

# **(9-3) Efficiency of Algorithms**

## **D & D Chapter 20**

Instructor - Andrew S. O'Fallon  
CptS 122 (March 9, 2018)  
Washington State University

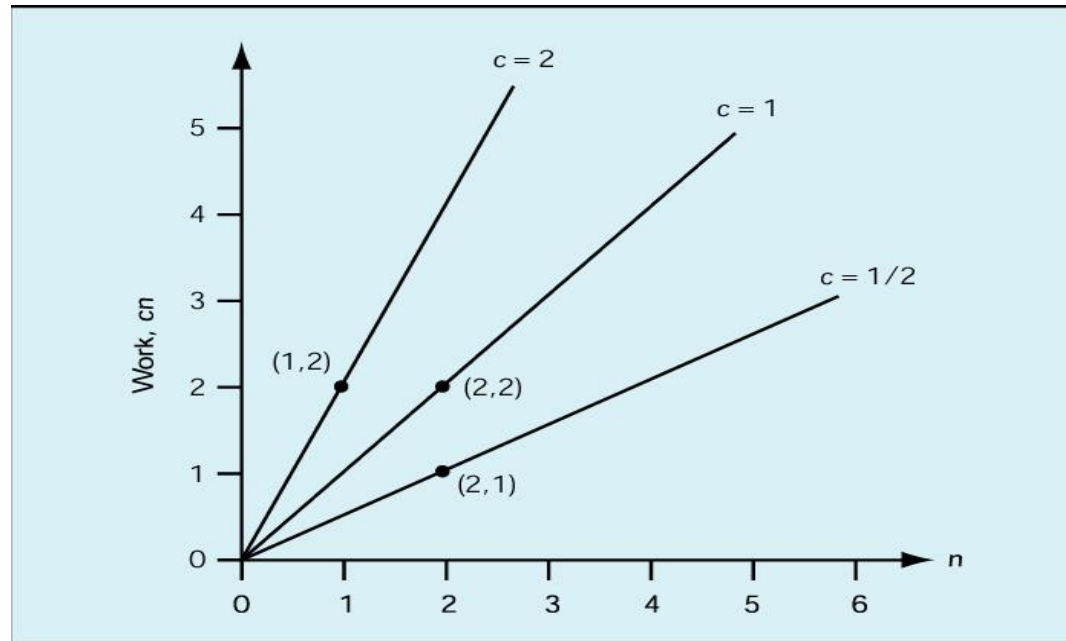
# Analysis of Algorithms (1)

- In general, we want...
  - to determine central unit of work by considering the operations applied in the algorithm
  - to express unit of work as function of size of input data: How quickly does amount of work grow as size of input grows?
  - classify algorithms according to how their running *time* and/or *space* requirements grow as input size grows
- For example, recall Sequential Search algorithm
  - Get list of  $n$  names to search, and target name to search for
  - Examine each name in sequence
    - If all names have been examined, set found to false and stop
    - If name equals target, set found to true and stop
    - If name not equal to target, advance to next name
  - Main unit of work: *comparisons*
  - Analysis
    - In best case, one comparison must be made (target is first item in list)
    - In worst case,  $n$  comparisons must be made (target not found; all items examined)
    - In average case  $n/2$  comparisons must be made



# Analysis of Algorithms (2)

- Order of magnitude analysis (“Big-O”)
  - Constant factors do not change shape of graph!



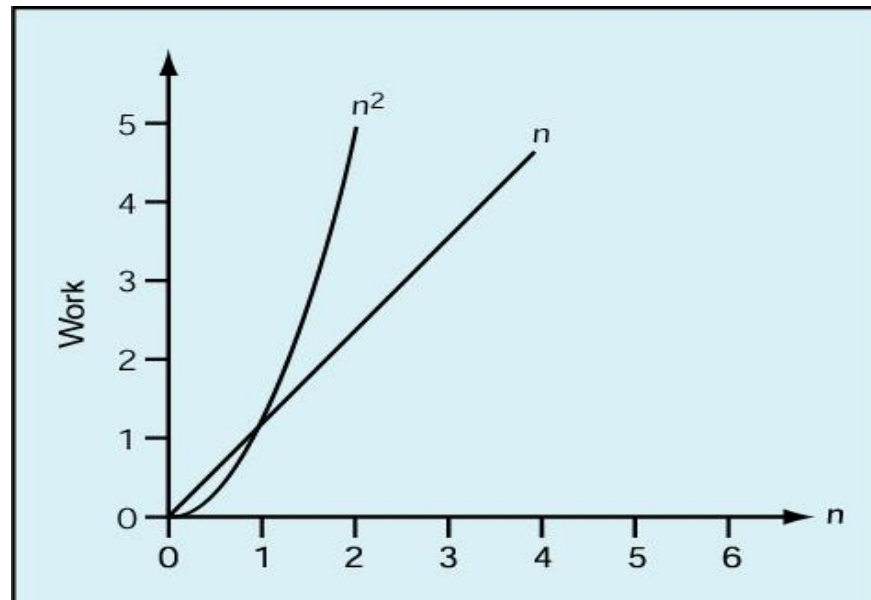
# Analysis of Algorithms (3)

- Order of magnitude (“Big-O”) (cont.)
  - Any algorithm whose work can be expressed as  $c * n$  where  $c$  is a constant and  $n$  is the input size is said to be “order of magnitude  $n$ ”, or  $O(n)$
  - Likewise, any algorithm whose work varies as a constant times the square of the input size is said to be “order of magnitude  $n$ -squared”, or  $O(n^2)$



# Analysis of Algorithms (4)

- Order of magnitude (“Big-O”) (cont.)
  - $O(n^2)$  always gets bigger than  $O(n)$  eventually!



# Analysis of Algorithms (5)

- Big-O Analysis of Sequential Search
  - Best case:  $O(1)$
  - Worst case:  $O(n)$
  - Average case:  $O(n/2) = O(n)$



# Analysis of Algorithms (6)

- Recall Selection Sort...
  - *Input*: a list of numbers
  - *Output*: a list of the same numbers in ascending order
  - *Method*:
    - Set marker that divides “unsorted” and “sorted” sections of list to the end of the list
    - While the unsorted section of the list is not empty
      - Find largest value in “unsorted” section of list
      - Swap with last value in “unsorted” section of list
      - Move marker left one position



# Analysis of Algorithms (7)

- Selection Sort (cont.)
  - Big-O Analysis
    - *Units of work*: comparisons and exchanges
    - In all cases, we need  $n + (n - 1) + \dots + 1$  comparisons =  $[n * (n - 1)]/2$  comparisons =  $1/2n^2 - 1/2n$  comparisons =  $O(n^2)$  comparisons
    - In best case, items are already in order, so 0 exchanges needed:  $O(n^2)$  comparisons + 0 exchanges =  $O(n^2)$
    - In worst case, items are in reverse order, so  $n$  exchanges needed:  $O(n^2)$  comparisons +  $n$  exchanges =  $O(n^2)$





# Analysis of Algorithms (8)

- Selection Sort (cont.)
  - Space Analysis
    - Major space requirement is list of numbers ( $n$ )
    - Other space requirements:
      - Extra memory location needed for marker between sorted and unsorted list
      - Extra memory location needed to store LargestSoFar used to find largest item in unsorted list
      - Extra memory location needed to exchange two values (why?)
      - Overall, space requirement is proportional to  $n$ .



# Analysis of Algorithms (9)

- Recall Binary Search...
  - *Input*: a list of  $n$  sorted values and a target value
  - *Output*: True if target value exists in list and location of target value, false otherwise
  - *Method*:
    - Set startindex to 1 and endindex to  $n$
    - Set found to false
    - While found is false and startindex is less than or equal to endindex
      - Set mid to midpoint between startindex and endindex
      - If target = item at mid then set found to true
      - If target < item then set endindex to mid – 1
      - If target > item then set startindex to mid + 1
    - If found = true then print “Target found at location mid”
    - Else print “Sorry, target value could not be found.”



# Analysis of Algorithms (10)

- Binary Search (cont.)
  - Big-O Analysis
    - Unit of work: comparisons
    - Best case
      - target value is at first midpoint
      - $O(1)$  comparisons
    - Worst case
      - target value is not found
      - list is cut in half until it is reduced to a list of size 0 (startindex is greater than or equal to endindex)
      - How many times can the list be cut in half? The number of times a number  $n$  is divisible by another number  $m$  is defined to be the  $\log_b(a)$ , so the answer is  $\log_2(n) = O(\lg n)$



# Analysis of Algorithms (11)

		$n$			
Order		10	50	100	1000
$\lg n$		0.0003 sec	0.0006 sec	0.0007 sec	0.001 sec
$n$		0.001 sec	0.005 sec	0.01 sec	0.1 sec
$n^2$		0.01 sec	0.25 sec	1 sec	1.67 min
$2^n$		0.1024 sec	3570 yrs	$4 * 10^{16}$ centuries	Too big to compute



# Summary of Orders of Magnitude

- $O(\lg n)$  = flying
- $O(n)$  = driving
- $O(n^2)$  = walking
- $O(n^3)$  = crawling
- $O(n^4)$  = barely moving
- $O(n^5)$  = no visible progress
- $O(2^n)$  = forget it, it will never happen



# References

- P.J. Deitel & H.M. Deitel, *C++ How to Program (9<sup>th</sup> Ed.)*, Pearson Education , Inc., 2014.
- J.R. Hanly & E.B. Koffman, *Problem Solving and Program Design in C (7<sup>th</sup> Ed.)*, Addison-Wesley, 2013



# Collaborators

- Chris Hundhausen

