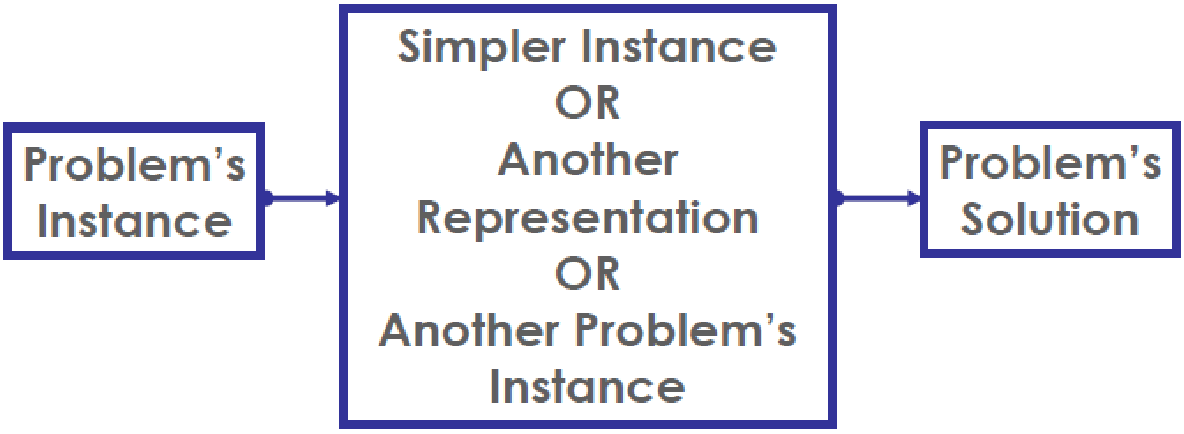
Transform and Conquer

> The secret of life is to replace one worry with another

Charles M. Schultz



Different types of transformations:

1. Instance simplification = a more convenient instance of the same problem
   - Presorting
   - Gaussian elimination
2. Representation Change = a different representation of the same instance
   - Balanced search trees
   - Heaps and heapsort
   - Polynomial evaluation by Horner's rule
   - Binary exponentiation
3. Problem reduction = a different problem altogether
   - Lowest Common Multiple
   - Reduction to Graph Problems

Instance Simplification

Presorting

- Solve instance of problem by preprocessing the problem to transform it into another simpler/easier instance of the same problem
- Many problems involving lists are easier when list is sorted
  - Searching
  - Computing the median (selection problem)
  - Finding repeated elements
  - Convex hull & Closest Pair
- Efficiency
  - Introduce the overhead of an Θ(nlogn) preprocess
  - But the sorted problem often improves by at least one base efficiency class over the unsorted problem (e.g.: $\Theta(n^2)$ -> Θ(n))

Example sorting is Θ(nlogn) so transformation to sorting is only worthwhile if other algorithms are less efficient

- Checking uniqueness
  - Brute force is $\Theta(n^2)$ so sorting is more efficient
- Finding the mode
  - Brute force is $\Theta(n^2)$ so sorting is more effective
- Searching an array
  - Brute force is Θ(n) so sorting is not better

Example Finding Repeated Elements

- Presorting algorithm for finding duplicated elements in a list
  - Use mergesort Θ(nlogn)
  - Scan to find repeated element: Θ(n)
- Brute force algorithm
  - Compare each element to every other: $\Theta(n^2)$
- Conclusion: presorting yields **significant** improvement

Example Presorted selection

- Finding the k^{th} smallest element in A[1], ..., A[n]
- Special cases
  - Min: k = 1
  - Max: k = n
  - Median: k = n/2
- Presorting-based algorithm
  - Sort list
  - return A[k]
- Partition-based algorithm (Variable Decrease & Conquer)

```
pivot/split A[s] usingPartitioning algorithm from Quicksort
if s == k:
    return A[s]
```

```
    return A[s]
else if s< k:
    repeat with sublist A[s+1], ..., A[n]
else if s>k repeat with sublist A[1], ..., A[s-1]
```

IF we look at this algorithm:

- the presorting based one is Θ(nlogn) + Θ(1) = Θ(nlogn)
- The partitioning based algorithm (which is variable size decrease & conquer)
  - Worst case T(n) = T(n-1) + (n+1) ∈ Θ (n^2)
  - Best case: Θ(n)
  - Average case: T(n) = T(n/2) + (n+1) ∈ Θ (n)
  - Also identifies the k smallest elements (not just the k^{th})
- Simpler linear (brute force) algorithm is better in the case of max & min
- Conclusion
  - Presorting does not help in this case

Representation Change

Trees

- Searching, insertion and deletion in a Binary Search Tree:
  - Balanced = Θ(logn)
  - Unbalanced = Θ(n)
- Instance Simplification
  - AVL & Red-black trees constrain imbalances by restructuring trees using rotations
- Representation Change
  - B+ Trees attain perfect balance by allowing more than one element in a node

Heapsort

- A heap is binary tree with conditions
  - It is essentially complete
  - The key at each node is >= keys at its children
  - The root has the largest key
  - The subtree rooted at any node of a heap is also a heap
- Heapsort algorithm
  1. Build heap
  2. Remove root - exchange with last (rightmost) leaf
  3. Fix up heap (excluding last leaf)
  4. Repeat 2,3 until heap contains just one node
- Efficiency
  - Θ(n) + Θ(nlogn) in both worst and average cases
  - Unlike mergesort it is in place

Calculating Polynomials

Here is an example of a polynomial (p(x)), and the associated calculation for p(3)

$p(x) = 2x^4 - x^3 + 3x^2 + x - 5$

- Evaluate for x = 3

The traditional, obvious, brute force way: $p(3) = 2(3)^4 - (3)^3 + 3(3)^2 + (3) - 5$

Brute Force Polynomial

- For a polynomial of size n, just the first term $a_n x^n$ requires n multiplications using brute force
- We can improve on this by efficiently calculating x^n
- But Horner's rule does even better for large polynomials and it's dead easy

Horner's Rule

Factor x out as much as possible - so using the same equation as above we can factor out x as follows:

$p(x) = 2x^4 - x^3 + 3x^2 + x - 5 = (2x^3 - x^2+3x+ 1)x - 5 =((2x^2 - x + 3)x+1)x-5 =(((2x-1)x+3)x+1)x-5$

Here is another example: $p(x) = 2x^3 - x^2 - 6x + 5 = (2x^2-x-6)x + 5 = ((2x-1)x-6)x+5$

Find p(x) at x = 3 $2x^3$ $-x^2$ -6x +5 c[]:2 -1 -6 +5 p: 2 2*3+(-1)= 5 5*3 + (-6) = 9 9*3 + 5 = 32

```
double horner(coefficients[0 .. n],x){
    p = coefficients[n]
    for i = n-1 to 0:
        p = x\*p + coefficients[i]

    return p
}
```

- Horner's rule addresses the problem of evaluating a polynomial $p(x) = a_n x^n + a_{n-1}x^{n-1} + ... + a_1 + a_0$ at a given point $x = x_0$
- Re invested by W. Horner in early 19th Century
- Approach
  - Convert p(x)
- Algorithm

```
p=P[n]
for i <- n-1 downto 0:
    p <- x * p + P[i]
return p
```

$Q(x) = 2x^3 - x^2 - 6x + 5$ at $x = 3$

P[ ]:	2	-1	-6	5
p:	2	3*2 + (-1) = 5	3*5 + (-6) = 9	3*9 + 5 = 32

The slides go into Horner's rule in a lot more depth here

Binary Exponentiation

To find a^n, represent n in binary as

$b_j\, b_{j-1} \cdots b_1\, b_0$

a^n can be computed as the product

$f_0 * f_1 * f_2 * \cdots * f_{j-1} * f_j$

where f_k = $\exp(a, 2^k)$ if b_k is 1

or 1 if b_k is 0

```
f = a; // start at exp(a,2^0);
if (b_0 == 1) ans = a else ans = 1; // bit 0 done
for (k=1; k<=j; k++)
    { f = f * f; // next power of 2
    if (b_k == 1) ans = ans * f }
```

Problem Reduction

- If you need to solve a problem reduce it to another problem that you know how to solve
- Used in Complexity Theory to classify problems
- Computing the Least Common Multiple
  - The LCM of two positive integers m and n is the smallest integer divisible by both m and n
  - Problem reduction is to say LCM(m,n) = m * n / GCD(m,n)
  - Example: LCM(24,60) = 1440 / 12 = 120
- Reduction of Optimisation Problems
  - Maximization problems seek to find a function's maximum. Conversely, minimization seeks to find the minimum
  - Can reduce between: min f(x) = -max[-f(x)]

Lowest Common Multiple

LCM(24,60) = ?

- 24 = 2 x 2 x 2 x 3

- 60 = 2 x 2 x 3 x 5
- LCM(24,60) = 2 x 2 x 3 x 2 x 5

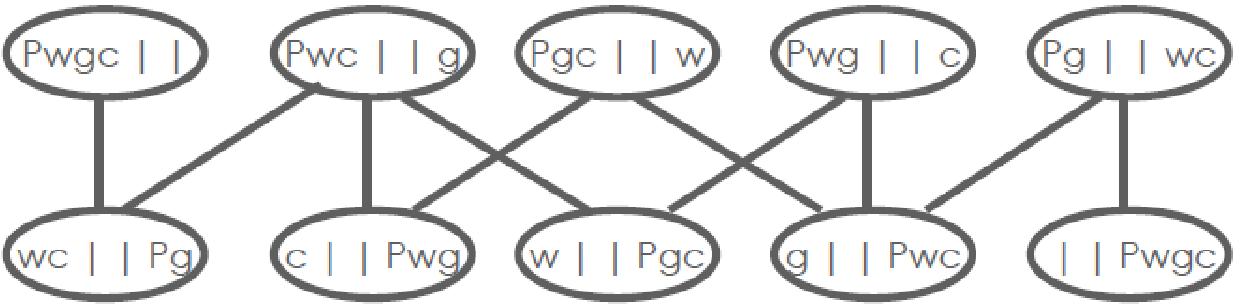
Better?

- LCM(m,n) = (m*n)/GCD(m,n)

Reduction to Graph Problems

- State-Space graphs
  - Vertices represent states and edges represent valid transitions between states
  - Start and goal vertices
  - Widely used in AI
- Example

River Crossing Puzzle [**P**easant, **W**olf, **G**oat, **C**abbage]



Strengths and Weaknesses of Transform & Conquer

- Strengths
  - Allows powerful data structures to be applied
  - Effective in Complexity Theory
- Weaknesses
  - Can be difficult to derive (especially reduction)