CS5002 Discrete and Data Structures	Prof Higger
Fall 2020	Nov 20, 2020

HW 09 Algorithms & Function Growth

Due: Nov 30, 2020

Instructions:

- This homework exists to strengthen your understanding of concepts so that you may apply them elsewhere
- To get full credit, show intermediate steps leading to your answers.
- You are welcome to work on problems with classmates though you may not directly view another student's solution to a given problem while working together. Include a brief statement at the beginning of your homework which lists your homework group members: "Homework group: person A, person B". If you did not work with other students on the assignment write "Homework group: none". A 5 point penalty will be applied to all work which does not include this statement.
- Questions whose points are labelled with an addition sign are extra credit (e.g. "+4 points"). These are designed to push you, so have fun and don't worry if you're not making headway immediately: they're supposed to take some time. Excellence will come with practice.

Problem 1 [22 points (2 pts each)]: True False

Tell whether each of the following statements is true or false. No explanation is needed.

i $n^2 + 3 = O(n^2)$

ii $19n = O(n)$

iii $\log_2 n = O(\log_3 n)$

iv $34n^3 + 2 = \Theta(\frac{n}{1000})$

v $123 = O(n!)$

vi $n^2 = \Omega(n^2 + 4)$

vii $n^2 = \Omega(123)$

viii $3 \log_2 n = \Theta(n \log_3 n)$

ix if $f(n) = \Theta(g(n))$ then $f(n) = O(g(n))$

x if $f(n) = \Omega(g(n))$ then $f(n) = O(g(n))$

xi if $f(n) = O(g(n))$ then $g(n) = O(f(n))$

Problem 2 [22 points (8, 5, 5, 4)]: Function Growth

i Organize the following functions into six columns. Items in the same column should have the same asymptotic growth rates (big-O). If a column is to the left of another column, all its growth rates should be slower than those of the column to its right.

n^2, $n!$, $\log_2 n$, $n \log_2 n$, $3n$, $5n^2 + 3$, 2^n, 10000, $n \log_3 n$, $100n, 3log_3n$

ii Using the definition of big-O, show $100n + 5 = O(n)$. Give a particular c and n_0.

iii Using the definition of big-O, show that $n = O(2^n)$. Give a particular c and n_0.

iv Identify a function $f(n)$ which has $3n + 4n^2 + 3n! = O(f(n))$. You may use the facts that $n = O(n!)$ and $n^2 = O(n!)$.

Problem 3 [26 points (13 pts each)]: Sorting Steps

Sort the following list with the following sorts, showing all the intermediate steps.

$$18, 45, 23, 2, -5, 10, 99, 0$$

i Insertion Sort
(Please use the $\square$ symbol, as shown in the practice problems, to divide the sorted portion of the array from the unsorted portion.)

ii Merge Sort
(You need only show the pyramid diagram of how the list is broken and recombined into a sorted list).

Problem 4 [30 points (10 pts each)]: Compute Power vs Algorithm Efficiency
Moe, Larry and Curly have just purchased three new computers and use three different algorithms[1] to sort lists:

	Comparisons / sec	Search Algorithm	$T(n)$
Moe	50	Linear Search	n
Larry	5	Optimal Chunk Search	$2\sqrt{n}$
Curly	1	Binary Search	$\log_2 n$

where $T(n)$ is the number of comparisons it takes, in the worst case, to sort a list of size n. How large must n be to ensure that, for every list, ...

i Curly's computer sorts faster than Moe's?

ii Larry's computer sorts faster than Moe's?

[1]Neither section studied these algorithms, their operation isn't necessary to do this problem. However, if you're interested, a great exposition is given in Fell & Aslam's text.

iii Curly's computer sorts faster than Larry's?

Problem 5 [+5 points (1, 4)]: Bubble Sort

An array is *nearly-d sorted* if any element is not further than d spots from its sorted position. Consider the sorted list of elements:

$$X = [1, 5, 9, 10, 15, 20, 34, 57, 66, 91]$$

The same elements form a nearly-sorted list:

$$A = [1, 5, 10, 15, 9, 20, 34, 57, 91, 66]$$

with $d = 2$ because each value is, at most, 2 spots from its sorted position. Consider that the value 9 is out of order as it is in index 4[2] in A while it is in index 2 in B. Similarly,

$$B = [1, 5, 10, 9, 15, 20, 34, 57, 91, 66]$$

is nearly sorted with $d = 1$ as value 9 is in index 3 instead of index 2, 66 is in index 9 instead of index 8 and so on.

Bubble Sort[3], a sorting algorithm we have not covered, has an advantage over other methods when operating on nearly-d sorted lists.

i Describe Bubble Sort's Advantage in the best case scenario over other methods.

ii Bubble Sort need only pass through a nearly-d sorted list d times to ensure the list is sorted. Justify why this is the case. (Hint: consider the early termination condition of Bubble Sort)

[2] We adopt the Python convention of indexing $0, 1, 2, 3, \ldots$

[3] Wikipedia is a great place to start your Bubble Sort studying `https://en.wikipedia.org/wiki/Bubble_sort`, the animation in particular was instructive. However, more kinesthetic learners may appreciate the following video too: `https://www.youtube.com/watch?v=lyZQPjUT5B4`.