

What can a computer be commanded to do?

# Doing What You're Told

*Based on slides from  
Lawrence Snyder  
University of Washington, Seattle*

# Thinking About Computing

- Computers seem to run really fast ... except when they don't
  - Usually we don't know why
  - Often it is communications congestion on the net
  - Other times, when saving files, say, we're waiting for the hard disk to copy everything
- Often the time a computer takes to solve a problem is proportional to how much data there is ... more pixels, more time to process

# Time Proportional To $n$

- CS folks say that problems whose work is proportional to  $n$  are  $n$ -time or linear time
  - Making an image lighter in your photo software
  - Adding a column of numbers in a spreadsheet
  - Crawling the Internet looking for links
  - ... many more ... linear problems are common
- Apparently some problems are not ...

# Searching

- Guess a number between 1 and 100. How many guesses do you need?
- Scatter a deck of cards on the floor. How many do you have to turn over to find the ace of spades?

# Sorting (exchange)

- Putting a sequence of items into alphabetical or numerical order

walrus seal whale gull clam

Algorithm: compare  
to all following items,  
reorder if needed

- Other ways to sort

wa se wh g c  
se wa wh g c  
se wa wh g c  
g wa wh se c  
c wa wh se g  
c wa wh se g  
c se wh wa g  
c g wh wa se  
c g wa wh se  
c g se wh wa  
c g se wa wh

# Sorting (bubble)

- Putting a sequence of items into alphabetical or numerical order

walrus seal whale gull clam

Algorithm: compare adjacent items, reorder if needed

wa se wh g c  
se wa wh g c  
se wa wh g c  
se wa g wh c  
se wa g c wh  
se wa g c wh  
se g wa c wh  
se g c wa wh  
se g c wa wh  
g se c wa wh  
g c se wa wh  
g c se wa wh  
c g se wa wh

# How Long To Sort w/Exchange?

- The pattern is, for n items

n-1 focus on first time

n-2 focus on second item

n-3 focus on third item

...

1 on next to last

n-1 rows in list, avg of each

row is  $n/2$ , so  $(n-1) \times n/2$

$$= (n^2 - n)/2$$

- Time proportional to  $n^2$

wa se wh g c

se wa wh g c

se wa wh g c

g wa wh se c

c wa wh se g

c wa wh se g

c se wh wa g

c g wh wa se

c g wa wh se

c g se wh wa

c g se wa wh

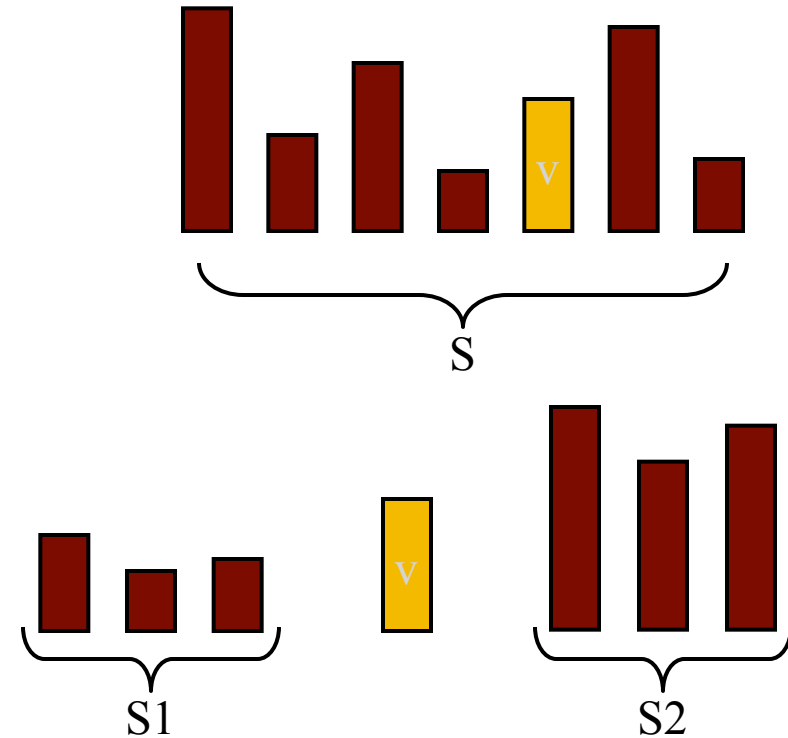
# There are Different Algorithms

- Is there a better way to do sorting?
- QUICKSORT
  - Fastest known sorting algorithm in practice
  - Average case:  $O(N \log N)$  (we don't prove it)
  - Worst case:  $O(N^2)$ 
    - But, the worst case seldom happens.
  - A divide-and-conquer recursive algorithm
- Video of selection vs quicksort: [http://youtu.be/cVMKXKoGu\\_Y](http://youtu.be/cVMKXKoGu_Y)



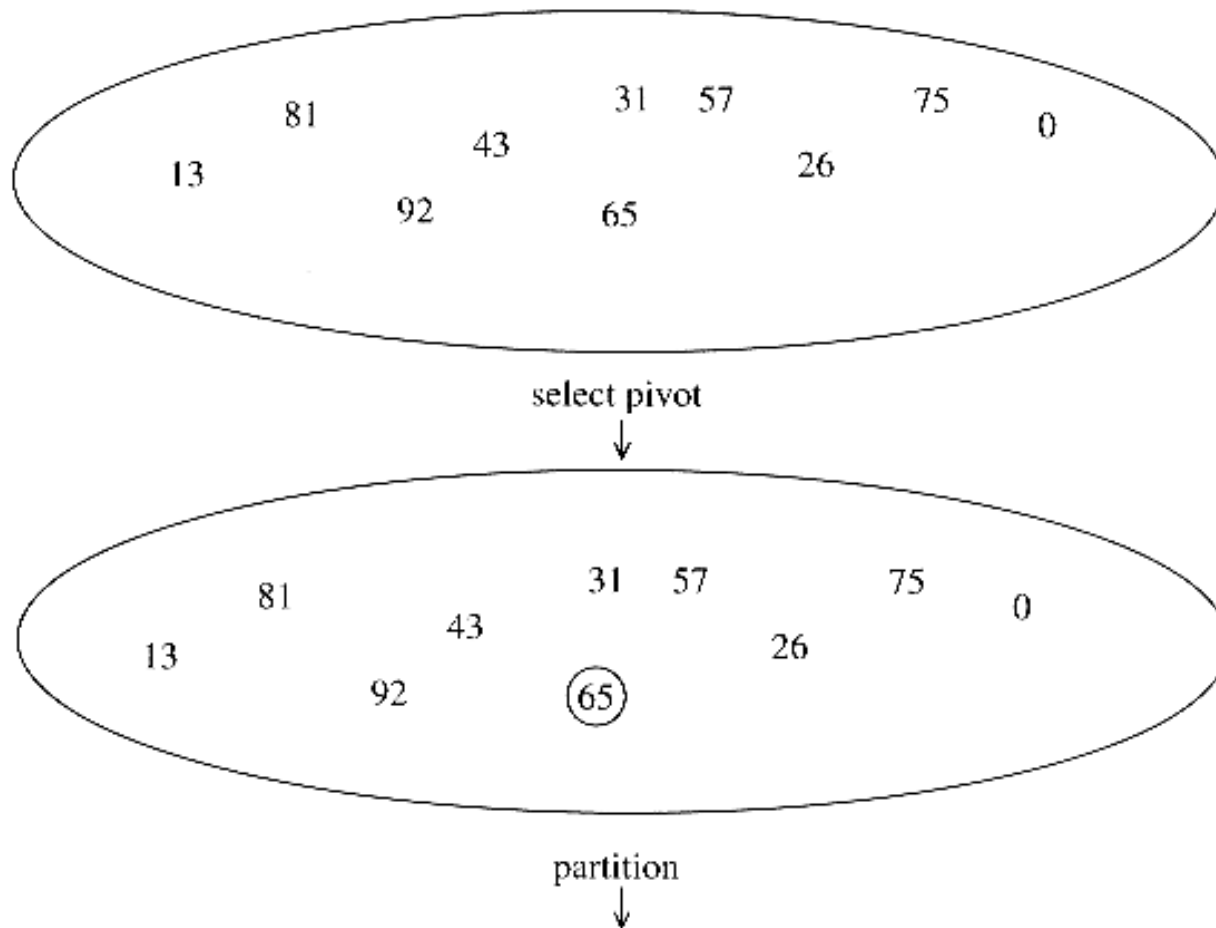
# Quicksort is the best : Divide and Conquer

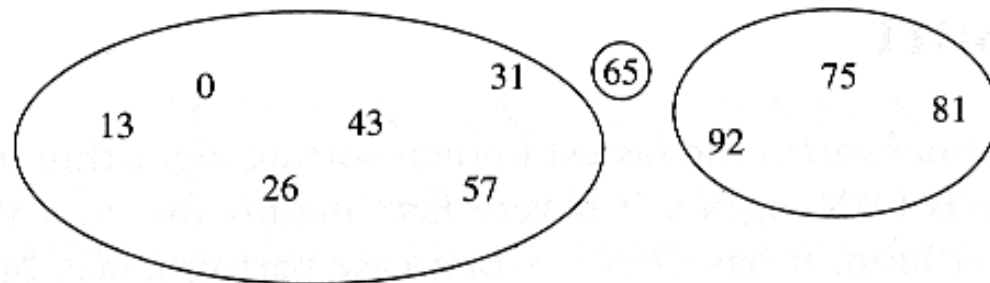
- Divide step:
  - Pick any element (**pivot**)  $v$  in  $S$
  - Partition  $S - \{v\}$  into two disjoint groups
$$S1 = \{x \in S - \{v\} \mid x \leq v\}$$
$$S2 = \{x \in S - \{v\} \mid x \geq v\}$$
- Conquer step: recursively sort  $S1$  and  $S2$
- Combine step: the sorted  $S1$  (by the time returned from recursion), followed by  $v$ , followed by the sorted  $S2$  (i.e., nothing extra needs to be done)



To simplify, we may assume that we don't have repetitive elements,  
So to ignore the 'equality' case!

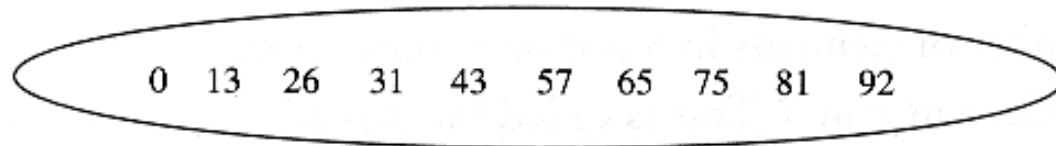
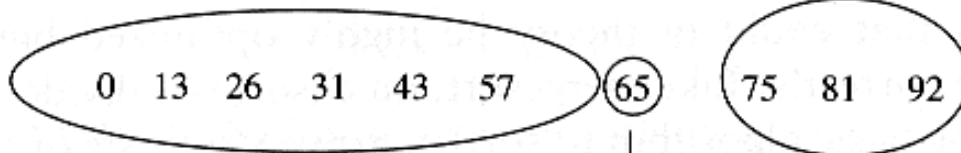
# Example





quicksort small

quicksort large



# Polynomial

- Other computations have running time
  - proportional to  $n^3$  – matrix multiplication
  - proportional to  $n^4$
  - ...
- All of them are lumped together as “*polynomial* time computations”
  - Considered to be realistic ... a person can wait
  - Polynomial, but not linear ... get a computer person to help develop your solution

# Which expression grows slowest as N gets bigger?

- A. Time proportional to  $N^2$
- B. Time proportional to  $N \times \log(N)$

# Different sorting algorithms and animations

- This one you can give different parameters for the number of things to sort, speed of animation etc.

<http://www.cs.oswego.edu/~mohammad/classes/csc241/samples/sort/Sort2-E.html>

- This one has a 'dancing animation' of quick sort.

[Hungarian Folk Dance Quick Sort http://www.youtube.com/watch?v=ywWBy6J5gz8](http://www.youtube.com/watch?v=ywWBy6J5gz8)

# To Infinity And Beyond

There are more complex computations ...

Suppose you want to visit 28 cities in the US (for a rock concert?) and you want to minimize how much you pay for airplane tickets

You could select an ordering of cities (SEA → PDX → SFO → LAX ...) and compute the ticket price.

Then pick another ordering (SEA → SFO → LAX → PDX ... ), compute this ticket price and compare to the previous one

Always keep the cheapest itinerary

This seems very dumb ... isn't there a better way?

# Traveling Salesman Problem

- Actually, no one knows a way to solve this problem significantly faster than checking all routes and picking the cheapest ...
- Not polynomial time ... guessing, No Poly sol' n
- This is an NP-Complete problem
  - Many many related problems ... the best solution is “generate and check”
    - Best way to pack a container ship
    - Most efficient scheduling for high school students' classes
    - Least fuel to deliver UPS packages in Washington
    - Fewest public alert broadcast stations for US



# Astonishing Fact

- Although there are thousands of NP-**Hard** Problems, meaning they're basically “generate and check” ...
- NP-**Complete** computations (like traveling salesman) have the property that if any one of them can be done fast ( $n^x$ -time, say) then EVERYONE of the related problems can be too!
- Is Traveling Salesman solvable in  $n^x$  time is one of the great open questions in computer science

Be Famous ... Answer This Question

# Finding an item in an unsorted list

What is the time complexity of finding an item in an unsorted list?

- A. Less than linear ( $\log(n)$  or constant)
- B. Linear
- C. Polynomial but more time than linear
- D. NP (exponential like  $2^n$ )

# There's Stuff A Computer Can't Do

- Some problems are too big – combinatorial explosion – like checking each chess game to see if there is a guaranteed win for White
  - Too many items to check
  - Doable in principle, however

# Some Well Formed Problems

- One problem that has a clear specification but can't be solved is

## Halting Problem

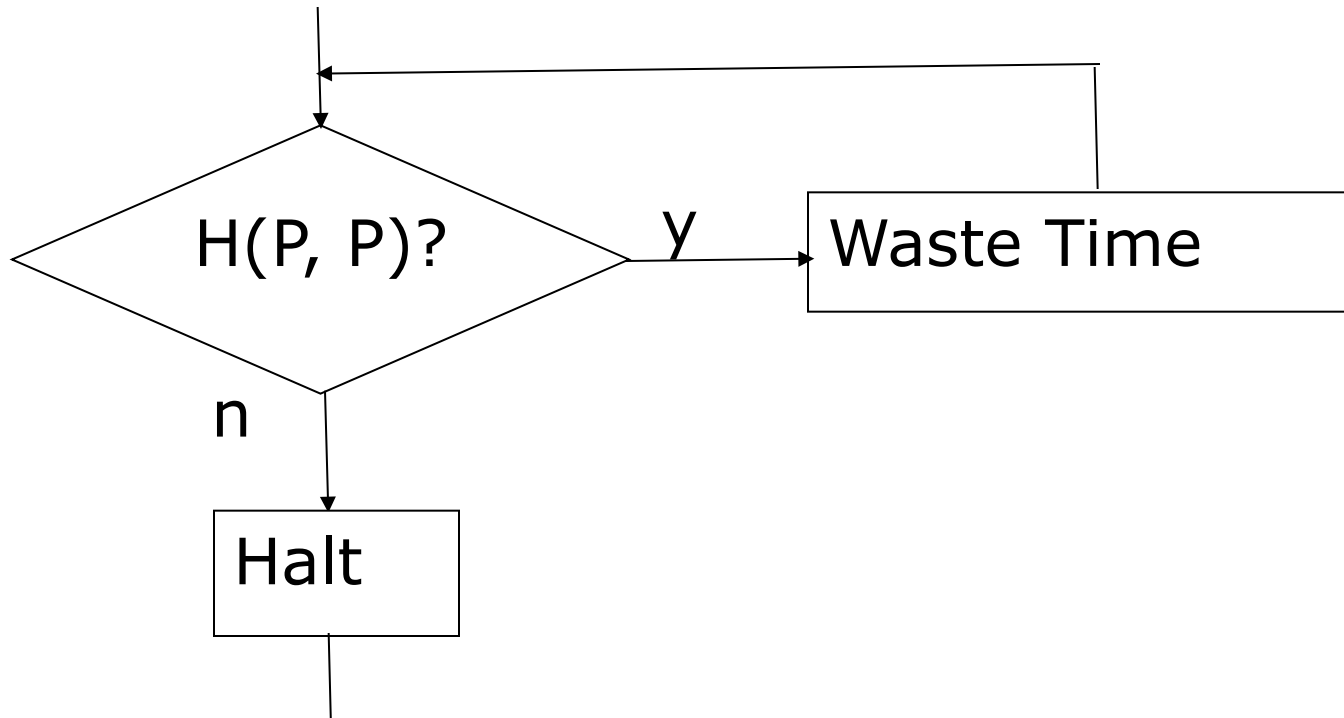
Decide, given a program  $P$  and input  $Q$  whether  $P(Q)$ , that is,  $P$  run on input  $Q$ , will eventually stop running and give an answer

- This seems pretty easy ... though running it won't work because it might not stop ... but maybe analysis could find any errors

# NOT

- The halting problem cannot be solved
- “This statement is false.” True or false?
- Here’s why ...
  - Suppose (for purposes of contradiction) that some program  $H(P, Q)$  answers the halting problem (will it halt and give an answer) for program  $P$  on data  $Q$
  - Notice the question is not, does it give the RIGHT answer ... just will it give any answer

# A New Program R(P)



An Odd Program, but what happens when we run R(R)

Two cases:

- H(R,R) says yes, it halts but by the flowchart R(R) won't halt
  - H(R,R) says no, it doesn't halt but by the flowchart R(R) halts
- Therefore, H cannot exist!

# Summary

- We have analyzed the complexity of computations, and learned ...
  - Many computations have *time proportional to  $n$*
  - Many, like sort, have running *time proportional to  $n^2$*
  - Others have running time proportional to  $n^3, n^4, \dots$
  - Some computations are computable in principle but not in practice: *NP-complete*
  - Some things cannot be computed at all, such as the *Halting Problem*

# Finding an item in a sorted list

What is the time complexity of finding an item in a sorted list?

- A. Less than linear ( $\log(n)$  or constant)
- B. Linear
- C. Polynomial but more time than linear
- D. NP (exponential like  $2^n$ )
- E. Unsolvable (like the halting problem)



# Summary

- The complexity of an algorithm gives an estimate of the running time increases as the size of the input to the algorithm increases.
- Good complexities are  $\log(n)$ , linear, or polynomial with small degree.
- Problems that have exponential times cannot be solved optimally for large inputs.
- Some problems cannot be answer (e.g. the Halting Problem).
- It is important to understand the complexity so you don't try and do the impossible.