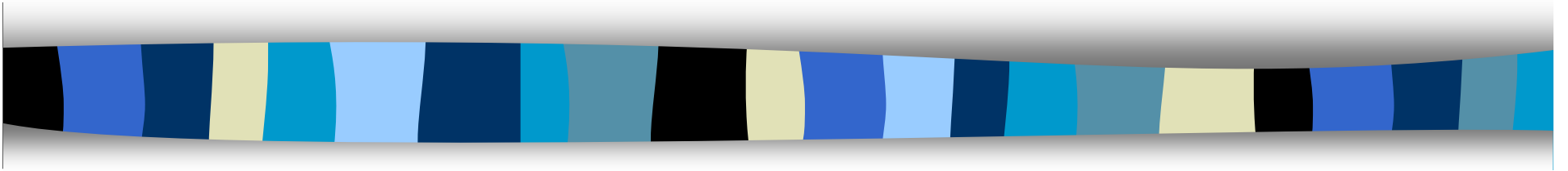


Algorithms





Problems, Algorithms, Programs

- Problem - a well defined task.
 - Sort a list of numbers.
 - Find a particular item in a list.
 - Find a winning chess move.



Algorithms

- A series of precise steps, known to stop eventually, that solve a problem.
- NOT necessarily tied to computers.
- There can be many algorithms to solve the same problem.



Characteristics of an Algorithm

- Precise steps.
- Effective steps.
- Has an output.
- Terminates eventually.



Trivial Algorithm

- Computing an average:
 - Sum up all of the values.
 - Divide the sum by the number of values.



Problems vs. Algorithms vs. Programs

- There can be many algorithms that solve the same problem.
- There can be many programs that implement the same algorithm.
- We are concerned with:
 - Analyzing the difficulty of problems.
 - Finding good algorithms.
 - Analyzing the efficiency of algorithms.



Example: Search

- Search through a list of items for a particular value.
- Example:
 - Search through an array of student records for the student with ID 12345.
 - Search through an array of address records for the address of the person with last name Doe.



Linear Search

- If we are searching in a list, start at the beginning and check each element until we find the one we want or reach the end.
- Best case?
- Worst case?
- Average case?



Binary Search

- If we are searching in a sorted list, we look at the middle item and then choose which half to continue looking in.
- We continue to cut the area we are searching in half until we find the value, or there are no more values to check.
- Best case?
- Worst case?
- Average case? (A little tricky)



Binary Search: Worst Case

- Let's say the list has 1024 items and the item is the last one we check.
 - Check midpoint of 1024 items.
 - Check midpoint of upper or lower half (512).
 - Check midpoint of a half of that half (128).
 - Successive ranges we are checking have lengths 64, 32, 16, 8, 4, 2, 1.
 - How many checks was that? 10
($\log 1024 = 10$)



Binary Search

- Aside: Note that binary search only works if the data in the list are **sorted** by the field on which we're searching!



Classifying Problems

- Problems fall into two categories.
 - Computable problems can be solved using an algorithm.
 - Non-computable problems have no algorithm to solve them.
- Historical note:
 - Hilbert's questions in 1900: complete?
Consistent? Decidable?



Classifying Problems

■ Historical note:

- Hilbert posed the following questions in 1900: Is mathematics complete? Is mathematics consistent? Is every statement in mathematics decidable?
- In 1930, he thought the all 3 answers would be “yes.”
- Almost immediately, Gödel showed that no closed system can be both complete & consistent.
- By the mid-1930’s, Turing showed that the answer to the 3rd question is “no.”



Classifying Problems

- Two categories of problems:
 - Computable
 - Non-computable
- Wouldn't it be nice to know which category a problem falls into? (Topic for later in the week: this problem itself is non-computable.)



Classifying Computable Problems

■ Tractable

- There is an efficient algorithm to solve the problem.

■ Intractable

- There is an algorithm to solve the problem but there is no efficient algorithm. (This is difficult to prove.)



Examples

- Sorting: tractable.
- The traveling salesperson problem: intractable. (we think...)
- Halting Problem: non-computable.
 - (More on this later in the week.)



Measuring Efficiency

- We are (usually) concerned with the time an algorithm takes to complete.
- We often count the number of times blocks of code are executed, as a function of the size of the input.
 - Why not measure time directly?
 - Why not count the number of instructions executed?



Example Code:

```
def aFunction(array) :  
    statementA  
    statementB  
    statementC  
    for x in array:  
        statementD  
        statementE  
    return someValue
```

- If the array has N elements, this function executes $4 + (2 * N)$ statements (i.e., $2N + 4$).



Some Mathematical Background

- Let's see some examples ...



Big O

- The worst case running time, discounting constants and lower order terms.
- Example:
 - $n^3 + 2n$ is $O(n^3)$



Exchange Sort

```
def exchangeSort(array):  
    for indx1 in range(len(array)):  
        for indx2 in range(indx1, len(array)):  
            if (array[indx1] > array[indx2]):  
                swap(array, indx1, indx2)
```

- Let's work out the big O running time...



Merge Sort

- Given a list, split it into 2 equal piles.
- Then split each of these piles in half. Continue to do this until you are left with 1 element in each pile.
- Now merge piles back together in order.



Merge Sort

- An example of how the merge works:
Suppose the first half and second half of an array are sorted:
5 9 10 12 17 1 8 11 20 32
- Merge these by taking a new element from either the first or second subarray, choosing the smallest of the remaining elements.
- Big O running time?



Big O Can Be Misleading

- Big O analysis is concerned with worst case behavior.
- Sometimes we know that we are not dealing with the worst case.



Searching an Array

```
def search(array, key):  
    for x in array:  
        if x == key:  
            return key
```

- Worst case?
- Best case?

A vertical decorative bar on the left side of the slide, composed of various colored segments including shades of blue, black, yellow, and grey, arranged in a pattern that resembles a stylized flag or a data visualization element.

Algorithms Exercise...