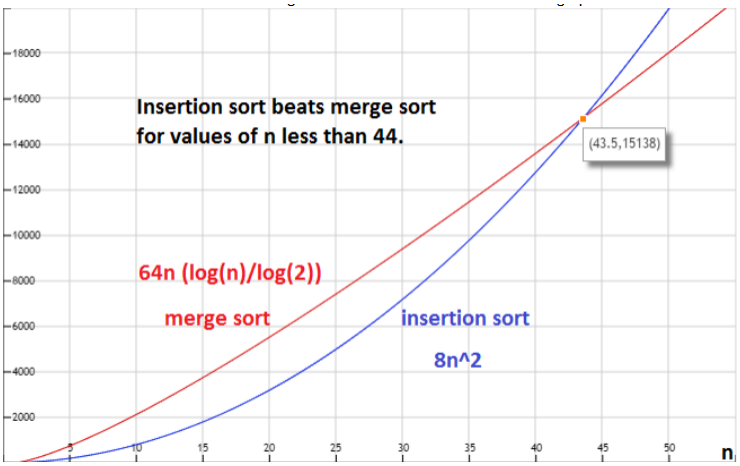
CS 325 - HW 1 Solutions Examples

Problem	1	2	3	4	5	6	7
Points	1	NOT GRADED	3	2	3	5	6

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

Problem 1: 1 point - must include either a graph, table or some explanation as to how they got the result n < 44.



To solve this, I made a simple spreadsheet that quickly found that insertion sort above will beat for values of n < 44.

value of n	8n^2	64nlgn
1	8	0
2	32	128
3	72	304
5	200	743
10	800	2126
20	3200	5532
40	12800	13624
43	14792	14933
44	15488	15374
45	16200	15817
50	20000	18060
100	80000	42521
1000	8000000	637810

CS 325 - HW 1 Solutions Examples

Problem 2: Not Graded

	1 second	1 minute	1 hour	1 day	1 month	1 year	1 century
lg n	2^10^6	2^6x10^7	2^36x10^8	2^864x10^8	2^2592x10^9	2^3154x10^10	2^3154x10^12
sqrt n	10^12	36x10^14	1296x10^16	749496x10^1	671846x10^18	994519296x10^18	995827586973696x10^16
n	10^6	6x10^7	36x10^8	864x10^8	2592x10^9	3154x10^10	3154x10^12
n lg n	62746	2801417	133378058	2755147513	71870856404	7.97634E+11	6.86547E+13
n^2	1000	7746	60000	293939	1609969	5616048	56160484
n^3	100	391	1533	4421	13737	31595	146652
2^n	19	25	31	36	41	44	51
n!	9	11	12	13	15	16	17
micro seconds	1.00E+06	6.00E+07	3.60E+09	8.64E+10	2.592E+12	3.154E+13	3.154E+15

4)	1 sec.	1 min.	1 hr.	1 day	1 month	1 year	1 century
lg n	2^{10^6}	$2^{6*(10^7)}$	$2^{36*(10^8)}$	$2^{864*(10^8)}$	$2^{2592*(10^9)}$	$2^{31536*(10^9)}$	$2^{31536*(10^{11})}$
sqrt n	10^{12}	$36(10^{14})$	$1296(10^{16})$	$746496(10^{16})$	$6718464(10^{18})$	$9.945193(10^{26})$	$9.945193(10^{30})$
n	10^6	$6(10^7)$	$36(10^8)$	$864(10^8)$	$2592(10^9)$	$31536(10^9)$	$31536(10^{11})$
n lg n	62746	2801417	$13.33(10^7)$	$27.55(10^8)$	$71.87(10^9)$	$79.76(10^{10})$	$68.61(10^{12})$
n^2	1000	7745	60000	293938	1609968	5615692	56156922
n^3	100	391	1532	4420	13737	31594	146646
2^n	19	25	31	36	41	44	51
n!	9	11	12	13	15	16	17

CS 325 - HW 1 Solutions Examples

Problem 3: 3 points total-

3 points = Base case, inductive hypothesis and induction correct

2 points = One of the above incorrect

1 point = Two of the above incorrect

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

The base case is when $n = 2$, and we have $n \lg n = 2 \lg 2 = 2 \cdot 1 = 2$.

For the inductive step, our inductive hypothesis is that $T(n/2) = (n/2)\lg(n/2)$.
Then

$$\begin{aligned} T(n) &= 2T(n/2) + n \\ &= 2(n/2)\lg(n/2) + n \\ &= n(\lg n - 1) + n \\ &= n \lg n - n + n \\ &= n \lg n , \end{aligned}$$

which completes the inductive proof for exact powers of 2.

Since n = 2^k, F(k) = T(2^k). So we can rewrite T(n) = n lg n as T(2^k) = 2^k lg 2^k.

Base Case: When k =1, then n=2 and T(2) = 2 lg 2 = 2 * 1 = 2. This is equal to the recurrence stating that T(n) = 2 if n =2, so the base case is proven.

Inductive Hypothesis: Assume that T(2^k) = 2^k lg 2^k. We need to show that T(2^(k+1)) = 2^(k+1) * lg 2^(k+1).

T(2^(k+1)) = 2T((2^(k+1))/2) + 2^(k+1)
= 2T(2^k) + 2^(k+1)
= 2(2^k * lg 2^k) + 2^(k+1)
= 2^(k+1) * (lg 2^k + 1)
= 2^(k+1) * (lg 2^k + lg 2)
= 2^(k+1) * lg 2^(k+1)

Since we have shown that T(2^k) implies T(2^(k+1)) when 2^k = n, we have proven that T(n) = n lg n and the proof is finished.

CS 325 - HW 1 Solutions Examples

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).

4. **2 points - 0.25 point deduction for each one missed.**

a.	$f(n) = n^{0.75}$;	$g(n) = n^{0.5}$	f(n) is Ω (g(n))
b.	f(n) = n;	$g(n) = \log^2 n$	f(n) is Ω(g(n))
c.	f(n) = log n;	g(n) = lg n	f(n) is Θ(g(n))
d.	$f(n) = e^n$;	$g(n) = 2^n$	f(n) is Ω(g(n))
e.	$f(n) = 2^n$;	$g(n) = 2^{n-1}$	f(n) is Θ(g(n))
f.	$f(n) = 2^n$;	$g(n) = 2^{2^n}$	f(n) is O(g(n))
g.	$f(n) = 2^n$;	g(n) = n!	f(n) is O(g(n))
h.	f(n) = nlgn;	$g(n) = n\sqrt{n}$	f(n) is O(g(n))

CS 325 - HW 1 Solutions Examples

5) 3 points total- There are several different correct algorithms

3 points = Algorithm logically correct with $\Theta(n \lg n)$ running time explained, example and pseudocode. The example can be a sample execution on a data set or an example of code.

2 points = Everything correct but missing the example

1 points = Algorithm is not $\Theta(n \lg n)$-or is not logically correct but an example is given.

Describe a $\Theta(n \lg n)$- time algorithm that, given a set S of n integers and another integer x, determines whether or not there exist two elements in S whose sum is exactly x. Give an example.

The following algorithm solves the problem:

1. Sort the elements in S.
2. Form the set $S' = \{z : z = x - y \text{ for some } y \in S\}$.
3. Sort the elements in S'.
4. Merge the two sorted sets S and S'.
5. There exist two elements in S whose sum is exactly x if and only if the same value appears in consecutive positions in the merged output.

To justify the claim in step 4, first observe that if any value appears twice in the merged output, it must appear in consecutive positions. Thus, we can restate the condition in step 5 as there exist two elements in S whose sum is exactly x if and only if the same value appears twice in the merged output.

Suppose that some value w appears twice. Then w appeared once in S and once in S'. Because w appeared in S', there exists some $y \in S$ such that $w = x - y$, or $x = w + y$. Since $w \in S$, the elements w and y are in S and sum to x.

Conversely, suppose that there are values $w, y \in S$ such that $w + y = x$. Then, since $x - y = w$, the value w appears in S'. Thus, w is in both S and S', and so it will appear twice in the merged output.

Steps 1 and 3 require $\Theta(n \lg n)$ steps. Steps 2, 4, 5, and 6 require $O(n)$ steps. Thus the overall running time is $O(n \lg n)$.

Note: Must also check if x/2 is in S

An Alternative method:

1) Perform a Merge Sort on set S
2) For each y in S{
    Perform a binary search on S to find a value z such that z = x - y.
    If z exists, return true, else return false.

Step 1 execution time is $n \cdot log_2(n)$.
Step 2 execution time is also $n \cdot log_2(n)$.
However combined they are still only $\Theta(n \cdot log_2(n))$.

CS 325 - HW 1 Solutions Examples

6)

a. **2 points: 0.5 for true, 1.5 for proof**

If $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$, then $f_1(n)+f_2(n) = O(g_1(n)+g_2(n))$.

True

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

$f_1(n) \le c_1 g_1(n))$ for $n \ge n_1$ and $f_2(n) \le c_2 g_2(n)$ for $n \ge n_2$

Since the functions are asymptotically positive

$f_1(n)+ f_2(n) \le c_1 g_1(n) + c_2 g_2(n) \le (c_1 + c_2)\, g_1(n) + (c_1 + c_2)\, g_2(n)$

$\le (c_1 + c_2)\, (g_1(n)) + g_2(n))$ for $n \ge \max(n_1, n_2)$

Let $k = (c_1 + c_2)$ and $n_0 = \max(n_1, n_2)$ then

$f_1(n) \cdot f_2(n) \le k(g_1(n) \cdot g_2(n))$ for $n \ge n_0$; $k, n_0 > 0$ and by definition

$f_1(n)+ f_2(n) = O(g_1(n)+g_2(n))$

an alternative is to take $k = \max(c_1, c_2)$ and/or $n_0 = (n_1 + n_2)$

b. **1 point for any counter example**

If $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$, then $\frac{f_1(n)}{f_2(n)} = O\left(\frac{g_1(n)}{g_2(n)}\right)$

False counter example:

Let $f_1(n) = x^4$ and $f_2(n) = x^3$, $g_1(n) = x^4$ and $g_2(n) = x^4$

c. **2 points: 0.5 for true, 1.5 for proof**

$\max(f_1(n), f_2(n)) = \Theta(f_1(n) + f_2(n))$.

True

First, let's clarify what the function $\max(f(n), g(n))$ is. Let's define the function $h(n) = \max(f(n), g(n))$. Then

$$h(n) = \begin{cases} f(n) & \text{if } f(n) \ge g(n), \\ g(n) & \text{if } f(n) < g(n). \end{cases}$$

Since $f(n)$ and $g(n)$ are asymptotically nonnegative, there exists n_0 such that $f(n) \ge 0$ and $g(n) \ge 0$ for all $n \ge n_0$. Thus for $n \ge n_0$, $f(n) + g(n) \ge f(n) \ge 0$ and $f(n) + g(n) \ge g(n) \ge 0$. Since for any particular n, $h(n)$ is either $f(n)$ or $g(n)$, we have $f(n) + g(n) \ge h(n) \ge 0$, which shows that $h(n) = \max(f(n), g(n)) \le c_2(f(n) + g(n))$ for all $n \ge n_0$ (with $c_2 = 1$ in the definition of Θ).

Similarly, since for any particular n, $h(n)$ is the larger of $f(n)$ and $g(n)$, we have for all $n \ge n_0$, $0 \le f(n) \le h(n)$ and $0 \le g(n) \le h(n)$. Adding these two inequalities yields $0 \le f(n) + g(n) \le 2h(n)$, or equivalently $0 \le (f(n) + g(n))/2 \le h(n)$, which shows that $h(n) = \max(f(n), g(n)) \ge c_1(f(n)+g(n))$ for all $n \ge n_0$ (with $c_1 = 1/2$ in the definition of Θ).

CS 325 - HW 1 Solutions Examples

7) **6 points total-**

a) 2 points - iterative code and recursive code

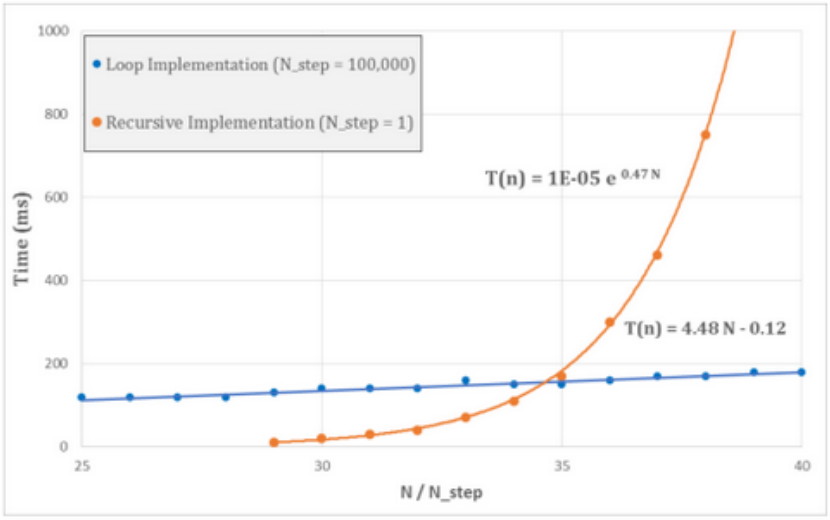
b) Not graded

c) 2 points - Plot the running time data you collected on graphs with n on the x-axis and time on the y-axis.

d) 2 points What type of function (curve) best fits each data set?

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

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