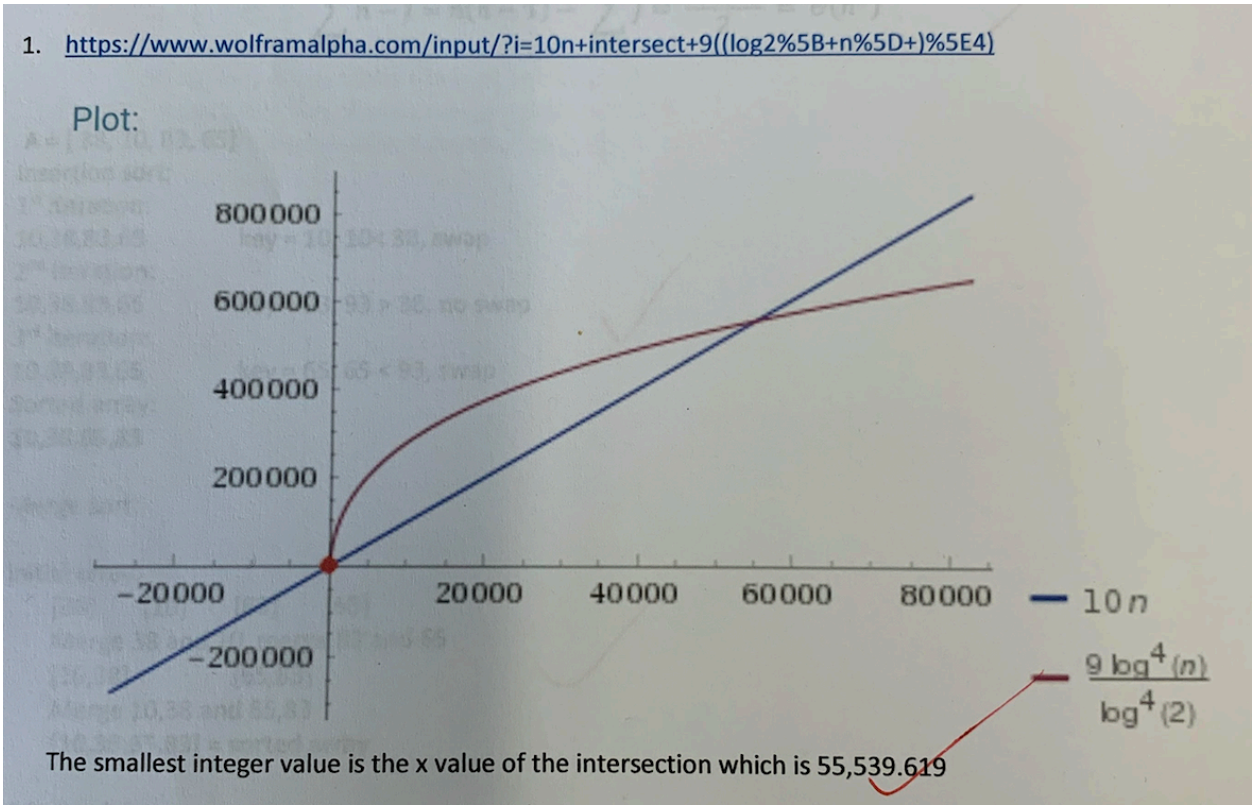
HW1

1. from Stavros Vallas

1. https://www.wolframalpha.com/input/?i=10n+intersect+9((log2%5B+n%5D+)%5E4)



The smallest integer value is the x value of the intersection which is 55,539.619

2&3&4 from Victor M Espaillat

(a) Write pseudocode for this algorithm, which is known as selection sort.

```
Selection_Sort(A,n)
1.  for i = 1 to n-1
2.      smallest = i
3.      for j = i+1 to n
4.          if A[j] < A[smallest]
5.              smallest = j
6.      swap(A[i], A[smallest])
```

(b) What loop invariant does this algorithm maintain?

Loop Invariant: The subarray A[1..i-1] contains the i-1 smallest elements from original array A[1..n] in sorted order.

Initialization: The first subarray is an empty array, and is therefore sorted.

Maintenance: After each iteration, the subarray is expanded by one element. This addition is always the smallest element in A that has not yet been inserted into the subarray. We are always adding the next smallest element from A we can find. The subarray will therefore always be sorted. The subarray's first element is the smallest element, then the second-smallest is inserted after that, then the third-smallest, and so on.

Termination: According to our loop invariant, by the end, the subarray will contain all elements in the original array but in sorted order.

(c) Why does it need to run for only the first $n-1$ elements, rather than for all n elements?

This algorithm needs to run for only $n-1$ elements, rather than for all n because the last element would already be in the correct position. There would be no more "smallest" reassignments to be made because there would only be one element left. All smaller elements have already been rearranged. Thus the last element is the largest element of the entire array and therefore belongs in the position it is already in.

(d) Give the best-case and worst-case running times of selection sort in Θ-notation.

	Selection_Sort(A,n)	Cost	Times
1.	for i = 1 to n-1	c_1	n
2.	smallest = i	c_2	$n-1$
3.	for j = i+1 to n	c_3	$\sum_{i=2}^{n}(t_i)$
4.	if A[j] < A[smallest]	c_4	$\sum_{i=2}^{n}(t_i-1)$
5.	smallest = j	c_5	$\sum_{i=2}^{n}(t_i-1)$
6.	swap(A[i], A[smallest])	c_6	$n-1$

The line of code that contributes most to the total running time is line 3.

Best-case: Array A is already sorted. Then the if statement of line 4 will always be **false** and so line 5 will never execute.

$T(n)$: total running time.

t_i : number of times that the for loop test of line 3 is executed for that value of i.

$t_i = i - 1$

$$\sum_{i=2}^{n}(t_i) = \sum_{i=2}^{n}(i-1)$$

$$= \sum_{i=2}^{n}(i) - \sum_{i=2}^{n}(1)$$

$$= \left(\frac{n^2}{2} + \frac{n}{2} - 1\right) - (n-1)$$

$$= \frac{n^2}{2} - \frac{n}{2}$$

$$\sum_{i=2}^{n}(t_i - 1) = \sum_{i=2}^{n}(i-2)$$

$$= \sum_{i=2}^{n}(i-1) - \sum_{i=2}^{n}(1)$$

$$= \left(\frac{n^2}{2} - \frac{n}{2}\right) - (n-1)$$

$$= \frac{n^2}{2} - \frac{3n}{2} + 1$$

$$T(n) = c_1 n + c_2(n-1) + c_3\left(\frac{n^2}{2} - \frac{n}{2}\right) + c_4\left(\frac{n^2}{2} - \frac{3n}{2} + 1\right) + c_6(n-1)$$

$$= c_1 n + c_2 n - c_2 + \frac{c_3 n^2}{2} - \frac{c_3 n}{2} + \frac{c_4 n^2}{2} - \frac{c_4 3n}{2} + c_4 + c_6 n - c_6$$

$$= \left(\frac{c_3}{2} + \frac{c_4}{2}\right) n^2 + \left(c_1 + c_2 - \frac{c_3}{2} - \frac{c_4 3}{2} + c_6\right) n + (c_4 - c_2 - c_6)$$

$$= an^2 + bn + c$$

$$\boxed{= \Theta(n^2)}$$

Worst-case: Array A is sorted in reverse order. Then the if statement of line 4 will always be **true** and so line 5 will always execute.

$$T(n) = c_1 n + c_2(n-1) + c_3\left(\frac{n^2}{2} - \frac{n}{2}\right) + (c_4 + c_5)\left(\frac{n^2}{2} - \frac{3n}{2} + 1\right) + c_6(n-1)$$

$$= c_1 n + c_2 n - c_2 + \frac{c_3 n^2}{2} - \frac{c_3 n}{2} + \frac{c_4 n^2}{2} - \frac{c_4 3n}{2} + c_4 + \frac{c_5 n^2}{2} - \frac{c_5 3n}{2} + c_5 + c_6 n - c_6$$

$$= \left(\frac{c_3}{2} + \frac{c_4}{2} + \frac{c_5}{2}\right) n^2 + \left(c_1 + c_2 - \frac{c_3}{2} - \frac{c_4 3}{2} - \frac{c_5 3}{2} + c_6\right) n + (c_4 + c_5 - c_2 - c_6)$$

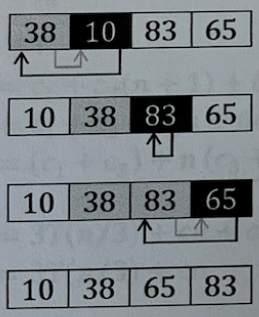
$$= an^2 + bn + c$$

$$\boxed{= \Theta(n^2)}$$

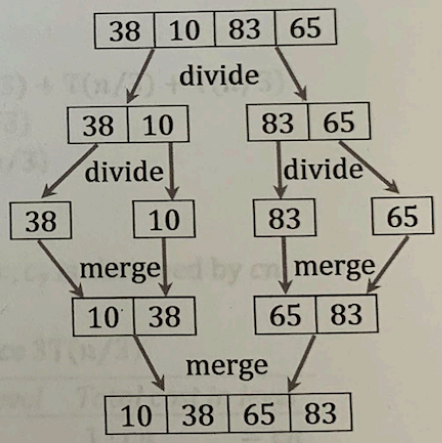
The running time will still be the same.

3. **Sorting Algorithms**: (20 points) Use textbook Figure 2.2 and Figure 2.4 as models to illustrate the operations of Insertion_Sort and Merge_Sort on the array A = <38, 10, 83 ,65>

Insertion Sort



Merge Sort



4. **Analysis:** (20 points) There is a mystery function called Mystery(n) and the pseudocode of the algorithm is shown below. Please analyze the worst-case asymptotic execution time of this algorithm using the method we learned in the class. Express the execution time as a function of the input value n. Assume that $n = 3^k$ for some positive integer $k \geq 1$. Justify your answer.

Mystery(n)	Cost	Times
1. if n ≤ 1	c_1	1
2. return 1	c_2	1
3. for i = 1 to n	c_3	$n+1$
4. for j = 1 to 5	c_4	$6n$
5. print "this is a recursive call."	c_5	$5n$
6. Mystery(n/3)	$T(n/3)$	1
7. Mystery(n/3)	$T(n/3)$	1
8. Mystery(n/3)	$T(n/3)$	1

Solving for the total running time $T(n)$.

Case $n \leq 1$:

$$T(n) = \underbrace{c_1 + c_2}_{c_6}$$
$$= c_6$$

Case $n > 1$:

$$T(n) = c_1 + c_3(n+1) + 6c_4 n + 5c_5 n + T(n/3) + T(n/3) + T(n/3)$$
$$= c_1 + c_3 + c_3 n + 6c_4 n + 5c_5 n + 3T(n/3)$$
$$= \underbrace{(c_1 + c_3)}_{c_7} + n\underbrace{(c_3 + 6c_4 + 5c_5)}_{c} + 3T(n/3)$$
$$= 3T(n/3) + cn + c_7$$
$$= 3T(n/3) + cn$$

As $n \to \infty$, c_7 is absorved by cn.

Recursion tree to solve the recurrence $3T(n/3)$

Level #	*# of leaves in level*	*Cost per leave in level*	*Total cost in level*	
1	1	cn	$1 \cdot cn$	$= cn$
2	3	$cn/3$	$3 \cdot cn/3$	$= cn$
3	9	$cn/9$	$9 \cdot cn/9$	$= cn$
4	27	$cn/27$	$27 \cdot cn/27$	$= cn$
⋮	⋮	⋮	⋮	
i	3^{i-1}	$cn/3^{i-1}$	cn	

Since we assumed that $n = 3^k$, for some $k \in \mathbb{Z}^+$, the problem size will eventually equal 1 and the recursive calls will stop since the base case will be executed. The number of leaves in this level will equal n, and so the cost for each leave will be $cn/n = c$.

Since we know that each leave has cost c at the last level, we can use the equation for *cost per leave* to solve for i, the total number of levels generated by the recurrence.

$$\begin{aligned}
\frac{cn}{3^{i-1}} &= c \\
cn &= c \cdot 3^{i-1} \\
n &= 3^{i-1} \\
\log_3 n &= \log_3(3^{i-1}) \\
\log_3 n &= i - 1 \\
\log_3 n + 1 &= i
\end{aligned}$$

To solve for the recurrence $3T(n/3)$, multiply the total number of levels i that it generates with the cost associated with each level (cn).

$$\begin{aligned}
3T(n/3) &= i \cdot cn \\
&= (\log_3 n + 1)cn \\
&= cn \log_3 n + cn
\end{aligned}$$

Plugging this back into $T(n)$ to solve for the total running time.

$$\begin{aligned}
T(n) &= 3T(n/3) + cn \\
&= cn \log_3 n + cn + cn \\
&= cn \log_3 n + 2cn \\
&= cn \log_3 n + c_8 n \\
&= cn \log_3 n \\
&= \boxed{\Theta(n \log_3 n)}
\end{aligned}$$

As $n \to \infty$, $c_8 n$ is absorved by $cn \log_3 n$.

5. from Tom Clunie

5)

a. For pair (i,j) with i < j in set {1...n}, the element with the most inversions will have j=n. If A[i] > A[j] for all elements in the set, the array {n, n-1, ..., 1} will have n – 1 + n – 2 + ... + n – (n – 1) inversions, or $\left(\sum_{k=1}^{n-1} n - k\right)$

b.

```
Count(A,p,q,r)
1.  N1 = q-p+1
2.  N2 = r-q
3.  Result = 0
4.  Let L[1..N1+1] and R[1..N2+1] be new arrays
5.  For i = 1 to N1
6.      L[i] = A[p+i-1]
7.  For j = 1 to N2
8.      R[j] = A[q+j]
9.  L[N1+1] = infinity
10. R[N2+1] = infinity
11. For k = p to r
12.     If L[i] <= R[j]
13.         A[k] = L[i]
14.         i = i + 1
15.     else
16.         A[k] = R[j]
17.         j = j + 1
18. // R[j] must be less than this and all other remaining elements in L
19.         Result = Result + N1 – i
20. Return Result
```

```
Count_Inversions(A,p,r)
1.  Result=0
2.  If (p < r)
3.      q = floor((p+r)/2)
4.      Result = Result + Count_Inversions(A,p,q)
5.      Result = Result + Count_Inversions(A, q+1, r)
6.      Result = Result + Count(A,p,q,r)
7.  Return Result
```