Data Structures and Algorithms

Analysis of Algorithms I

Parts of this presentation are based on the slides of Prof. David Galles

What is an algorithm?

- Algorithm = Computer program?

What is an algorithm?

- Algorithm ≠ Computer program!
- Algorithm is a set of steps for solving a given problem

Algorithm: Selection sort

- Examine all n elements of a list, and find the smallest element
- Move this element to the front of the list
- Examine the remaining $(n - 1)$ elements, find the smallest one
- Move this element into the second position in the list
- Repeat until the list is sorted

Algorithm vs Computer Program

- Algorithm is a set of steps for solving a problem
- Computer Program is an *implementation* of the algorithm
- Different implementations of the same algorithm

Balance Puzzle (9 coins)

- There are 9 coins. 8 are good, one is counterfeit (lighter)
- You have a balance scale, that can compare the weights of two sets of coins
- Can you determine which coin is counterfeit, using the scale only 2 times?

9 Coins

- Weigh Coins 1,2,3 against 4,5,6
- What are the possible outcomes?

9 Coins

- Weigh Coins 1,2,3 against 4,5,6
- What are the possible outcomes?
  - {1,2,3} set is lighter -> counterfeit coin in {1,2,3}
  - {4,5,6} set is lighter -> counterfeit coin in {4,5,6}
  - Two sets have equal weight -> counterfeit coin in {7,8,9}

9 Coins

- Now have a set of 3 coins {A, B, C}
- One is counterfeit
- Weigh A against B. What are the possible outcomes?

9 Coins

- Now have a set of 3 coins {A, B, C}
- One is counterfeit
- Weigh A against B. What are the possible outcomes?
  - A is lighter -> A is a counterfeit coin
  - B is lighter -> B is counterfeit coin
  - A=B -> counterfeit coin is C

Classic Version: 12 coins

- The “classic” version of this problem:

  12 coins, 3 weighings,

  The counterfeit coin could be either heavy or light

Analysis of Algorithms

- Space complexity
  - How much space is required
- Time complexity
  - How much time does it take to run the algorithm

- There is often time-space tradeoff
- We will concentrate on time complexity

Running Time

- Running time depends on the input
- Best case
  - Shortest time that the algorithm will take to run
- Worst case
  - Longest possible time that the algorithm will take to run
- Average case
  - How long, on average, does the algorithm take to run

Best case/Worst case

How long does the following function take to run?

```
boolean find(int list[], int element) {
      for (i=0; i < list.length; i++) {
            if (list[i] == elem)
                  return true;
      }
      return false;
}
```

Best case/Worst case

How long does the following function take to run?

```
boolean find(int list[], int element) {
      for (i=0; i < list.length; i++) {
            if (list[i] == elem)
                  return true;
      }
      return false;
}
```

Depends on where the element is in the list:
Best Case: elem = the first element of the list
Worst Case: elem= the last element or not in the list

Measuring time efficiency

- Experimental approach
  1. Implement the algorithm
  2. Run the code with input of different sizes
  3. Record how long it took

- Bad idea. Why?

Experimental Approach

- Need to implement the algorithm / language dependent implementation
- Can run only on limited set of inputs
- How to compare two algorithms? Hard to guarantee that the same hardware and software will be used

Theoretical Approach

- Based on high-level description of the algorithms (pseudocode)
- Make assumptions about the computer model
- Can compare algorithms independent of the hardware and software environments

Assumptions: Fetching and Storing

- The time required to fetch an operand from memory is a constant, t_{fetch}
- The time required to store a result in memory is a constant, t_{store}

- Ex:

  y = x;   // has $t_{fetch} + t_{store}$ running time

Assumptions: Elementary Operations

- The time required to perform elementary arithmetic operations is constant

```
a = a + 1; // running time: 2*t_fetch + t_+ + t_store
```

$a = a + 1;$ // running time: $2*t_{fetch} + t_{+} + t_{store}$

Can make this assumption because the number of bits used to represent a value is fixed

Assumptions: Methods

- The time required to call a method and to return from a method is constant
- The time required to pass an argument to the method = the time required to store a value in memory

Assumptions: Array Subscripting

- The time to compute the address of the element a[i] is constant . Add time to:
  - compute subscript expression
  - to fetch the element at this address

Running Time

- The number of simple operations required for an input of size n
- Is a function of n

Example

- Example: compute the sum of elements of an array

```
computeSum(A, n):
  Input: An array A storing n integers.
  Output: The sum of  elements of A.
  sum ← 0                    ⟵ simple operation
  for i ← 0 to n-1 do             ⟵ n times
     sum ← sum + A[i]      ⟵     simple operation
  return sum                 ⟵      simple operation
```

- Running time is a linear function of n

Running Time

- We will concentrate on time complexity for large inputs n
- Example: Algorithms A1 and A2 solve the same problem
  - A1: time complexity is 1000*n
  - A2: time complexity is 2^n
  - Which one is faster?

Running Time

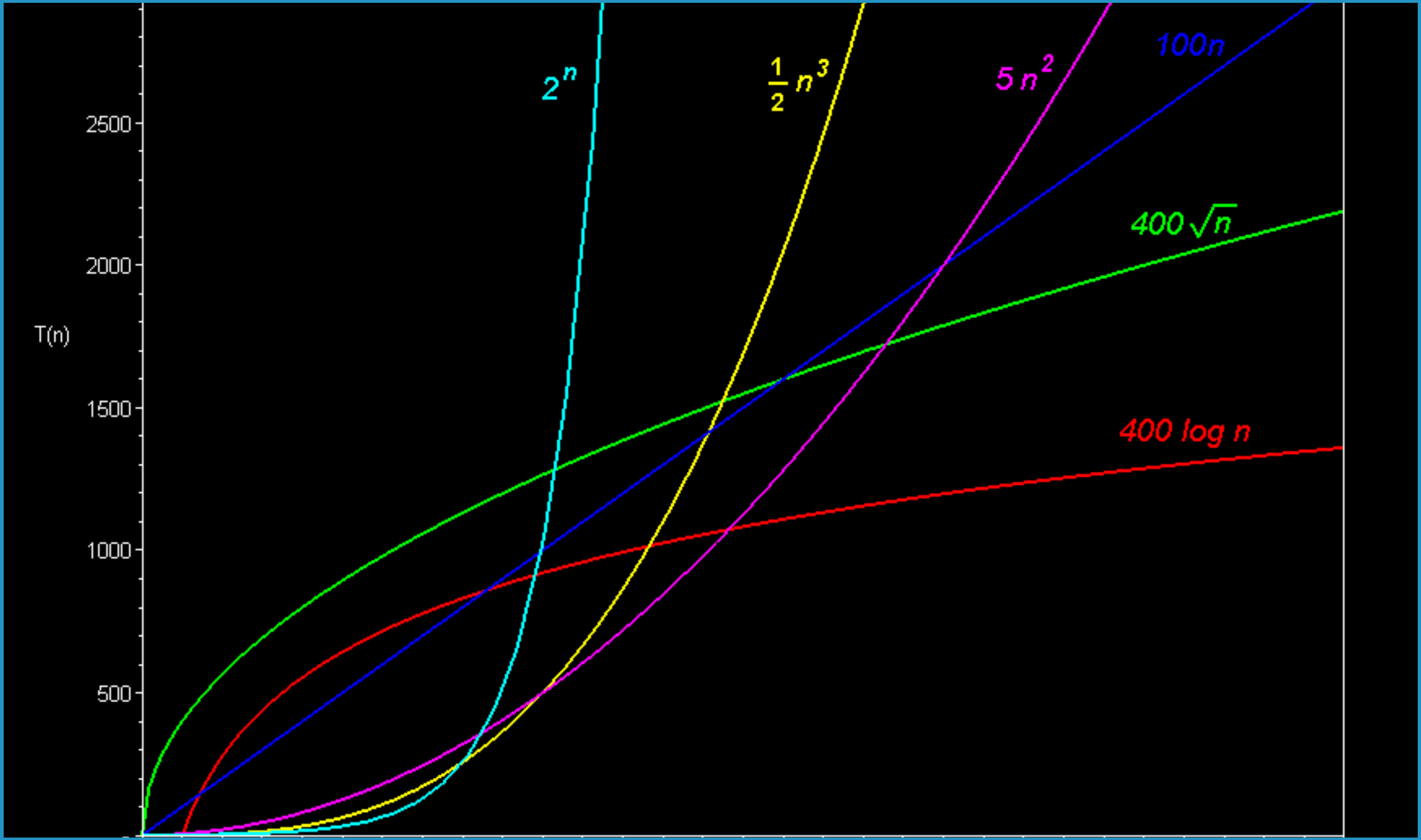
- We will concentrate on time complexity for large inputs n
- Example: Algorithms A1 and A2 solve the same problem
  - A1: time complexity is 1000*n
  - A2: time complexity is 2^n

n	A1	A2
10	10,000	1024
10^3	10^6	$\sim 10^{300}$

Mathematical Foundations

- Review plots of functions, summations

Function growth rate



http://science.slc.edu/~jmarshall/courses/2002/spring/cs50/BigO/

Logarithm Rules

$\log(a*b) = \log(a) + \log(b)$

$\log(a/b) = \log (a) – \log(b)$

$\log(a^b) = b * \log(a)$

Changing the base: $\log_a b = \log_c b / \log_c a$

Exercise

- Which of these two grows faster?

- $n^2 * \log n$ and 2^n (Hint: take the logarithm of both)
- 10^5 and $0.01*n$
- $0.1*n^3$ and $10000*\log n$
- n^{10} and 1.01^n