Mohamed Kharma

Professor: Ahmet C. Yuksel

Class: CSC 22000

Assignment #1

Q1: To prove the correctness of Insertion Sort using induction, we need to show that the algorithm is true for all n >= 1 (positive integers only because any array less than 1 cannot be sorted).

Prove by induction consists of two main cases:

1. The base case: in our case, let P(n) represent the Insertion Sort algorithm, then we need to prove that the statement P (1) holds true.
2. The induction step: Assume that the algorithm holds true for some given (n) values in P(n), we need to prove that the derive P (n+1) is true for these (n) values.

INSERTION-SORT (A, n) ⊳ A[1 . . n]

```
for j ← 2 to n
  do key ← A[j]
    i ← j – 1
    while i > 0 and A[i] > key
       do A[i+1] ← A[i]
          i ← i – 1
    A[i+1] = key
```

In insertion sort, we start from the first element in the array by taking each number from the unsorted array and place it in the correct position in the sorted left side. The left side of the array will contain all the sorted values and the right side will contain the unsorted as we move the between the elements until we reach the last one, then they will all be sorted.

The iterations start at j = 2, which means that for j = 2, we move A[j] times. This means that P(n) will equal to the subarrays of A1, A2, …, An which are sorted in terms of iteration n+1.

Therefore, for the base case. Since we established that when j = 2, we only get one element to the left. This means that the subarray A[1 … j-1] will be trivially sorted when n = 1. Hence P (1) is indeed true.

For case 2: To prove that the subarrays will be sorted at the end, we start by considering the two subarrays A1, A2 and how they are iterated by j = n+1 (which is the derive for P(n)). Hence, this means that P (n+1) is indeed true for some (n) values. Therefore, algorithm works correctly, and correctness of Insertion Sort has been proved using induction.

Next. Rewriting the correctness of Insertion Sort algorithm from the book.

Loop Invariant:

At the beginning of each iteration of the for loop, the subarray A[1 .. j-1] will contain all the elements that are originally in the subarray A[1 .. j-1] but in the sorted order.

Initialization:
Since we are starting with j=2. The subarray A[1 .. j-1] will contain a single element A[1], and since there is only a single element (the original element of A[1]), its going to be trivially sorted.

Maintenance:
The outer for loop selects element A[j] and position it properly into A[1..j-1] element using the while loop. Since the array A[1..j-1] began sorted, inserted element A[j] into the proper place produces A[1..j] in sorted order(first contain j element). Another way to look at it would be by looking at the iteration elements A[j-1], A[j-2], A[j-3], and so on, which are shifted to the right. This help the sequence of the array to maintain in the proper order until the proper place for key (which is the previous element of A[j]) is found. Therefore, at the next iteration, the subarray A[1 .. j] has the same elements but in sorted order.

Termination:
The for loop terminates once j=n+1. Therefore, we can conclude that n= j-1. Plugging the value of (n) into the loop invariant, then the subarray A[1..(n+1)-1] will equal to A[1.. j-1] which equal to A[1 .. n] (which is the whole array) what contain all of the elements that are in the original array A[1 .. n] but in sorted order.

Q2:-

1) Show that $\frac{6n^3}{\log_2 n^K} \in \Omega(n^2)$ For any integer constant $K>1$.

$$\Rightarrow \lim_{n\to\infty} \frac{g(n)}{F(n)} = \left\{ \begin{matrix} \infty \\ \{c, c \in \mathbb{R}^+\} \end{matrix} \right\}$$

$$\Rightarrow \lim_{n\to\infty} \left(\frac{\left(\frac{6n^3}{\log_2 n^K}\right)}{n^2} \right) = (6) \lim_{n\to\infty} \left(\frac{\left(\frac{n^3}{\log_2 n^K}\right)}{n^2} \right)$$

$$= (6) \lim_{n\to\infty} \left(\frac{n}{\log_2 n^K} \right) = (6) \lim_{n\to\infty} \left(\frac{n}{K \log_2 n} \right)$$

$$= (6)\left(\frac{1}{K}\right) \lim_{n\to\infty} \left(\frac{n}{\log_2 n} \right)$$

↳ using L'Hopital's Rule, we get:

$$= (6)\left(\frac{1}{K}\right) \lim_{n\to\infty} \left(\frac{\frac{1}{n}}{n \ln(2)} \right) = (6)\left(\frac{1}{K}\right) \lim_{n\to\infty} \left(n \ln(2) \right)$$

$$= (6)\left(\frac{1}{K}\right)(\ln(2)) \underbrace{\lim_{n\to\infty} (n)}_{\infty} = \infty$$

↳ check when $K=2$

$$= (6)\left(\frac{1}{2}\right)(\ln(2)) \underbrace{\lim_{n\to\infty} (n)}_{\infty} = \infty$$

↳ check when $K=5$

$$= (6)\left(\frac{1}{5}\right)(\ln(2)) \underbrace{\lim_{n\to\infty}}_{\infty} = \infty$$

∴ $g(n) \in \Omega(F(n))$ is True.

*Note: Any value for (K) where $K>1$ won't change the result because it's multiplied by (∞)

2) Show that $n+5\log_2 n \in \Theta(n\log_2 n)$.

$\Rightarrow \lim_{n\to\infty} \frac{g(n)}{f(n)} = C, \; C \in \mathbb{R}^+$

$\Rightarrow \lim_{n\to\infty} \left(\frac{n+5\log_2 n}{n\log_2 n}\right) = \lim_{n\to\infty} \left(\frac{n}{n\log_2 n} + \frac{5\log_2 n}{n\log_2 n}\right)$

$= \lim_{n\to\infty} \left(\frac{1}{\log_2(n)} + \frac{5}{n}\right) = \lim_{n\to\infty} \left(\frac{1}{\log_2 n}\right) + \lim_{n\to\infty} \left(\frac{5}{n}\right)$

$= \underbrace{\lim_{n\to\infty} \left(\frac{1}{\log_2(n)}\right)}_{0} + (5) \underbrace{\lim_{n\to\infty} \left(\frac{1}{n}\right)}_{0}$

$0 + (5)\,0 = 0$

$\therefore g(n) \in \Theta(f(n))$ is False.

3) Show that $n^2 \in \Omega(n^K)$ For any integer constant $K > 0$

$\Rightarrow \lim_{n\to\infty} \frac{g(n)}{f(n)} = \left\{\begin{matrix} \infty \\ \{C, C \in \mathbb{R}^+\} \end{matrix}\right.$

$\Rightarrow \lim_{n\to\infty} \left(\frac{n^2}{n^K}\right)$

↳ let $K = 1$

$= \lim_{n\to\infty} \left(\frac{n^2}{n}\right)$

$= \lim_{n\to\infty} (n)$

$= \infty$

↳ let $K = 3$

$= \lim_{n\to\infty} \left(\frac{n^2}{n^3}\right)$

$= \lim_{n\to\infty} \left(\frac{1}{n}\right)$

$= 0$

↳ let $K = 5$

$= \lim_{n\to\infty} \left(\frac{n^2}{n^5}\right)$

$= \lim_{n\to\infty} \left(\frac{1}{n^3}\right)$

$= 0 \in \mathbb{R}^*$

$\therefore g(n) \in \Omega(f(n))$ is False.

4) Show that $n^3 - 2n + \log_2 n \in \Omega(n^2)$.

$$\Rightarrow \lim_{n\to\infty} \frac{g(n)}{f(n)} = \{\infty, c, c \in \mathbb{R}^+\}$$

$$\Rightarrow \lim_{n\to\infty}\left(\frac{n^3 - 2n + \log_2 n}{n^2}\right) = \lim_{n\to\infty}\left(\frac{n^3}{n^2} - \frac{2n}{n^2} + \frac{\log_2 n}{n^2}\right)$$

$$= \underbrace{\lim_{n\to\infty}(n)}_{\infty} - \underbrace{\lim_{n\to\infty}\left(\frac{2}{n}\right)}_{0} + \underbrace{\lim_{n\to\infty}\left(\frac{\log_2 n}{n^2}\right)}_{0} = \infty$$

$\therefore g(n) \in \Omega(f(n))$ is True.

5) Show that $4n + 2n\log_2 n \in \Theta(n\log_2 n)$.

$$\Rightarrow \lim_{n\to\infty} \frac{g(n)}{f(n)} = c, \; c \in \mathbb{R}^+$$

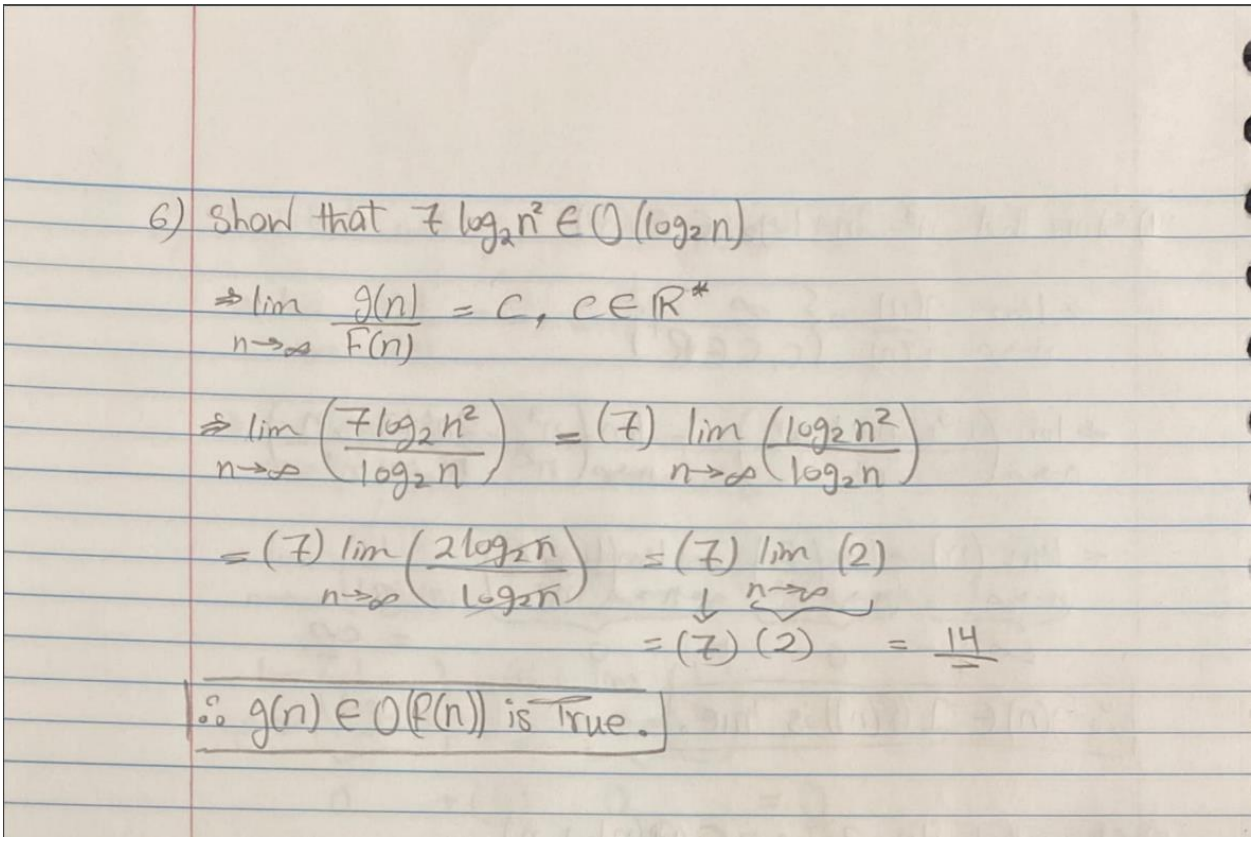
$$\Rightarrow \lim_{n\to\infty}\left(\frac{4n + 2n\log_2 n}{n\log_2 n}\right) = \lim_{n\to\infty}\left(\frac{4n}{n\log_2 n} + \frac{2n\log_2 n}{n\log_2 n}\right)$$

$$= \lim_{n\to\infty}\left(\frac{4}{\log_2 n}\right) + \lim_{n\to\infty}(2)$$

$$= (4)\underbrace{\lim_{n\to\infty}\left(\frac{1}{\log_2 n}\right)}_{0} + \lim_{n\to\infty}(2)$$

$$(4)\;0 + 2 = 2$$

$\therefore g(n) \in \Theta(f(n))$ is True.



==========

Q3: To prove the correctness of Merge Sort using induction, we will use the same approach we did for the insertion sort. The algorithm Merge Sort is a divide-and-conquer solution for sorting an array. The algorithm works in three simple steps:

1. Divide: Divide an array of n elements into two arrays of n/2 elements each.
2. Conquer: Sort the two arrays recursively.
3. Combine: Merge the two sorted arrays.

Prove by induction consists of two main cases:

1. Base Case: consider an array of 1 element (which is the base case of Merge Sort algorithm). An array with one element is already sorted, so the base case is correct.

2. The induction step: Assume that the algorithm works correctly for all integral values of n not larger than k, where k >= 1 is an integer. We will need to show that the algorithm works correctly for n = k +1. So, Merge Sort can correctly sort n=1, 2, ..., k elements. Let's suppose that we call the Merge Sort on an array of size n. It will then recursively call Merge Sort on two subarrays of size n/2.

MergeSort(A, left(L), right(R)) {

```
  if (left < right) {  //line1
```

```
    mid = floor((left + right) / 2); //line2
    MergeSort(A, left, mid); //line3
    MergeSort(A, mid+1, right); //line4
    Merge(A, left, mid, right); //line5
  }
}
```

When n = k +1, lines 1-5 will be execute in the algorithm because k >= 1 meaning that n>1. This indicate that R – L +1 > 1, thus Left < Right is true in line 1. Therefore, the algorithm will split the inputs of the two subarrays A [L](left side) and A[R](right side) into two parts A[L: m] and A[m+1: R], where m = mid = ((left +right)/2)

Then line 3 recursive call will sort the left part A [L: m] using the induction step we discussed (which is size of the subarray/2). The same will be done for the recursive call that sorts the right part A [m+1: R]. Thus, since the 2 parts are sorted, and by assuming that Merge algorithm can merge two sorted arrays into one. Then, the Merge call in line 5 will correctly merges both the left and the right parts of the subarrays into a single sorted one.

Hence, the algorithm works correctly for n = k+1 and it is proved using induction.

Next. Rewriting the correctness of Merge Sort algorithm from the book.

```
Merge (A, p, q, r)
  n1 = q – p + 1                                          //line 1
  n2 = r – q                                              //line 2
  let L[1 .. n1 + 1] and R[1 .. n2 + 1] be new arrays     //line 3
  for i = 1 to n1                                         //line 4
     L[i] = A[p + i – 1]                                  //line 5
  for j = 1 to n2                                         //line 6
     R[i] = A[q + j]                                      //line 7
  L[n1 + 1] = ∞                                           //line 8
  R[n2 + 1] = ∞                                           //line 9
  for k = p to r                                          //line 10
    if L[i] <= R[j]                                       //line 11
         A[k] = L[i]                                      //line 12
         i = i + 1                                        //line 13
    else A[k] = R[j]                                      //line 14
         j = j + 1                                        //line 15
```

Loop Invariant:

At the beginning of each iteration of Merge lines 12-17, we will have:

1. The subarray A[p … (k-1)] consists of the k-p smallest element of L[1 … (n_1 + 1)] and R[1 … (n_2 + 1)] in sorted order.

2. The smallest elements of their arrays are L[i] and R[j] and these elements values are yet to be copied back into A.

Initialization:
After 1st iteration, we will get k=p. Thus, the subarray A[p …(k-1)] with length k – p = 0 is set to empty and consists of 0 smallest element of L to R (by trivial). Also, by definition, L [1] and R [1] are the smallest element in their respective arrays.

Maintenance:
Two cases to maintains:
1. L[i] <= R[j] (After line 14), where the smallest element that is yet to be copied to A (in this case L[i] is the smallest element that is being referred to) will be copied into A[k]. This means that A[p … k] consists of the k-p+1 smallest element of L and R.
2. L[i] > R[j] (After line 16), where the smallest element that is yet to be copied to A (in this case R[i] is the smallest element that is being referred to) will be copied into A[k]. This means that A[p … k] consists of the k-p+1 smallest element of L and R.
Additionally, when L[i] <= R[j] happen, the value of (i) will increase by 1, in which L[i] will still be the smallest element on the left(L) that is yet to be copied to A and Vice versa for L[i] > R[j].

Termination:
At termination, k = r +1, which means that A[p … r+1] consists of r-p+1 smallest element of left(L) and right(R), which are indeed the entire elements that we wanted to sort.

Lastly, we need to consider that the sum of the length of L and R is calculated as n_1= q-p+2 and n_2 = r-q+1. Therefore, we can find the sum of the elements using $n_2 + n_1$ = r-p+3, which prove that all the elements of L and R combined expect the largest 2 have already been moved/copied to A. Although, these last 2 large elements are not copied, this doesn't mean that our algorithm is not good because these 2 elements have a value of infinity (one is added to L and the other in R). They are added in lines 8-9 just to ensure that we never run out of elements, which will be a problem since we are comparing L and R and they might not have the same input length. Thus, these 2 infinities are added to L and R as a safe measurement but they are not going to be copied into A