CS101 HW1

1. pseudo code:

```
for i = 1 to A.length - 1          1
    j = i                          2
    for k = i+1 to A.length        3
        if A[k] < A[j]             4
            then j = k             5
    temp = A[j]                    6
    A[j] = A[i]                    7
    A[i] = temp                    8
```

The loop invariant in lines 1-8 is this statement:

initialization → at the first iteration, when $i = 1$ the loop is an empty statement since there's no number, j, such that $1 \le j < i$ $\therefore$ ~~the~~ true at the begining of the loop.

maintenence → With the initialization being true, that means $A[j] \le A[k]$ for all integers j and k with $1 \le j < i$ and $j < k \le n$. Lines 2-8 $A[i]$ is switched with $A[j]$ where $i < j$ and $A[j]$ is a smaller number. After lines 2-8 i is increase by 1 showing $A[j] \le A[k]$ for all j with $1 \le j < i$ & $j < k \le n$ at the start of the next iteration.

Termination → The loop ends when i equals n. The loop invariant shows $A[j] \le A[k]$ for all j and k, having $1 \le j < k \le n$ $\therefore$ $A[1...n]$ are sorted. This also shows ~~that~~ $A[1..n]$ is equal to the orignal $A[1...n]$ just sorted. Proving the correctness ~~of the~~ of the algorim

1 continued.

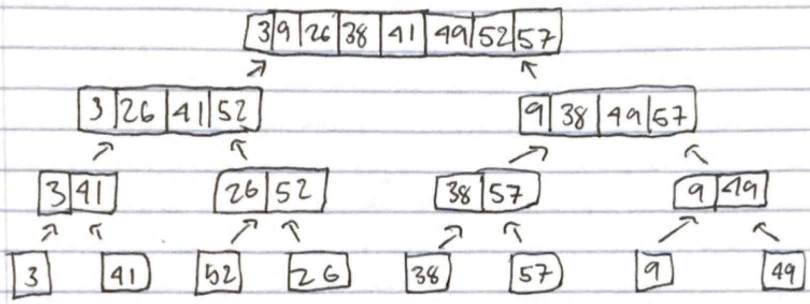
The algorim only runs to n-1 and not all n since the last iteration will have compared A[n-1] and A[n], ~~start~~ putting the largest number last in the array.

Best case : $\Theta(n^2)$

Worst case : $\Theta(n^2)$

for both cases the algorithm takes one element at a time and compare it with all other elements $\therefore$ same run time.

2 visualizing merge sort on the array $A = (3, 41, 52, 26, 38\ 57, 9, 49)$



3

$$T(n) = \begin{cases} \Theta(1) & \text{if } n=1 \\ T(n-1) + O(n) & \text{if } n>1 \end{cases}$$

4 recursiveBinarySearch (A, start, end, x)

```
    if (start<end)
        return NIL

    mid = floor((start + end)/2);
    if x == A[mid]
        return mid
    if x < A[mid]
        recursiveBinarySearch (A, start, mid, x)
    else
        recursiveBinarySearch (A, mid+1, end, x)
```

recurance : $T(n) = T(\frac{n}{2}) + \Theta(1)$. $T(n) = \Theta(n^{\log_2 1} \lg n) = \Theta(\lg n)$

by master theorem

- case 2

5. a. $(2,1)$; $(3,1)$; $(8,1)$; $(6,1)$; $(8,6)$

b. $\langle n, n-1, \ldots, 1 \rangle$. It has $\rightarrow (n-1)+(n-2+\ldots+1 = \frac{(n-1)n}{2}$ conversions

c. In Insertion-Sort the while loops has the condition of having $i<j$ & $A[i] > A[j]$. The count of inversions in the input array, x, would be equal to the number of times the body of the while loop is executed for $2 \leq j \leq A.length$. $\rightarrow x = \sum_{j=2}^{n} (t_j - 1)$

we can then express $T(n) = c_1 n_1 + c_2 (n-1) + c_4 (n-1)$

$+ c_5 (X + (n-1)) + c_6 X + c_7 X + c_8 (n-1)$

d. next page ->

```
mergeCount(A,p,r)
    count=0
    if p<r
        q = floor((p+r)/2)
        count = count + mergeCount(A,p,q)
        count = count + mergeCount(A,q+1,r)
        count = count + modMerge(A,p,q,r)
    return count
```

d. modMerge (A, p, q, r)

```
count = 0
x = q - p + 1
y = r - q
make arrays B[1...x] & C[1...y]
for i = 1 to x
    B[i] = A[p+i-1]
for j = 1 to y
    C[j] = A[q+j]
i = 1
j = 1
k = p
while i != x+1 & j != y+1
    if B[i] ≤ C[j]
        A[k] = B[i]
        i = i+1
    else A[k] = C[j]
        count = count + j
        j = j+1
    k = k+1
if j == y+1
    for m = i to x
        A[k] = B[m]
        count = count + y
        k = k+1
return count
```