Algorithm Analysis

Text Readings: None

CMPT 125 / 128

Topics

1) Algorithm Efficiency & Big-O Notation

Goal based on n

- Goal:
  -
- Algorithms process a set of numbers.
  - the number of elements to process.
- Run-times are compared to mathematical functions:
  - $\log_2(n)$
  - n
  - n^2

Algorithm Speed

- Don't solve exactly how long an algorithm takes:
  - Too complicated.
  - Relies on low-level hardware details.
  - Relies on specific details of the implementation.
- So, use rough approximation to...

Example Approximate Function:

$$an^2 + bn + c$$

Big-O

- Example:
  - Assume algorithm does $an^2 + bn + c$ comparisons
- Big-O (Big-Oh) simplifications
  1. Uses only the largest term:

     For large enough *n*, the largest term dwarfs other terms.

  2. Remove constants:

     Constants represent how long it takes on a specific machine. We want an machine and language independent measure.
- It's $O(n^2)$, or "Big-O of *n* squared"

Estimating Big-O

- Estimating Process:
  -
  - How many iterations of the loop?
  - How many times is the loop called?
- Answer is relative to the size of the data set: *n*.
  - "Inner loop is called ~*n* times, each time it loops between 1 and n times, so total is $O(n^2)$"

Linear Search Implementation

```
// Find the index of the target element.
//    data:      Elements to search.
//    size:      Number of elements in data[]
//    target:    Value to find.
//    returns:   Index of target; -1 for not found.
int linearSearch (int data[], int size, int target)
{
    // Cycle through all elements
    for (int index = 0; index < size; index ++) {
        // When we find the item, return it's index.
        if (data[index] == target) {
            return index;
        }
    }
    // Item not found:
    return -1;
}
```

Total is ~*n*
Therefore it is $O(n)$

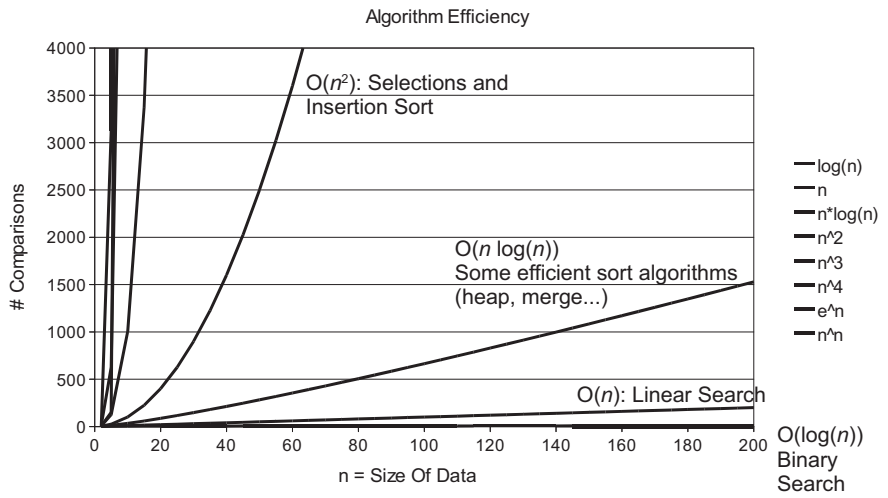
Insertion Sort Efficiency

```
void insertionSort (int data[], int size) {
    for (int index = 1; index < size; index++) {
        int key = data[index];
        int position = index;

        //  Shift larger values to the right
        while ( (position > 0)
            && (key < data[position-1]) )
        {
            data[position] = data[position-1];
            position--;
        }
        // Put the key into the hole we made
        data[position] = key;
    }
}
```

Total is ~$n*n = n^2$
Therefore it is $O(n^2)$

Why Efficiency Matters



Why Efficiency Matters

- Example:
  - 10 million elements (n=10,000,000)
  - Computer does 1 million comparisons per second.
- Run Times:
  - O(log(n)) 0.000023s
  - O(n)
  - O(n*log(n))
  - O(n^2)
  - O(n^3) (Horses evolved)
  - O(n^4) 10^{24} seconds (Universe is 10^{17} seconds old!)

Review & Summary

- What is more important:
  A faster computer, or a faster algorithm?
- Often gauge an algorithm's efficiency based on the time it takes to run.
- Big-O is a useful estimate of the rate a function grows (based on n).

Review Questions

- This material is not covered in the text.
- List 3 applications where the efficiency of an algorithm would make a significant difference.
- Graph the following functions for n=1...10,000:
  - n^2
  - $6n^2 + 2n + 5$
- Use the above graph to argue why Big-O notation is useful.