CS216: Program and Data Representation
University of Virginia Computer Science
Spring 2006 David Evans

# Lecture 8: Crash Course in Computational Complexity

http://www.cs.virginia.edu/cs216

## Procedures and Problems

- So far we have been talking about **procedures** (how much work is our brute force subset sum algorithm?)
- We can also talk about **problems**: how much work is the subset sum problem?

  What is a problem?

  What does it mean to describe the work of a problem?



## Problems and Solutions

- A **problem** defines a desired output for a given input.
- A **solution** to a problem is a procedure for finding the correct output for all possible inputs.
- The **time complexity** of a problem is the running time of the best possible solution to the problem



## Subset Sum Problem

- **Input:** set of $n$ positive integers, $\{w_0, \ldots, w_{n-1}\}$, maximum weight $W$
- **Output:** a subset $S$ of the input set such that the sum of the elements of $S \leq W$ and there is no subset of the input set whose sum is greater than the sum of $S$ and $\leq W$

  What is the time complexity of the subset sum *problem*?



## Brute Force Subset Sum Solution

```
def subsetsum (items, maxweight):
  best = {}
  for s in allPossibleSubsets (items):
    if sum (s) <= maxweight \
        and sum (s) > sum (best)
      best = s
  return best
```

Running time $\in \Theta(n2^n)$

What does this tell us about the time complexity of the subset sum *problem*?



## Problems and Procedures

- If we know a *procedure* that is that is $\Theta(f(n))$ that solves a *problem* then we know the problem is $O(f(n))$.
- The subset sum **problem** is in $\Theta(n2^n)$ since we know a **procedure** that solves it in $\Theta(n2^n)$
- Is the subset sum **problem** in $\Theta(n2^n)$?

  No, we would need to **prove** there is **no better procedure.**

## Lower Bound

- Can we find an Ω bound for the subset sum problem?
- It is in $\Omega(n)$ since we know we need to at least look at every input element
- Getting a higher lower bound is tough



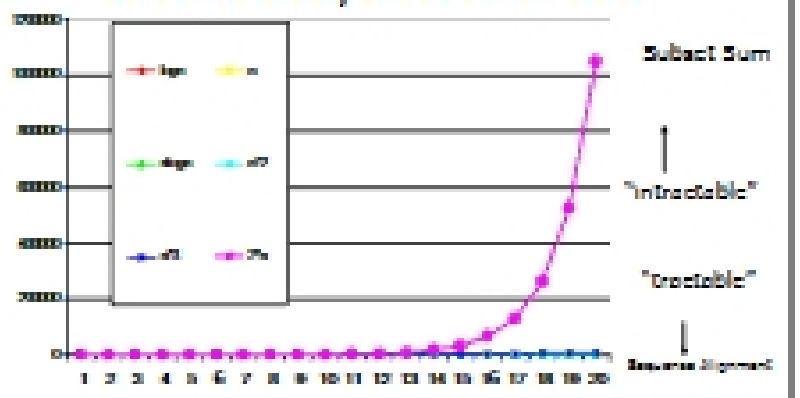
## How much work is the Subset Sum Problem?

- Upper bound: $O(2^n)$
  Try all possible subsets
- Lower bound: $\Omega(n)$
  Must at least look at every element
- Tight bound: $\Theta$(?)

No one knows!



## Tractable/Intractable




*I do nothing that a man of unlimited funds, superb physical endurance, and maximum scientific knowledge could not do.*
– Batman (may be able to solve intractable problems, but computer scientists can only solve tractable ones for large $n$)



## Complexity Class P "Tractable"

**Class P**: problems that can be solved in polynomial time

$O(n^k)$ for some constant $k$.

Easy problems like sorting, sequence alignment, simulating the universe are all in **P**.



## Complexity Class NP

**Class NP**: problems that can be solved in *nondeterministic* polynomial time

If we could try all possible solutions at once, we could identify the solution in polynomial time.

Alternately: If we had a magic guess-correctly procedure that makes every decision correctly, we could devise a procedure that solves the problem in polynomial time.

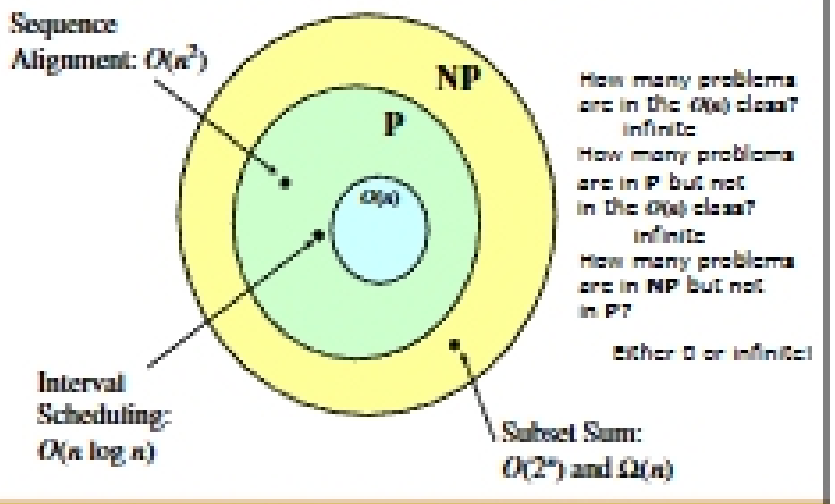


## Complexity Classes

**Class P**: problems that can be solved in polynomial time ($O(n^k)$ for some constant $k$): "myopic" problems like sequence alignment, interval scheduling are all in **P**.

**Class NP**: problems that can be solved in polynomial time by a nondeterministic machine: includes all problems in **P** and some problems *possibly* outside **P** like subset sum

## Complexity Classes: Possible View



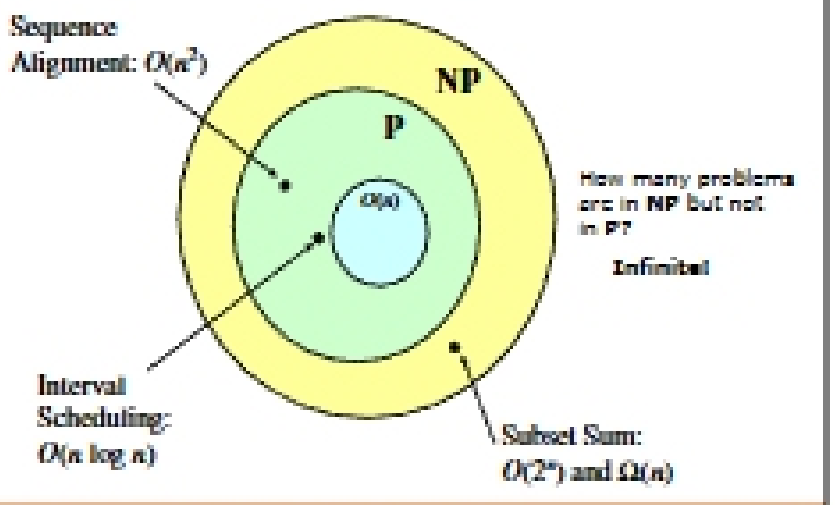



## P = NP?

- Is P different from NP: is there a problem in NP that is not also in P
  - If there is one, there are infinitely many
- Is the "hardest" problem in NP also in P
  - If it is, then every problem in NP is also in P
- No one knows the answer!
- The most famous unsolved problem in computer science and math
  - Listed first on Millennium Prize Problems

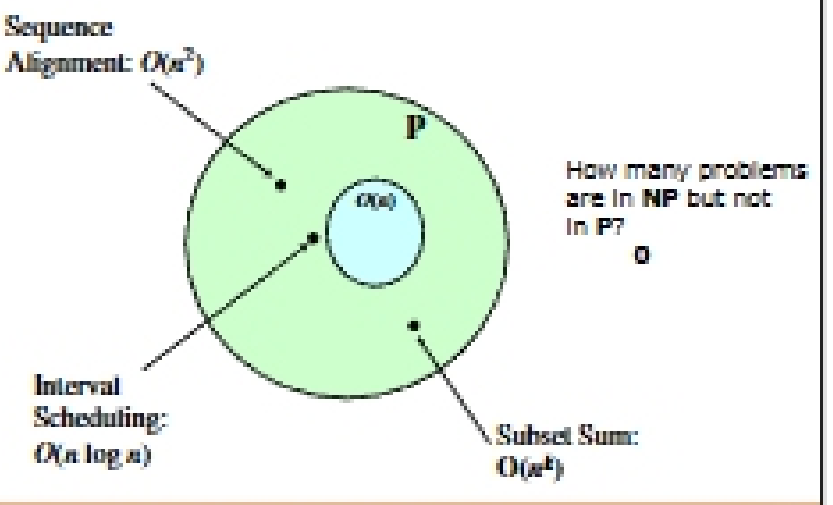


## Problem Classes if P ⊂ NP:






## Problem Classes if P = NP:






## Distinguishing P and NP

- Suppose we identify the *hardest* problem in NP - let's call it Super Arduous Task (SAT)
- Then deciding is P = NP should be easy:
  - Find a P-time solution to SAT ⇒ P = NP
  - Prove there is no P-time solution to SAT ⇒ P ⊂ NP



## The Satisfiability Problem (SAT)

- Input: a sentence in propositional grammar
- Output: Either a mapping from names to values that satisfies the input sentence or **no way** (meaning there is no possible assignment that satisfies the input sentence)