
Assuming this string is of length n, how many substrings will you print out in terms of big-O notation?

Sorting

Sorting

Putting collections of things in order

- Numerical order
- Alphabetical order
- An order defined by the domain of use
  - E.g. color, yumminess …



http://www.hugkiss.com/platinum/candy.shtml

Bubble Sort

```
BubbleSort( int a[], int n)
Begin
    for i = 1 to n-1
        sorted = true
        for j = 0 to n-1-i
            if a[j] > a[j+1]
                temp = a[j]
                a[j] = a[j+1]
                a[j+1] = temp
                sorted = false
        end for
        if sorted
            break from i loop
    end for
End
```

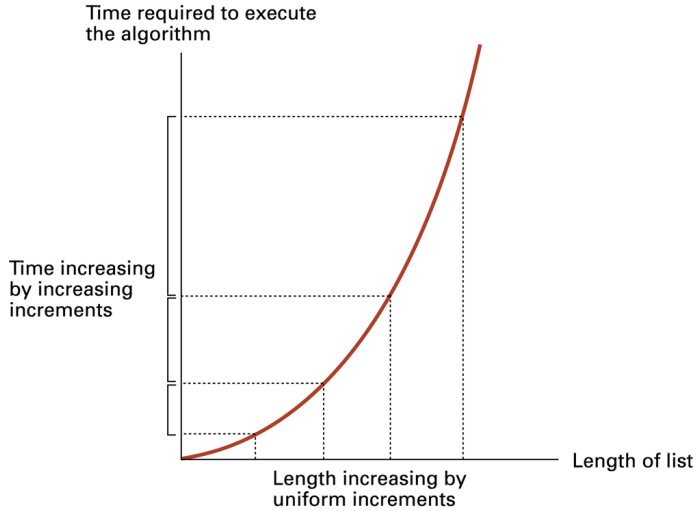
The dark gray represents a comparison and the white represents a swap.

http://www.youtube.com/watch?v=P00xJgWzz2c

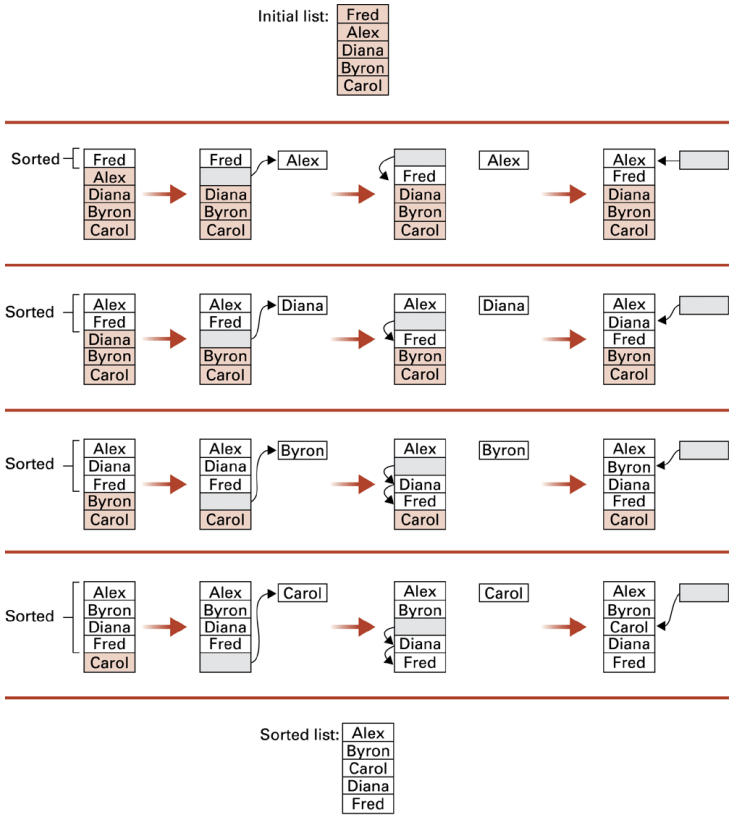
Insertion Sort

```
procedure Sort (List)
N ← 2;
while (the value of N does not exceed the length of List) do
      (Select the Nth entry in List as the pivot entry;
       Move the pivot entry to a temporary location leaving a hole in List;
        while (there is a name above the hole and that name is greater than the pivot) do
              (move the name above the hole down into the hole leaving a hole above the name)
       Move the pivot entry into the hole in List;
       N ← N + 1
      )
```

Brookshear Figure 5.11

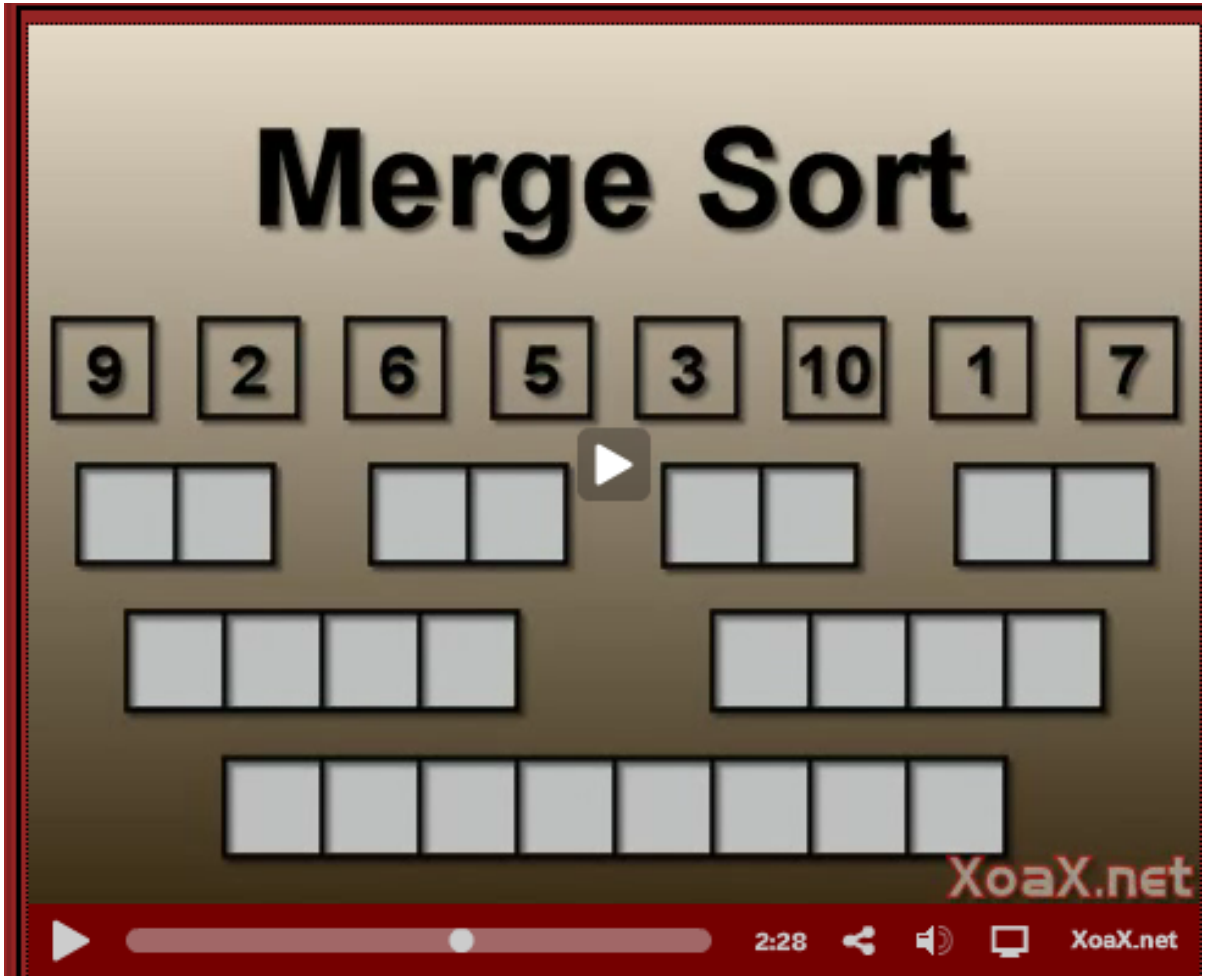


Brookshear Figure 5.19



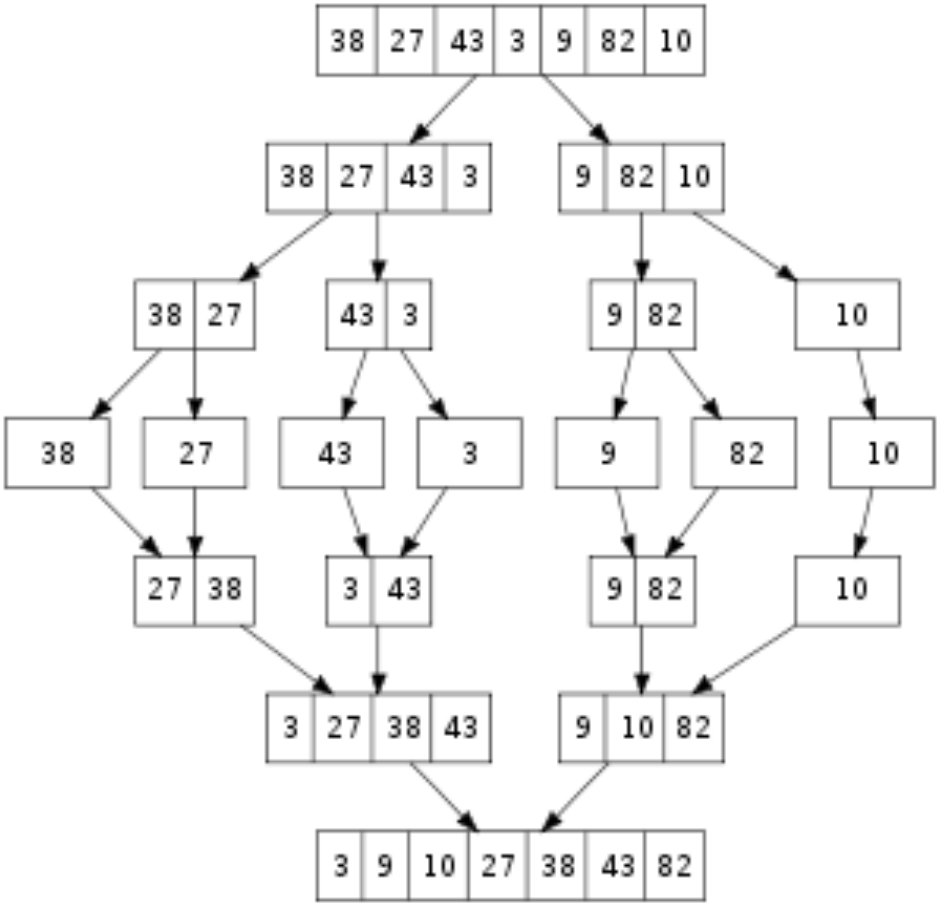
http://xoax.net/comp/sci/algorithms/Lesson2.php

Merge Sort



http://www.youtube.com/watch?v=GCae1WNvnZM

Merge Sort



https://en.wikipedia.org/wiki/Mergesort

Merge Sort O(n log n)

```
function merge_sort(list m)
    // if list size is 1, consider it sorted and return it
    if length(m) <= 1
        return m
    // else list size is > 1, so split the list into two sublists
    var list left, right
    var integer middle = length(m) / 2
    for each x in m up to middle
         add x to left
    for each x in m after or equal middle
         add x to right
    // recursively call merge_sort() to further split each sublist
    // until sublist size is 1
    left = merge_sort(left)
    right = merge_sort(right)
    // merge the sublists returned from prior calls to merge_sort()
    // and return the resulting merged sublist
    return merge(left, right)
```

http://xoax.net/comp/sci/algorithms/Lesson2.php

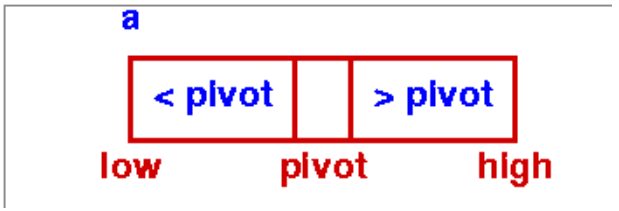
Merge Sort O(n log n)

```
function merge(left, right)
    var list result
    while length(left) > 0 or length(right) > 0
        if length(left) > 0 and length(right) > 0
            if first(left) <= first(right)
                append first(left) to result
                left = rest(left)
            else
                append first(right) to result
                right = rest(right)
        else if length(left) > 0
            append first(left) to result
            left = rest(left)
        else if length(right) > 0
            append first(right) to result
            right = rest(right)
    end while
    return result
```

http://xoax.net/comp/sci/algorithms/Lesson2.php

QuickSort

- Another Divide-and-Conquer recursive algorithm
- Divide:
  - If only one item in list, return it.
  - Else choose a pivot, and
  - Divide the list into 3 subsets
    - Items < pivot, pivot, items > pivot



- Recurse:
  - Recursively solve the problem on the subsets
- Conquer:
  - Merge the solutions for the subproblems
    - Concatenate the three lists

QuickSort Visualizations

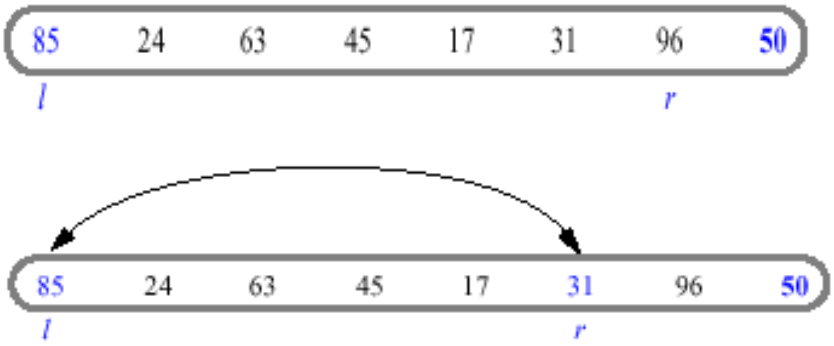
- Wikipedia:
  - https://en.wikipedia.org/wiki/File:Sorting_quicksort_anim.gif
- Xoax:
  - http://www.youtube.com/watch?v=y_G9BkAm6B8
- The Sorter:
  - http://www.youtube.com/watch?v=2HjspVV0jK4
- Robot battle:
  - http://www.youtube.com/watch?v=vxENKlcs2Tw

Partitioning in QuickSort

- Partition:
  - Select a key (called the Pivot) from the dataset
  - Divide the data into two groups such that:
    - All the keys less than the pivot are in one group
    - All the keys greater than the pivot are in the other
  - Note: the two groups are NOT required to be sorted
    - Thus the data is not sorted, but is “more sorted” than before
    - Not too much more work to get them sorted.
  - What is the order of this operation?
    - Linear (or O(n))

QuickSort

- An advantage: can be done “in place”
  - Meaning you don’t need to allocate an additional array or vector to hold the temporary results
- Do the in-place swapping in the divide step
  - Choose pivot
  - Have two indexes: L and R
  - While L is < R
    - Move L forwards until A[L] > pivot
    - Move R backwards until A[R] < pivot
    - Swap them



Quick Sort – Choosing the Pivot

- Alternative Strategies
  - Choose it randomly
    - Can show the expected running time is O(n log n)
  - Sample from the set of items, choose the median
    - Takes more time
    - If sample is small, only a constant overhead

Quick Sort

- Running Time analysis
  - If the pivot is always the median value
    - Each list is divided neatly in half
    - The height of the sorting tree is O(log n)
    - Each level takes O(n) for the divide step
    - O(n log n)
  - If the list is in increasing order and the pivot is the last value
    - Each list ends up with one element on the right and all the other elements on the left
    - The height of the sorting tree is O(n)
    - Each level takes O(n) for the divide step
    - O(n^2)
  - However, in practice (and in the general case), QuickSort is usually the fastest sorting algorithm.

QuickSort vs. MergeSort

- Similarities
  - Divide array into two roughly equal-sized groups
  - Sorts the two groups by recursive calls
  - Combines the results into a sorted array
- The difficult step
  - Merge sort: the merging
  - Quick sort: the dividing
- Running Time
  - QuickSort has a worse worst case running time
  - But in practice it is faster than the others
    - Constants, and the way data is distributed
  - Also, MergeSort requires an additional array

Merge Sort O(n log n)

An average and worst-case performance of O(n log n).

Based on how the algorithm works, the recurrence
T(n) = 2T(n/2) + n = n log n describes the average running time.

(Apply the algorithm to two lists of half the size of the original list, and add the n steps taken to merge the resulting two lists.)

In the worst case, the number of comparisons merge sort makes is equal to or slightly smaller than
($n \log n - 2^{\log n} + 1$),
which is between ($n \log n - n + 1$) and ($n \log n + n + O(\lg n)$).

Santa's Socks

- Problem:
  - The elves have packed all 1024 boxes with skates
  - But Santa's dirty smelly socks fell into one of them.
  - We have to find the box with the dirty socks! But it's almost midnight!!!!

Santa’s Socks

- http://www.youtube.com/watch?v=wVPCT1VjySA

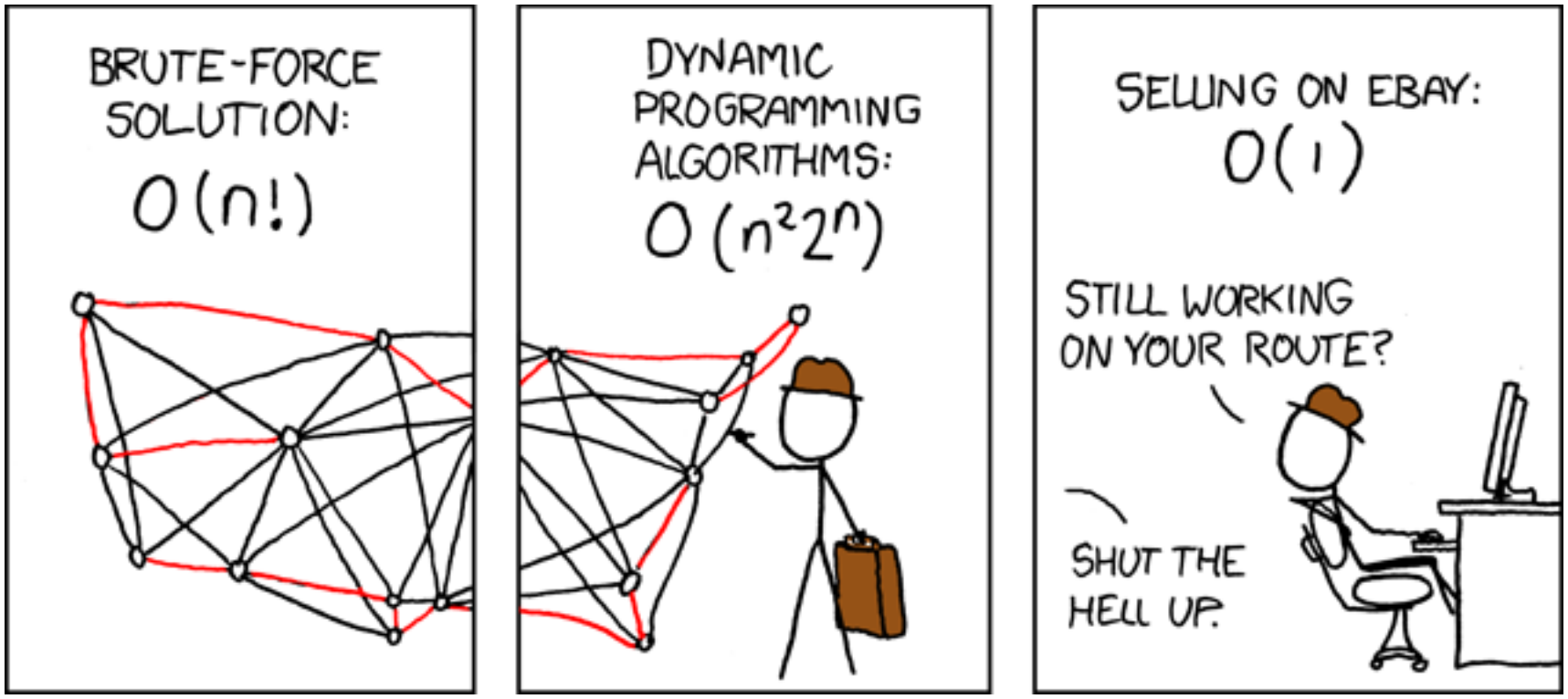
Hint for Homework Problem 1

- Hint 1:
  - Write a program that computes the total number of gifts for each value of N just to get a feeling for how it works.

- Hint 2:
  - Triangle proof from webpage shows that:
    1 + 2 + 3 + … + n = n(n + 1)/2

  - Similarly, to add up the number of gifts, a closed form version of the total numbers is:
    (n/6)(n+1)(n+2)

Traveling Salesman Problem



http://xkcd.com/399/

Summary: Analysis of Algorithms

- A method for determining, in an abstract way, the asymptotic running time of an algorithm
  - Here asymptotic means as *n* gets very large
- Useful for comparing algorithms
- Useful also for determing *tractability*
  - Meaning, a way to determine if the problem is intractable (impossible) or not
  - Exponential time algorithms are usually intractable.
- We'll revisit these ideas throughout the rest of the course.