Syeda Kazmi-Shah

* Chapter 2: 2.2-1, 2.2-2, 2.3-3, 2.3-5, 2.3-6
* Chapter 3: 3.1-1, 3.1-4, 3.1-8, 3.2-3

**CH:2**

2.2-1) Θ(n^3)

2.2-2) Best case: Θ (n^2), worst case: Θ (n^2)

**pseudocode: Selection sort**

Selection\_Sort(a,n) #array, size of array

{ for i🡨 0 to n-2

{ min🡨 i

for j🡨i+1 to n-1

{ if (a[j] < a[min])

min🡨j

}

temp🡨 a[i]

a[i]🡨a[min]

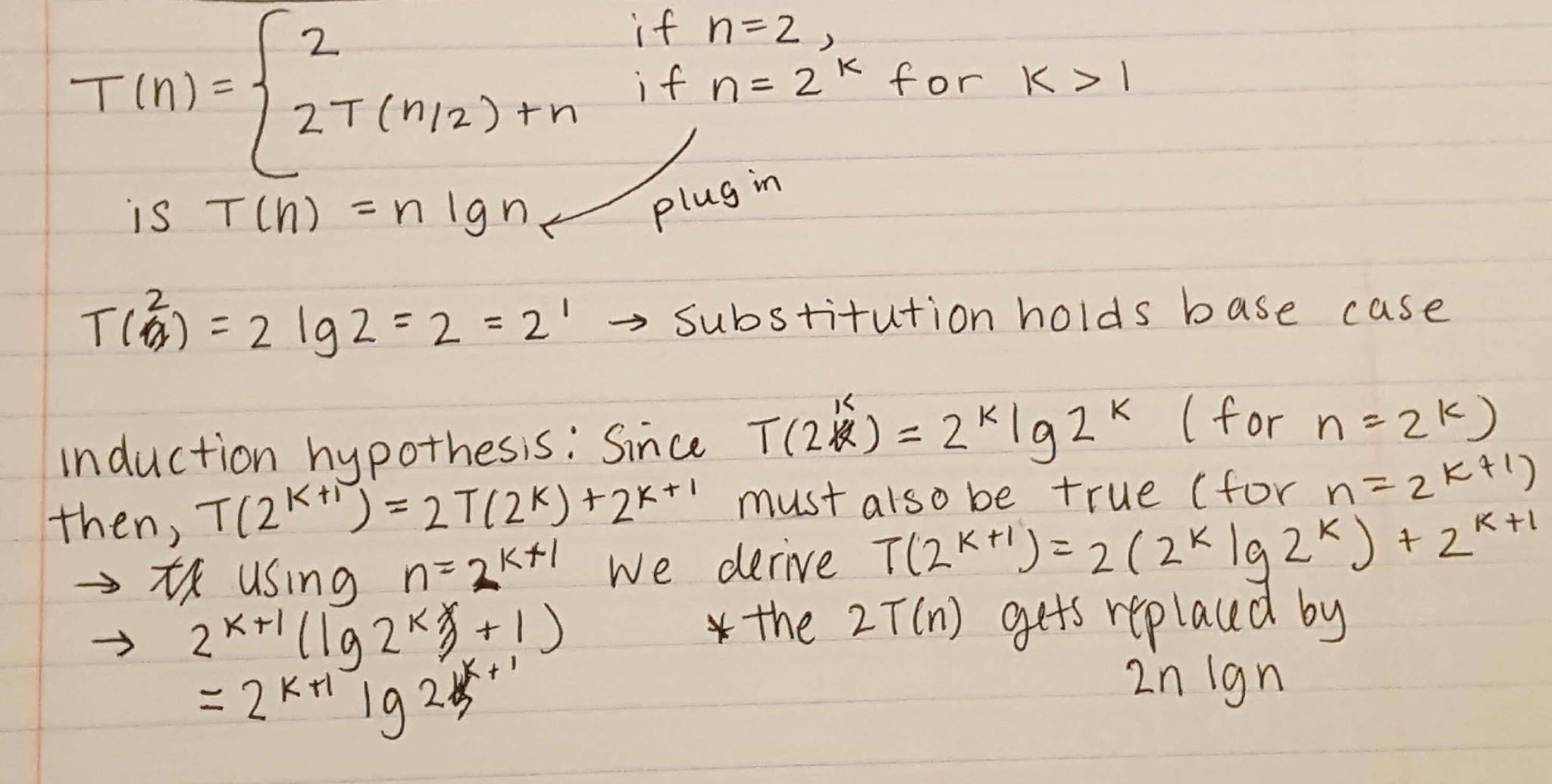
a[min]🡨temp

}

}

**the invariant it maintains is** that after i iterations, the subarray holding the first i elements of the array a is sorted.

**It runs for n-1** because the last element does not need to be sorted because it is already sorted. Going through all n elements will be unnecessary. If it were not sorted, the loop invariant would not hold.

2.3-3) 

2.3-5)

**Psuedocode: Binary Search:**

Binary\_Search( list, start, end, k) {

m 🡨 (start + end)/2

if list[m] == k

return true

if list[m] > k

return Binary\_Search(list, start, m-1, k)

if list[m] < k

return Binary\_Search(list, m+1, end, k)

}

The worst case time of binary search is Θ(lg n) because the list is already sorted (Θ(1)). The algorithm begins by cutting the list in half and checking which side of the half has the element we are looking for and ignores the other half that doesn’t. We do this recursively until our binary search is complete and we have found our number. If it were unsorted it would have a multiplication by n. (note: T(n)=T((n-1)/2)+θ(1)=θ(log(n)))

2.3-6) No, a binary search would not help the running time of insertion sort. It would narrow down which elements need to be sorted, but it would not be able to get rid of the inner insertion loop copying the elements which need to be sorted over to their correct spots.

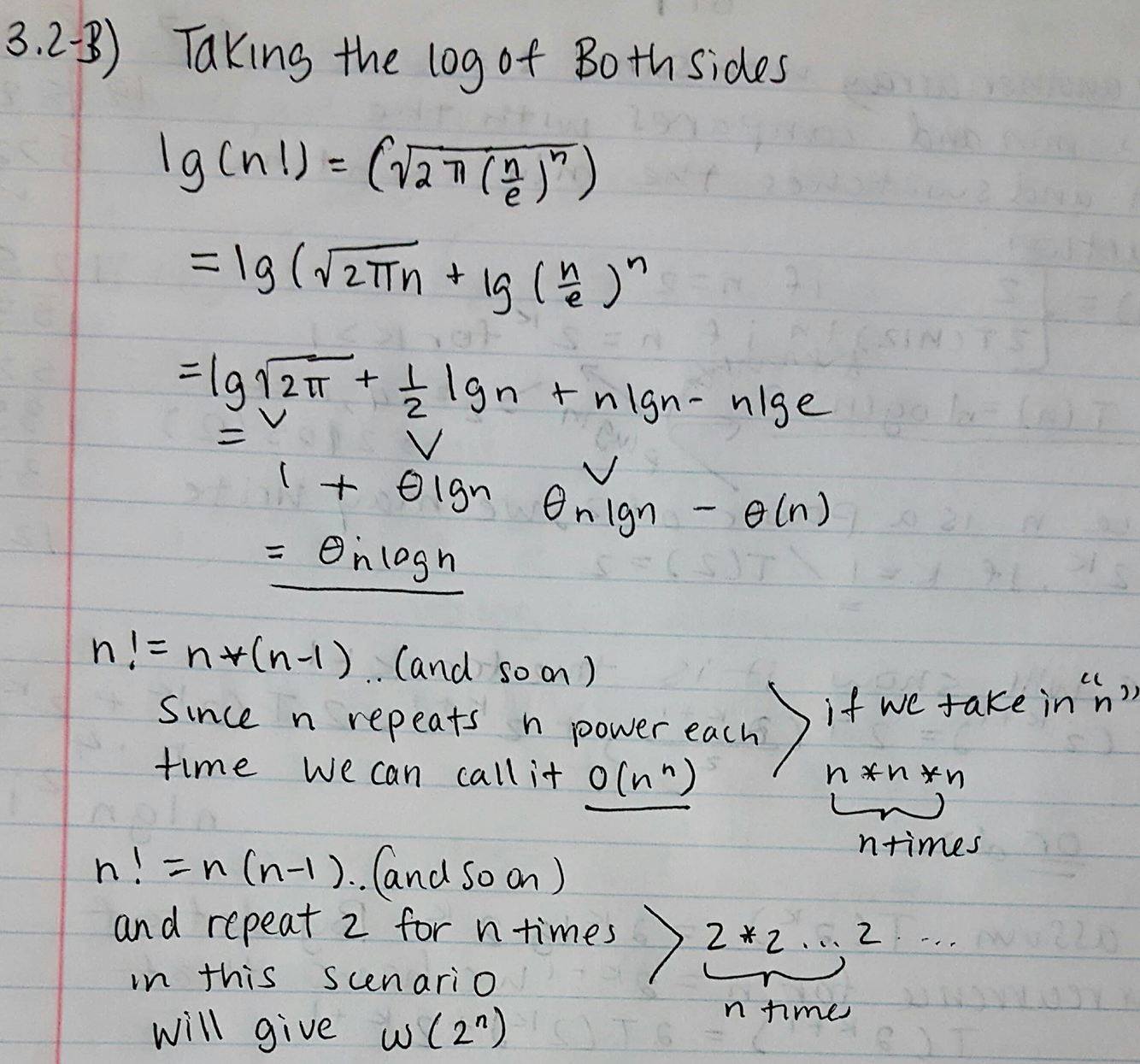
**Ch:3**

3.1-1) (f(n),g(n)) = Θ(f(n)+g(n)) because functions f(n) and g(n) are asymptotically non negative. f(n)≥0 and g(n)≥0 . f(n)+g(n) ≥ f(n) and f(n)+g(n) ≥ g(n) are both ≥ 0 . Thus we can say f(n) + g(n) ≥ max(f(n), g(n)). (Also in terms of big-oh and omega).

3.1-4) Yes, 2^(n+1) is O(2^n). However, 2^2n is not dominated by O(2^n). This is because as the value of n goes up the constant is not negligible because it leads the exponent to be a greater number. 2^2n = 2^n \* 2^n and as a result it is a much bigger number than 2^n, thus it cannot be dominated by a smaller number such as 2^n. There is also no c that can make 2^n greater than 2^2n for all n (too big growth).

3.1-8) Ω(g(n,m)) = {f(,n) there exist positive constants c, n0, and m0 such that 0 ≤ cg(n, m) ≤ f(n, m) .

Θ((g(n,m)) = {f(,n) there exist positive constants c1, c2, n0, and m0 such that c1g(n, m) ≤ f(n, m) ≤ c2g(n, m) for all n ≥ n0 and m ≥ m0 (bounded between the two)

3.2-3) 

3.2-4) lgn! Is not polynomially bounded. But, lglgn! is polynomially bounded. It’s bounded by the concept g(x)≤f(x)≤h(x).