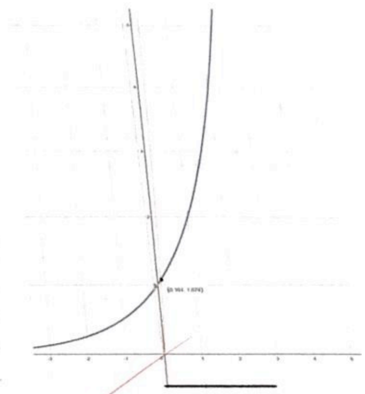
HW1

1. By Bonnie Liu

1.2-3
What is the smallest value of n such that an algorithm whose running time is $100n^2$ runs faster than an algorithm whose running time is 2^n on the same machine?



From the problem, we want the smallest value of n, when running time $100n^2 > 2^n$. We can draw a graph to see, when $100n^2 > 2^n$.
From the graph, we know when $n = 0.1037$, $100n^2$ and 2^n have the same value. (orange line is $y = 100n^2$, blue line is $y = 2^n$)
But n need greater and equal to 2, because the sort must have 2 or more values to sort. The graph will not help us.
We can write a program to help us to do this.

```
int main()
{ int n=2;
  while(100*n*n > pow(2,n))
  { n++; }
  cout << n;
  return 0;
}
```

The program gives me the result is 15

We can verify it.

① when $n = 14$, $100 \times 14 \times 14 = 19600$, $2^{14} = 16384$ → $100 \cdot n^2 > 2^n$

② when $n = 15$, $100 \times 15 \times 15 = 22500$, $2^{15} = 2768$ → $2^n > 100n^2$

③ when $n = 16$, $100 \times 16 \times 16 = 25600$, $2^{16} = 65536$ → $2^n > 100n^2$

2. By DangNhi Ngoc Ngo

20

```
Linear_Search (A, v)
for i = 1 to A.length
    if A[i] == v
        return i
return NIL
```

Loop invariant: At the start of each iteration of for loop $A[i] \neq v$

Initialization: The function shows true before first iteration

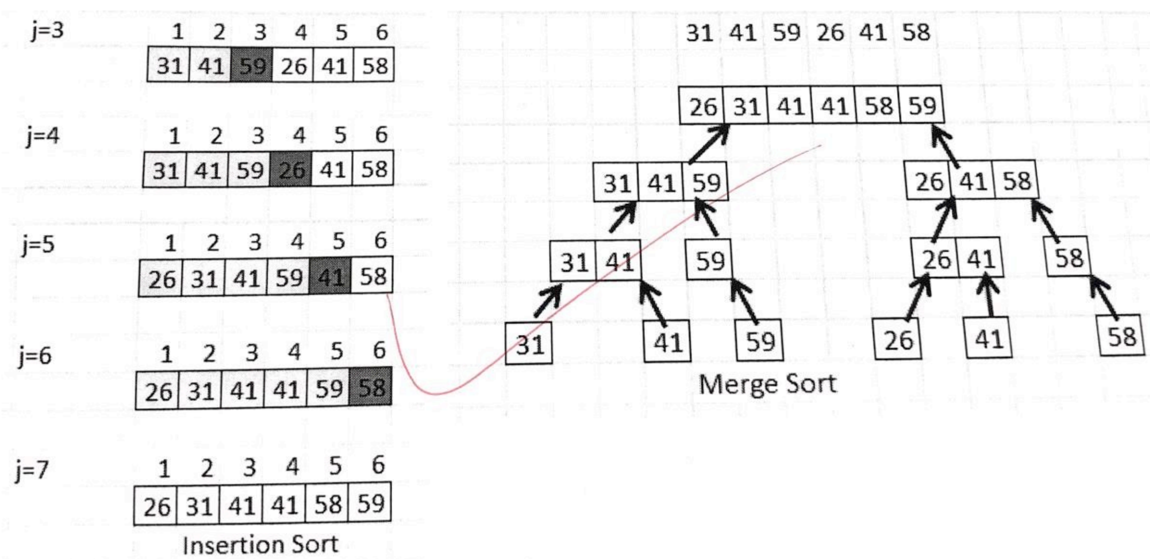
Maintenance: The loop invariant holds true for every iteration

If a match is found, the function will return

Termination: The function will either return an index or NIL when the loop ends

3. **Sorting Algorithms**: (20 points)

By Sainath Gopinath



4.

By Amy Mazzucotelli

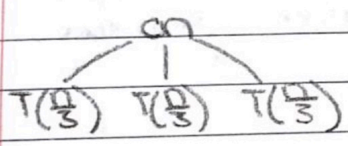
4. Mystery(n)

	run time	# times
if $n \le 1$	c_1	1
return 1	c_2	1
for $i = 1$ to 5	c_3	6
for $j = 1$ to n	c_4	$6(n+1)$
print " "	c_5	$6n$
Mystery($\frac{n}{3}$)	$T(\frac{n}{3})$	1
Mystery($\frac{n}{3}$)	$T(\frac{n}{3})$	1
Mystery($\frac{n}{3}$)	$T(\frac{n}{3})$	1

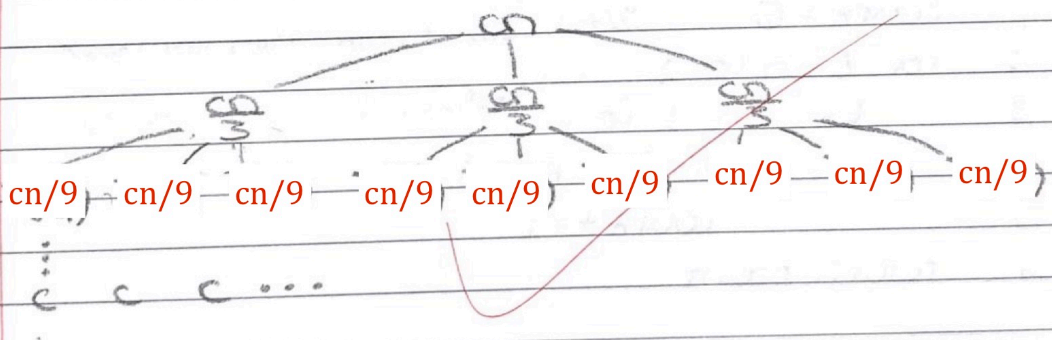
base case: $\Theta(1)$

$\Theta(n)$

$$T(n) = \begin{cases} \Theta(1) & n \le 1 \\ 3T(\frac{n}{3}) + \Theta(n) & n > 1 \end{cases}$$



$$T(\frac{n}{3}) = 3T(\frac{n}{9}) + \frac{cn}{3}$$



→ each level has 3^k elements, sum of elements per level = cn

$c = \frac{cn}{3^{k-1}}$

$n = 3^{k-1}$

$\log_3 n = k-1$

$k = \log_3 n + 1$ layers

$T(n) = cn \cdot k = cn(\log_3 n + 1)$

$= cn\log_3 n + cn$

$\Rightarrow \boxed{\Theta(n \log_3 n)}$

5. Algorithm Design (20 points)

By David Baumann

a) The reverse-order array $(n, n-1, \dots, 2, 1)$ has the most inversions.
In this case, the number of inversions is: $\sum_{i=1}^{n} i-1 = \boxed{\frac{n(n-1)}{2}}$

b)
```
count_inversions(A, n)
   let Temp[1..n] be new Array
   for i=1 to n
        Temp[i] = A[i]
   return merge-sort(temp, 1, n)

merge_Sort(A, p, r)
   num_inversions = 0
   if (p < r)
       q = ⌊(p+r)/2⌋
       num_inversions = count_inversions(A, p, q)
       num_inversions += count_inversions(A, q+1, r)
       num_inversions += Merge(A, p, q, r)
   return num_inversions
```

⑤ b cont.)
```
merge(A, p, q, r)
   n1 = q - p + 1
   n2 = r - q
   let L[1..n1 +1] and R[1..n2 +1] be new arrays
   for i = 1 to n1
       L[i] = A[p+i-1]
   for j = 1 to n2
       R[j] = A[q+j]
   L[n1 +1] = ∞
   R[n2 +1] = ∞
   i = 1
   j = 1
   num_inversions = 0
   for k = p to r
       if L[i] ≤ R[j]
           A[k] = L[i]
           i = i+1
       else
           A[k] = R[j]
           j = j+1
           num_inversions += q - i      // if we take from R, the remaining values
                                        // in L are are all inversions
   return num_inversions
```