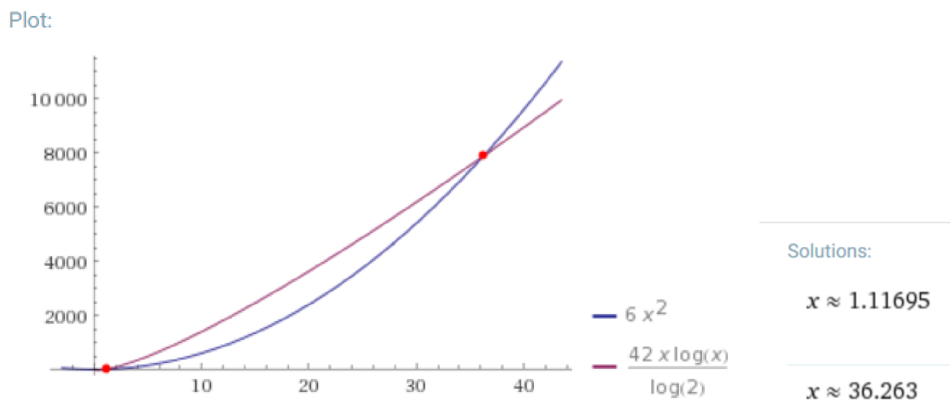
CS 325 - HW 1 Solutions Examples

Problem	1	2	3	4	5	6
Points	2	4	5	4	4	6

Since some problems have multiple right answers I have compiled some possible correct solutions submitted by students.

Problem 1: 2 points - must include either a graph, table or some explanation as to how they got the result n <=36 insertion sort beats merge sort (or merge sort beats insertion when n>=37)



Or

n	Insertion Sort	Merge Sort
1	6	0
2	24	84
4	96	336
8	384	1008
16	1536	2688
32	6144	6720
36	7776	7816.926602
37	8214	8095.49053
64	24576	16128

CS 325 - HW 1 Solutions Examples

Problem 2: 4 points total- (deduct 0.5 for very minor errors)

4 points = Base case, inductive hypothesis and induction correct

3 points = One of the above incorrect

2 point = Two of the above incorrect

1 point = effort points only

Use mathematical induction to show that when n is an exact power of 2, the solution of the recurrence

$$T(n) = \begin{cases} 2, & if\ n = 2 \\ 2T\left(\frac{n}{2}\right) + n, & if\ n = 2^k, \text{for } k > 1 \end{cases}$$

is $T(n) = n \lg n$.

See sample solutions below. You can assume that n is a power of 2.

Sample Solution 1

Base Case: n = 2 and we have T(2) = 2lg(2) = 2

For the inductive hypothesis assume that T(n/2) = (n/2)lg(n/2).

Then

T(n) = 2T(n/2) + n
= 2(n/2)lg(n/2) + n
= nlg(n/2) + n
=n(lgn – lg2) + n
= n(lgn – 1) + n
= nlgn – n + n
= nlgn

Therefore T(n) = nlgn which completes the inductive proof when n is an exact power of 2.

Sample Solution 2

Let n = 2^k

Base Case: k = 1 which implies that n = 2^1= 2 and we have T(2) = 2lg(2) = 2

For the inductive hypothesis assume true when n=2^{k-1} that T(2^{k-1}) = (2^{k-1})lg(2^{k-1}).

Then when n = 2^k

T(n) = T(2^k)
= 2T(2^k /2) + 2^k
= 2T(2^{k-1}) + 2^k
= 2(2^{k-1})lg(2^{k-1}) + 2^k
= (2^k)lg(2^{k-1}) + 2^k
= (2^k)lg(2^{k-1}) + 2^k
=2^k (lg(2^{k-1}) + 1)
=2^k (lg(2^k) - 1 + 1)
=2^k (lg(2^k))

= nlgn

Therefore T(n) = nlgn which completes the inductive proof when n is an exact power of 2.

Sample Solution 3

Base Case:

n = 2, then T(2) = 2 and T(2) = 2*lg(2) = 2

Hypothesis:

Assume T(n) = n * lg(n) for $n = 2^k$ where $k > 1$. We need to show that

$$T(2^{k+1}) = 2^{k+1} * \lg(2^{k+1})$$

Induction:

$$Let\ n = 2^{k+1}$$

$$T(2^{k+1}) = 2T\left(\frac{2^{(k+1)}}{2}\right) + 2^{k+1}$$

$$= 2 * T\left(\frac{2^k * 2^1}{2}\right) + 2^{k+1}$$

$$= 2 * T(2^k) + 2^{k+1}$$

$$= 2 * [2^k * \lg(2^k)] + 2^{k+1}$$

$$= 2^{k+1} * \lg(2^k) + 2^{k+1}$$

$$= 2^{k+1} * (\lg(2^k) + 1)$$

$$T(2^{k+1}) = 2^{k+1} * \lg(2^{k+1})$$

This is the same form as $T(n) = n * \lg(n)\ \ for\ n = 2^{k+1}$. Therefore when n is an exact power of 2, T(n) = n*log(n).

CS 325 - HW 1 Solutions Examples

3. **5 points - 0.5 point deduction for each one missed.**

	f(n)	g(n)	Relationship	Explanation
a.	$f(n) = n^{0.25}$;	$g(n) = n^{0.5}$	$f(n)$ is $O(g(n))$	$\lim\limits_{n\to\infty} \frac{n^{0.25}}{n^{0.5}} = 0$
b.	$f(n) = n$;	$g(n) = \log^2 n = (\log_{10} n)(\log_{10} n)$	$f(n)$ is $\Omega(g(n))$	Applying l'Hopital rule, $\lim_{n\to\infty} \frac{f'(n)}{g'(n)} = \frac{1}{\frac{1}{x}*\frac{1}{x}} = x^2 = \infty$,
c.	$f(n) = \log n$;	$g(n) = \ln n$	$f(n) = \Theta(g(n))$	$\lim_{n\to\infty} \frac{\log_{10} n}{\log_e n} = \frac{\log_{10} n}{\log_{10} n / \log_{10} e} = \log_{10} e$ which is a constant > 0
d.	$f(n) = 1000n^2$;	$g(n) = 0.0002n^2 - 1000n$	$f(n) = \Theta(g(n))$	Since lower order terms are not significant, and the ratio of the higher order terms is a constant > 0
e.	$f(n) = n\log n$;	$g(n) = n\sqrt{n}$	$f(n)$ is $O(g(n))$	$f(n)/g(n) = n\log(n)/n\sqrt{n} = \log(n)/\sqrt{n}$, so applying the l'Hôpital rule $\lim_{n\to\infty} \frac{f'(n)}{g'(n)} = \lim\limits_{n\to\infty} \frac{\frac{1}{n}}{0.5n^{-0.5}} = \lim\limits_{n\to\infty} 2/n^{0.5} = 0$
f.	$f(n) = e^n$;	$g(n) = 3^n$	$f(n)$ is $O(g(n))$	$\lim_{n\to\infty} \frac{e^n}{3^n} = \lim_{n\to\infty} \left(\frac{e}{3}\right)^n = \lim_{n\to\infty} n = \infty$ since $\left(\frac{e}{3}\right)^n$ is a continuous function of n.
g.	$f(n) = 2^n$;	$g(n) = 2^{n+1}$	$f(n) = \Theta(g(n))$	$f(n)/g(n) = \frac{2^n}{2^{n+1}} = \frac{2^n}{2*2^n}$, so the $\lim_{n\to\infty} \frac{2^n}{2*2^n} = \frac{1}{2}$ which is a constant $>$ 0
h.	$f(n) = 2^n$;	$g(n) = 2^{2^n}$	$f(n)$ is $O(g(n))$	$\frac{f(n)}{g(n)} = \frac{2^n}{2^{2^n}} = 2^{n-2^n}$, so $\lim_{n\to\infty} 2^{n-2^n} = 0$ because the -2^n term grows fastest.
i.	$f(n) = 2^n$;	$g(n) = n!$	$f(n)$ is $O(g(n))$	$\lim_{n\to\infty} \frac{2^n}{n!} = 0$ because n! grows faster than 2^n. For a given n the factors of 2^n are all equal to 2, but the factors of n! range from 1 to n.
j.	$f(n) = \lg n$;	$g(n) = \sqrt{n}$	$f(n)$ is $O(g(n))$	applying the l'Hôpital rule $\lim_{n\to\infty} \frac{\lg n}{\sqrt{n}} = \lim\limits_{n\to\infty} \frac{\frac{\lg e}{n}}{0.5n^{-0.5}} = \lim\limits_{n\to\infty} \frac{2\lg e}{n^{0.5}} = 0$ because the numerator is a constant and the denominator goes to infinity.

CS 325 - HW 1 Solutions Examples

4) 4 points total- There are several variations of the algorithm

    1 points = Verbal explanation of algorithm with in less than 1.5 comparison

    1 points = Pseudocode

    1 points = show comparisons is 1.5n

    1 points = Demonstrated on example data

Samples of algorithms that received full credit are below.

CS 325 - HW 1 Solutions Examples

```
Input:  An array L containing n > 0 values.
Output:  The minimum and maximum values in the array.
Description: Compares pairs of numbers sequentially, comparing the smaller value of each pair
to the global minimum and the larger value of each pair to the global maximum.

MIN-MAX(L)

// Initialize variables
min = L[0]
max = L[0]

// Account for odd or even number of values
lo = L.length mod 2
hi = floor(L.length/2)*2

// Find min and max in 3n/2 comparisons
for i = lo to hi by 2
    if L[i] < L[i+1]
        if L[i] < min
            min = L[i]
        if L[i+1] > max
            max = L[i+1]
    else
        if L[i+1] < min
            min = L[i+1]
        if L[i] > max
            max = L[i]

return min, max
```

The for-loop performs at most n/2 loops if n is even, or (n-1)/2 if n is odd. For each pass of the loop the if-else construct performs one comparison, and the nested if-statements perform 2 additional comparisons for a total of 3 comparisons per loop. So the total number of comparisons is less than 3n/2, or 1.5n.
The following table demonstrates the execution of the algorithm with the input L = [9, 3, 5, 10, 1, 7, 12]. The first row shows the state of variables after the variables are initialized, and then at the end of each pass of the loop.

[9, 3, 5, 10, 1, 7, 12]	i = 0	min = 9	max = 9	min and max both set to L[0]
[9, 3, 5, 10, 1, 7, 12]	i = 1	min = 3	max = 5	3 comparisons in if block of if-else
[9, 3, 5, 10, 1, 7, 12]	i = 3	min = 1	max = 10	3 comparisons in else block of if-else
[9, 3, 5, 10, 1, 7, 12]	i = 5	min = 1	max = 12	3 comparisons in if block of if-else

There are 7 values in L. Since L.length is odd, we do a total of 9 comparisons, which is less than 1.5 * 7 = 10 (rounding down). The min =1 and max = 12, which is what we expect.

CS 325 - HW 1 Solutions Examples

Part II: A more efficient way of doing this is blocking the array out into pairs and comparing pairs with each other iteratively. One rather clunky aspect of this method is that the algorithm has to handle sets with odd quantities differently than sets with even quantities. Also note that the algorithm does not support lists with less than 2 values. When the quantity of the list is even, the comparisons are $1 + \frac{3n-2}{2}$, which simplifies out to $\underline{1.5n}$. For odd quantities, the comparisons are $\frac{3n-1}{2}$, which simplifies out to $\underline{1.5n - .5}$.

```
betterMinMax(Array[0...n])
    if Array has even quantity
        if Array[0] > Array[1]

            minimum = Array[1]
            maximum = Array[0]
        else
            minimum = Array[0]
            maximum = Array[1]
        counter = 2

    else
        minimum = Array[0]
        maximum = Array[0]
        counter = 1

    while(counter < length of Array )
        if (Array[counter ] > Array[counter + 1])
            if (Array[counter] > maximum)
                maximum  = Array[counter]
            if (Array[counter+1] < minimum)
                minimum  = Array[counter+1]

        else
            if (Array[counter+1] > maximum)
                maximum  = Array[counter+1]
            if (Array[counter] < minimum)
                minimum  = Array[counter]
        counter = counter + 2
```

CS 325 - HW 1 Solutions Examples

Part III: For the input A = [9, 3, 5, 10, 1, 7, 12], which has an odd number of elements, the algorithm executes as follows:

Step 1 – set initial values in first else:

minimum = 9

maximum = 9

counter = 1

Comparisons Count = 0

Step 2 – first pass through while loop:

A[1] < A[2], so we go to the else statement

A[2] is not greater than *maximum*, so nothing changes

A[1] is less than *minimum,* so *minimum* = 3

counter = 3

Comparisons Count Afterward = 3

Step 3 – second pass through while loop:

A[3] >A[4], so we stay in the if statement

A[3] > *maximum,* so *maximum* = 10

A[4] < *minimum,* so *minimum* = 1

counter = 5

Comparisons Count Afterward = 6

Step 4 – third pass through while loop:

A[5] < A[6], so we go to the else statement

A[6] > *maximum,* so *maximum* = 12

A[5] is not less than *minimum*, so nothing happens

Comparisons Count Afterward = 9

So, with a list of 7 numbers and a total of 9 comparisons, we can easily see that this algorithm beats the goal of 1.5*n* comparisons, as 7x1.5 = 10.5.

Problem 4 example:

A revised algorithm sets the initial maximum and minimum values to the first value of the input array. If there is an even number of array values, our for-loop index will begin at item 1, and if there is an odd number of values, our for-loop index will begin at item 2 (to prevent exceeding the bounds of the array in the loop block comparisons). This for-loop compares two array items to one another. The greater of the two is then compared with the current maximum, which is replaced if the item is greater than it, and the lesser of the two is compared with the current minimum, which is replaced if the item is less than it.

In the worst case, this algorithm performs 3 comparisons in each loop iteration, and goes through the loop $\lfloor \frac{n}{2} \rfloor$ times. This gives a total of $3 \cdot \lfloor \frac{n}{2} \rfloor$ comparisons in the worst case.

CS 325 - HW 1 Solutions Examples

Algorithm 2 Revised Min-Max Algorithm

```
function minMax(A)
    minimum = maximum = A[1]
    index = 2
    if A.length  mod 2 == 0  then
        start = 1
    end if
    for i = start to A.length by 2 do
        if A[i] > A[i + 1] then
            if A[i] > maximum then
                maximum = A[i]
            end if
            if A[i + 1] < minimum then
                minimum = A[i + 1]
            end if
        else
            if A[i + 1] > maximum then
                maximum = A[i + 1]
            end if
            if A[i] < minimum then
                minimum = A[i]
            end if
        end if
    end for
    return minimum, maximum
end function
```

CS 325 - HW 1 Solutions Examples

5) a. **2 points: 0.5 for false (disprove), 1.5 for counter example**

If $f_1(n) = O(g(n))$ and $f_2(n) = O(g(n))$ then $f_1(n) = \Theta(f_2(n))$.

Example: $f_1(n) = n$, $f_2(n) = n^2$ *and* $g(n) = n^3$ then $f_1(n) \neq \Theta(f_2(n))$.

b. **2 points: 0.5 for true (prove), 1.5 for correct proof.**

If $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$ then $f_1(n) + f_2(n) = O(\max\{g_1(n), g_2(n)\})$

By definition there exists a $c_1, c_2, n_1, n_2 > 0$ such that

$f_1(n) \leq c_1 g_1(n))$ for $n \geq n_1$ and $f_2(n) \leq c_2 g_2(n)$ for $n \geq n_2$

Since the functions are asymptotically positive

$$\begin{aligned} f_1(n) + f_2(n) &\leq c_1 g_1(n) + c_2 g_2(n) \\ &\leq c_1 \max\{g_1(n), g_2(n)\} + c_2 \max\{g_1(n), g_2(n)\} \\ &\leq (c_1 + c_2) \max\{g_1(n), g_2(n)\} \end{aligned}$$

Let $k = (c_1 + c_2)$ and $n_0 = \max(n_1, n_2)$ then **(-0.5 if n_0 is missing)**

$f_1(n) \cdot f_2(n) \leq k \max\{g_1(n), g_2(n)\}$ for $n \geq n_0$; $k, n_0 > 0$ and by definition

$f_1(n) + f_2(n) = O(\max\{g_1(n), g_2(n)\})$

CS 325 - HW 1 Solutions Examples

7) **6 points total-**

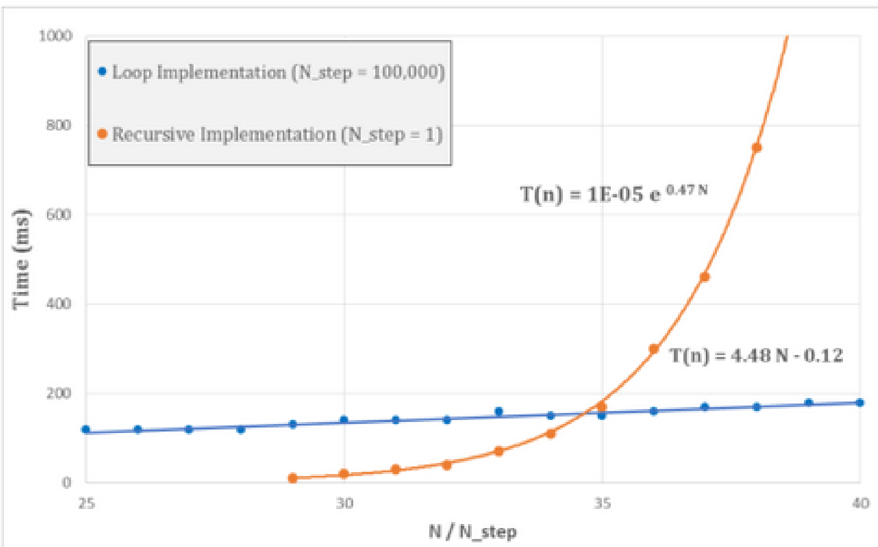
a) 2 points - iterative code and recursive code

b) & c) 2 points - Plot the running time data you collected on graphs with n on the x-axis and time on the y-axis. (-1 if times are all 0)

d) 2 points What type of function (curve) best fits each data set? Iterative is linear. The recursive is exponential. Fitted curves may vary

The following example of a graph would get full credit for parts c and d.

Fibonacci Performance Comparison:



The time vs. N data w as model fitted in excel for both loop and recursive implementation of the fibbonacci sequence. The loop implementation yielded a linear best fit while the recursive implementation showed time to have an exponential depence with N. The loop implementation was able to calculate 400,000 fibonnaci numbers in 200 miliseconds while the recursive implmentation took minutes. The reason for the exponential time increase in the recursive implementation is that there are 2 recursive calls as the return value of the recursive function.

Loop implmentation: $T(N) = 4.48 * N - 0.12 \Rightarrow \; O(N)$

Recursive Implementation: $T(N) = 1 * 10^{-5} e^{0.47N} \;\Rightarrow\; \Theta(2^n)$