CSCI 2270 – CS 2: Data Structures

University of Colorado Boulder

University of Colorado
Boulder

Topics

- Algorithm Analysis (Big O)

Algorithm Analysis

- Choosing the right algorithm can make a huge difference!
  - Government databases
  - Search engines
  - Air Traffic Control Games
  - Medical equipment
  - Computer/video games
- We can also time algorithms, but…

Algorithm Analysis

- We can count the number of operations that the algorithm performs.
  - Count the # of instructions it performs
- The # of instructions performed may vary based on:
  - Size of the input
  - The organization of the input
- The # of instructions can be written as a cost function on the input size.

Algorithm Analysis

- Counting example

```
void printArray(int arr[], int size){
    for (int i = 0; i < size; i++){
        cout << arr[i] << endl;
    }
}
```

Operations performed on an array of length 10 = 32

Algorithm Analysis

- Let's not choose a particular input size (e.g. 10), but rather we will express a cost function for input of size *n.*
- Assume that the running time, *t*, of an algorithm is proportional to the number of operations.
- **Express *t* as a function of *n***
  - Where *t* is the time required to process the data using some algorithm *A*
  - Denote a cost function as $tA(n)$
    - ***i.e. the running time of algorithm A, with input size n***

Algorithm Analysis

- Go back to our example

```
void printArray(int arr[], int size){
   for (int i = 0; i < size; i++){
      cout << arr[i] << endl;
   }
}
```

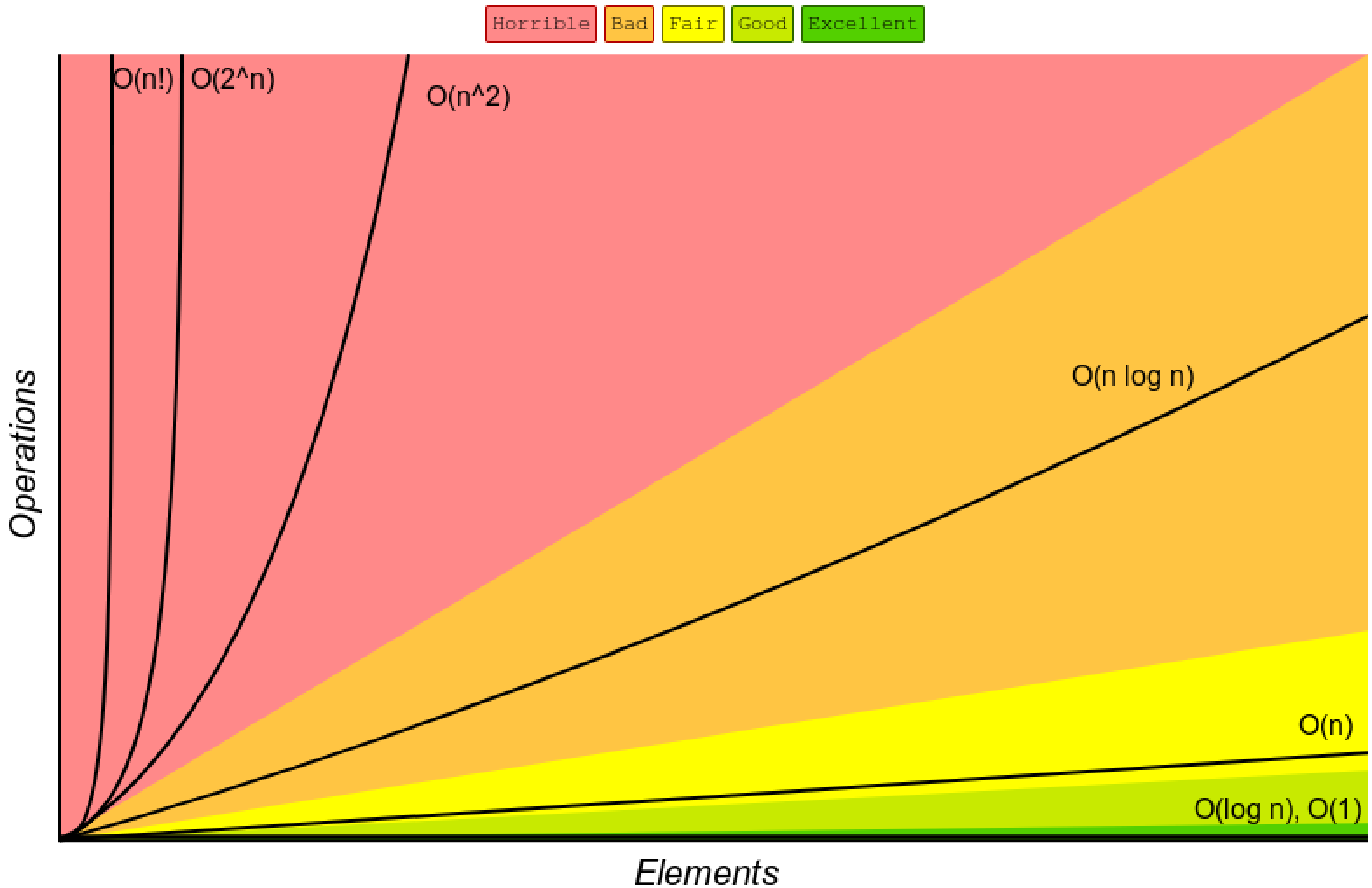
Declare and initialize *i*: 1
Comparison, print array element, increment *i*: 3n
Make comparison when $i = n$: 1

Consequently, our result is: t = 3n+2

Algorithm Analysis

- Big O Notation - is the way of measuring the efficiency of an algorithm and how well it scales based on the size of the dataset.
- O notation approximates a cost function that allows us to estimate growth rate.
  - function $f(n)$ is the algorithm's growth rate function
- Big O notation does not give a precise formulation of the cost function for a particular data size.

Algorithm Analysis



Algorithm Analysis

- $O(1)$/Constant Complexity
- $O(n)$/Linear Complexity
- $O(\log n)$/Logarithmic Complexity
- $O(n \log n)$
- $O(n^2)$/Quadratic Complexity

Algorithm Analysis

- Arrays
  - Accessing an array index: O(1)
  - Insert item to beginning or middle: O(n)
    - Time taken will be proportional to the size of the list
  - Insert/Add item to end: O(1)
    - Assumes the array has space
  - Adding in middle: O(n)
  - Deletion from beginning or middle: O(n)
    - Shifting required after removing an element
  - Deleting from the end: O(1)

Algorithm Analysis

- Linked Lists
  - Insert node to beginning: O(1)
    - Same for deletion
  - Insert in middle: O(n) (linear time)
    - Must iterate over *n* elems to get to correct location
    - Same for deletion
  - Insert node to end: O(n)
    - Must iterate over *n* elements to get to end
  - Search: O(n)
    - Slower search times than arrays because we can't use indexes. Must traverse LL!
  - Using tail pointer, adding or removing from end will be O(1)

Arrays vs Linked Lists

- Arrays have faster search time
- Arrays utilize less memory per element
  - More memory needed per node in LL because additional storage needed for pointer
- Linked Lists have faster insertion/deletion time
- Linked Lists allow for dynamic size
  - Shrink and grow
- Linked Lists are more efficient memory-wise
  - Allocation & utilization

Questions