Problem 2-2 Correctness of Bubblesort

a.

We must also prove that the the new array is a sorted set of the original array. In this case, A', is the set of sorted elements of the original array A.

b.

The loop invariant for the inner loop in lines 2-4 asserts that the subarray A[j... n] contains the elements of the original array, but in a potentially sorted order and the first element will be the smallest.

Proof

The loop invariant must hold for initialization, maintenance, & termination.

Initialization:

The sub array contains the smallest element A[n]

Maintenance

The length of the sub array will increase by one on every iteration. Each iteration will compare A[j] and A[j-1] and will swap them if A[j] < A[j-1] thus putting the smaller element first.

Termination

The loop must terminate. In this case, it terminates when $j = i + 1$. At this point, the first element is the smallest & the last is the largest.

c. The loop invariant for lines 1-4 asserts that each iteration will start with a sub array of elements $A[1 \dots i-1]$ that must be smaller than the elements in the subarray $A[1 \dots n]$ and will be sorted.

Proof

The loop invariant must hold for initialization, maintenance, & termination.

Initialization

Prior to the first iteration, the sub array $A[1 \dots i-1]$ is empty.

Maintenance

When the inner loop finishes, the smallest element of the subarray $A[1\dots n]$ will be $A[i]$. Each iteration, the outer loop will contain a subarray of elements $A[1\dots i-1]$ that are smaller than the elements in $A[1\dots n]$ and sorted. The subarray $A[1\dots i]$ will contain elements that are smaller than $A[i+1\dots n]$ in sorted order after each iteration.

Termination

The loop must terminate. This loop terminates when i equals the length of the array A. When it terminates, all the elements of A will be in a sorted order.

d.

The worst case for bubblesort is an array sorted in reverse.

- Each iteration, we will have to execute n number of cheeks and swaps and thus the worst case $\Theta(n^2)$.

- Insertion sort's worst case is also $\Theta(n^2)$, though, the number of cheeks and swaps is far fewer. The difference in time complexity is with the constant terms and therefore insertion sort will sort slightly faster than bubblesort.

Problem 2-4 Inversions

a.

- $(3,4)$
- $(1,5)$
- $(2,5)$
- $(3,5)$
- $(4,5)$

b.

An array in reverse sorted order will have the most inversions.

The first index will have $n-1$ inversions & the second will have $n-2$ inversions.

Therefore, $(n-1)+(n-2)+(n-3)+\dots+1$ can be simplified to $\frac{n(n-1)}{2}$.

c.

The number of inversions in the array increases the number of times that the inner loop will run.

We know this is true because in the last problem we showed that a reverse sorted list has the most inversions. This also explains why a reverse sorted list takes the longest to sort.

reverse sorted list → more (most) inversions → more operations → longer running time

Therefore, inversions negatively impact the running time of insertion sort.

d.

We already know we are going to divide and conquor when we see the "lg n" part. The solution is almost the same as merge sort (as suggested). We will divide the arrays in half recursively and then have to count the number of inversions each time. This means that we will perform n operations on lg n steps. Therefore, the running time of said algorithm is $\Theta(n \lg n)$.

Problem 3-3 Ordering asymptotic growth rates

a. This list is in reverse order where the highest growth rate is in the bottom row far right.

This way the problem is satisfied such that $f(n) = \Theta(g(n))$ where g_1 is $2^{2^{n+1}}$ & g_2 is 2^{2^n}..... so on and so forth.

Slowest growth rates start here

1	$\frac{1}{n^{\lg n}}$	$\lg(\lg^* n)$	$\lg^*(\lg n)$	$\lg^* n$
$2^{lg^* n}$	$\ln \ln n$	$\sqrt{\lg n}$	$\ln n$	$\lg^2 n$
$2^{\sqrt{2 \lg n}}$	$(\sqrt{2})^{\lg n}$	$2^{\lg n}$	n	$\lg(n!)$
$n \lg n$	n^2	$4^{\lg n}$	n^3	$(\lg n)!$
$n^{\lg \lg n}$	$(\lg n)^{\lg n}$	$\left(\frac{3}{2}\right)^n$	2^n	$n \cdot 2^n$
e^n	$n!$	$(n+1)!$	2^{2^n}	$2^{2^{n+1}}$

b. Find a f(n) such that

$f(n) > O(g(n))$ and $f(n) < \Omega(g(n))$

We know $2^{2^{n+2}}$ is $>$ all $g_i(n)$

and $1/n$ is $<$ all $g_i(n)$.

We can define a function

$$f(n) = \begin{cases} 2^{2^{n+2}} & n \text{ is even} \\ 1/n & n \text{ is odd} \end{cases}$$

This way f(n) growth rate is larger and smaller than all other functions $g_i(n)$ and satisfies all the criteria in this problem.

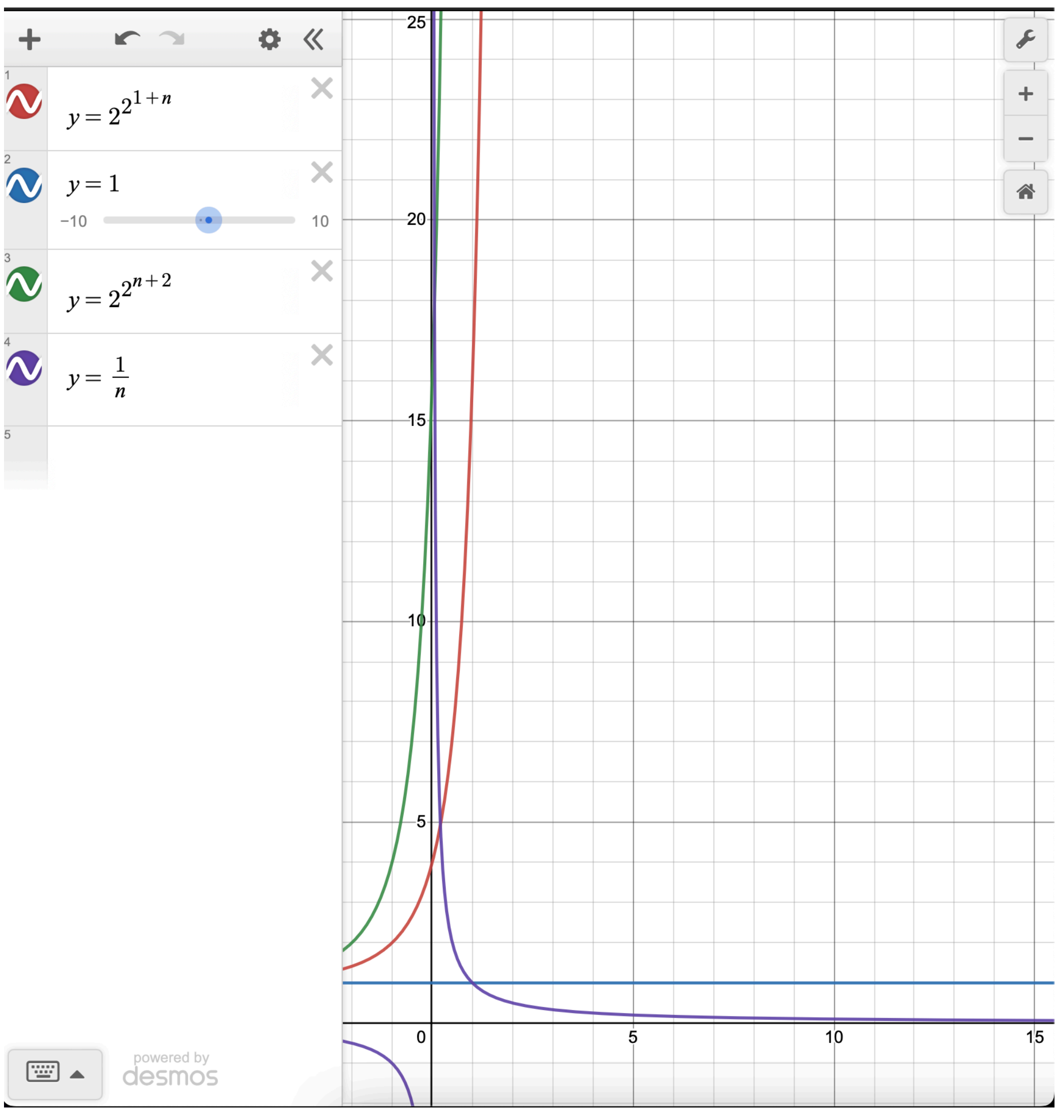
I made the following graph to show the growth rates and to show how the new f(n) compares to all of the $g_i(n)$.

fastest $g_i(n)$

slowest $g_i(n)$

faster f(n)

slower f(n)



Problem 3-7 Iterated Functions

	$f(n)$	c	$f_c^*(n)$
a	$n-1$	0	$\theta(n)$
b	$\lg n$	1	$\theta(\lg(n))$
c	$n/2$	1	$\theta(\lg(n))$
D	$n/2$	2	$\theta(\lg(n))$
e	$\sqrt{n}$	2	$\theta(\lg(\lg(n)))$
f	$\sqrt{n}$	1	undefined
g	$n^{1/3}$	2	$\theta(\log_3(\lg(n)))$
h	$n/\lg n$	2	$\theta(\lg(\lg(n))/\lg(n))$