CS112
Data Structures

Review of Big-Oh

- Efficiency of algorithms
- Sequential or Linear Search
- Asymptotic Complexity

Efficiency of Algorithms

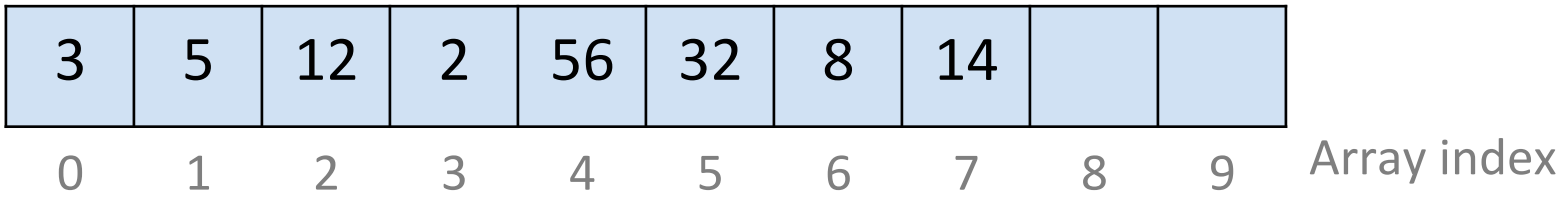
1. Identify the basic operations in an algorithm
2. Determine the running time of an algorithm by counting its basic operations
3. Express the running time of an algorithm as a function of the input size
4. Derive the big *O* order of the running time

Identifying Operations

- We use just a few primitive operations to implement algorithms
  - statements, conditionals, loops, nesting and method calls
- We assume it takes constant time to execute one statement
  - int a = 1 + 45;
  - int b = 67 * 375665;
  - System.out.println(a);
  - if (a == b)

Sequential Search

- Searching an unordered array
  - How to find a target value?
    - check each element in sequence



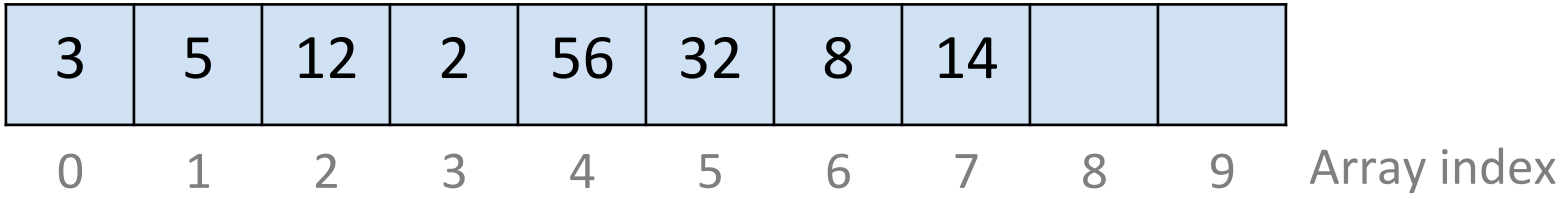

```
int SequencialSearch (int[] array, int n, int target) {
    for (int i = 0; i < n; i++) {
        if (array[i] == target) {
            return i;
        }
    }
    return -1;
}
```

Loop mechanics

Algorithm Basic Operation

Sequential Search: Efficiency Analysis

- Check each element in sequence to find the target



- Scenarios
  - Best case
    - which number is fastest to find (requires the least number of comparisons)?
  - Worst case
    - which number is the longest to find (requires the largest number of comparisons)?
  - Average case
    - average all possibilities (takes into account the probability of a possibility)

Sequential Search: Efficiency Analysis Success

Operation	Time cost	How many (success)
Initializing i	1	1
Compare i against array length	1	Best: 1 Worst: n
Incrementing i	1	Best: 0 Worst: n-1
Comparing target against array entry	1	Best: 1 Worst: n

Does not change with input length: IGNORE

Loop mechanics time:
Best: 1+0 = 1
Worst: n + n-1 = 2n-1

Algorithm time:
Best: 1
Worst: n

Running time function
Best case: f(n) = 1 + 1 = 2
Worst case: f(n) = 2n-1 + n = 3n - 1

Asymptotic Complexity

- Asymptotic analysis is a method of describing a limiting behavior

- The O notation asymptotically bounds a function
  - estimate the complexity in asymptotic sense, i.e. estimate the complexity of a function for arbitrarily large inputs

Big O

- O(f(n)) is a group of functions that
  - behave like f(n) as n gets large (have the same **growth factor**)
  - ignoring constant multiples
- O(n) includes
  - n, n*20, n+17, 5*n+log(n)
- O(n^2) includes
  - n^2, n^2+20*n-99, n^2-$n^{1/2}$

1. Express the running time as a function of the input size
2. Simplify by keeping only the fastest growing term
   - Drop constants
   - Drop insignificant terms

Sequential Search: Worst case for success

Simplify the running time function to get the order of growth

$f(n) = 3n - 1$

– drop constants and keep only the fastest growing term

$f(n) = n$

- the order of growth is linear, therefore sequential search is $O(n)$

Algorithm vs Program

More often than not, we will need to analyze the running time of an algorithm BEFORE we even implement it (we may need to explore choices, analyze each, pick best, THEN implement)

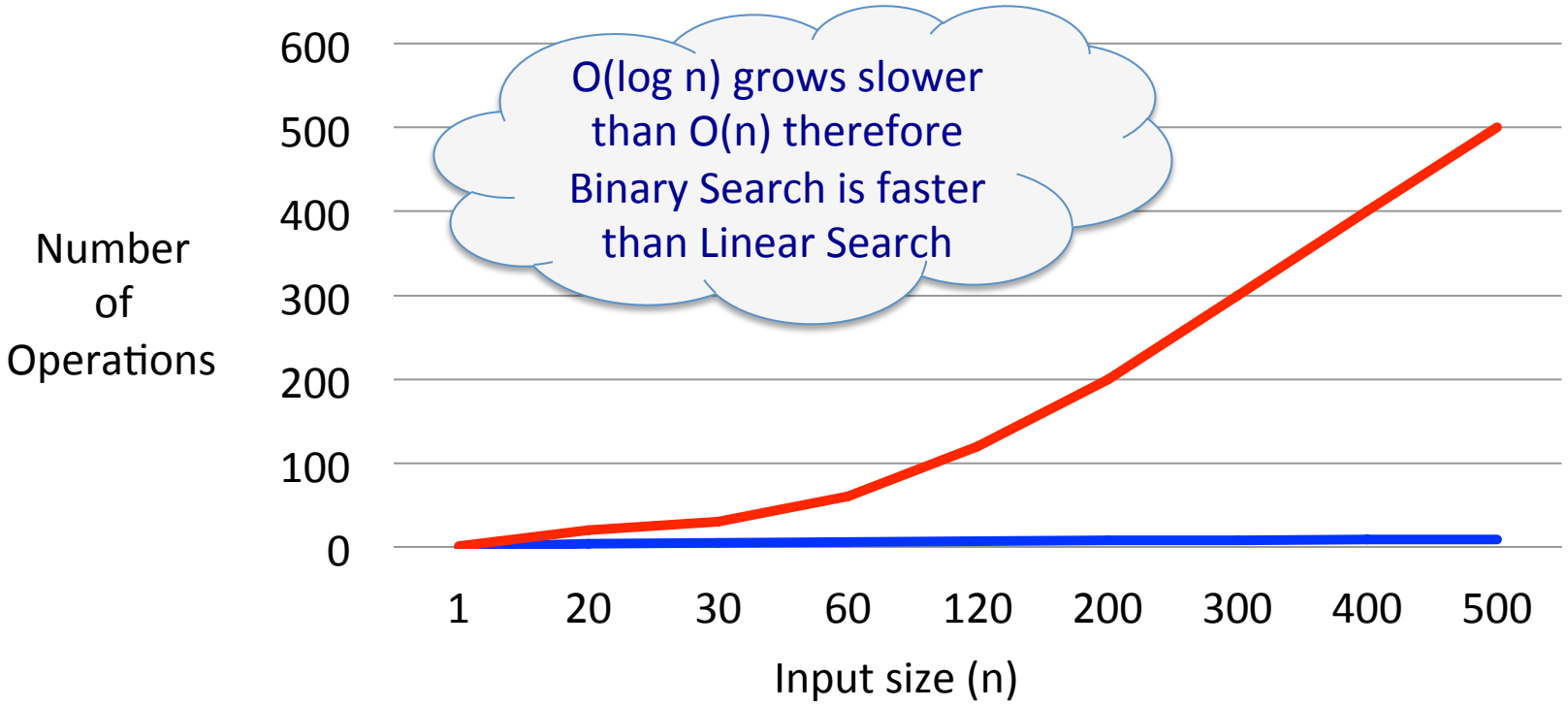
Which means we will count ONLY operations that are characteristic of the algorithm (looping mechanism is implementation specific)

Sequential Search	Running Time Units	Big Oh
Looping mechanism + Algorithm	3n-1	O(n)
Algorithm	n	O(n)

Ignoring looping mechanism does NOT change the Big O (stands to reason because looping mechanism is just supporting the execution of the algorithm, so it cannot supersede the algorithm's running time.)

Efficiency Analysis

- Linear Search: O(n) linear growth
- Binary Search: O(log n) logarithmic growth



Order-of-growth classification

description	order of growth	typical code	description	example
constant	1	`a = b + c;`	*statement*	*add two numbers*
logarithmic	log n	[just learned]	*divide in half*	*binary search*

linear	n	`double max = a[0];`
		`for (int i=1; i<n; i++)`
		`  if (a[i] > max)`
		`  max = a[i];`
			loop	*find the maximum*

logarithmic	n log n	[will learn later in CS111]	*divide and conquer*	*mergesort*

Order-of-growth classification

description	order of growth	typical code	description	example

quadratic	n^2	`for(int i=0; i<n; i++)`
		`  for(int j=i+1; j<n; j++)`
		`  if(a[i]+a[j] == 0)`
		`  cnt++`
			double loop	*check all pairs*

cubic	n^3	`for(int i=0; i<n; i++)`
		`  for(int j=i+1; j<n; j++)`
		`  for(int k=j+1; k<n; k++)`
		`  if(a[i]+a[j]+a[k] == 0)`
		`  cnt++`
			triple loop	*check all triples*

exponential	2^n	[will learn later in CS]	*exhaustive search*	*check all subsets*