

Christopher Tam - Assignment 1

Student Number: 250803892

February 5, 2019

Question 1

- a. Suppose we have n/k sublists, each of length k . Performing insertion sort on each of the sublists will take k^2 time per sublist in the worst case. Since there are n/k sublists to perform insertion sort on, it will take $k^2 * n/k$ time to sort, which simplifies to $O(nk)$ time in the worst case.
- b. Having coarsened the leaves of the recursion by using insertion sort within the merge sort algorithm, we may begin the usual merge sort procedure starting at the level in which each array has a size of at most k . The depth of the merge tree is thus $\lg(n) - \lg(k) = \lg(n/k)$. The time it takes for each level to merge is cn , so in total the merge sort algorithm augmented with insertion sort takes $O(n \lg(n/k))$ worst case time.
- c. In order for $O(nk + n \lg(n/k)) = O(n \lg(n))$, either $nk = O(n \lg(n))$ or $n \lg(n/k) = O(n \lg(n))$. We know that the larger of these two, in terms of asymptotic complexity for k , is $O(\lg n)$. Therefore, while $k(n) \in O(\lg(n))$ it will have the same asymptotic complexity as a standard merge sort.
- d. To determine the value of k in practice, we should calculate the running time expressions with proper values for the constant factors with different k values until a sufficient one is found.

Question 2

- a. O - yes, o - yes, Ω - no, ω - no, Θ - no
- b. O - yes, o - yes, Ω - no, ω - no, Θ - no
- c. O - no, o - no, Ω - no, ω - no, Θ - no
- d. O - no, o - no, Ω - yes, ω - yes, Θ - no
- e. O - yes, o - no, Ω - yes, ω - no, Θ - yes
- f. O - yes, o - no, Ω - yes, ω - no, Θ - yes

Question 3

a)

1. $T(n) = T(n/2) + \Theta(1)$. Solving the recurrence relation via the master method we get $T(n) = \Theta(\lg n)$.

2. $T(n) = T(n/2) + \Theta(n)$. Solving the recurrence relation we get $T(n) = \Theta(n \lg n)$.

3. $T(n) = T(n/2) + \Theta(n/2)$. Solving the recurrence relation via the master method we get $T(n) = \Theta(n)$.

b)

1. $T(n) = 2T(n/2) + cn$. Solving the recurrence relation via the master method we get $T(n) = \Theta(n \lg n)$.

2. $T(n) = 2T(n/2) + cn + 2\Theta(n)$. Solving the recurrence relation we get $T(n) = \Theta(n \lg n) + \Theta(n^2) = \Theta(n^2)$.

3. $T(n) = 2T(n/2) + cn + 2(c'n/2)$. Solving the recurrence relation we get $T(n) = \Theta(n \lg n)$.

Question 4

See last page for recurrence tree. The following is the proof for the time complexity of the program:

*Note: All log values are log base 2.

$$T(2) = 1$$

$$T(n) \leq 2T(n/2) + n \log(n)$$

$$T(n) = 2T(n/2) + n \log n$$

$$T(n) = 4T(n/4) + n \log n + n \log(n/2)$$

$$T(n) = 8T(n/8) + n \log(n/2) + n \log(n/4)$$

$$T(n) = 16T(n/16) + n \log(n) + n \log(n/2) + n \log(n/4) + n \log(n/8)$$

$$T(n) = \dots$$

$$T(n) = n/2 + \sum_{i=0}^{\log-2} n \log(n/2^i)$$

$$\text{Therefore, } T(n) = O(n(\log(n))^2)$$

Question 5

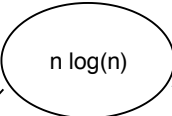

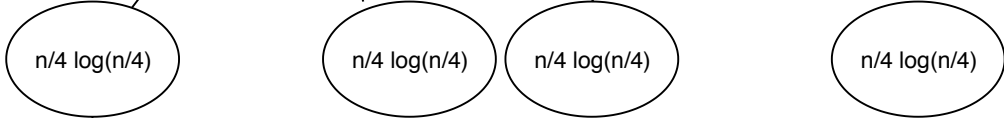
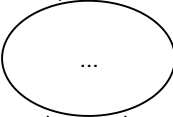
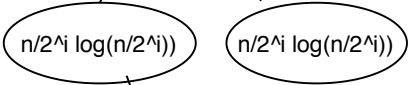

a) See asn1-a shell script file.

b) See asn1-b shell script file.

c) See asn1-c shell script file.

d) We do not want to run a) on input size of 200,000,000 given the worst-case time complexity of insertion sort. Since we designed our input array to be reversely sorted, the algorithm would take the worst-case time to correctly sort the array. In comparing the sorting algorithms in b) and c), the best value for k is when $k = 16$. This is because at length 16, insertion sort finishes faster than merge sort.

(Recurrence Tree for Q4)

Level	Tree	# Nodes	Cost
0		1	$n \log(n)$
1		2	$2(n/2) \log(n/2)$
2		4	$4(n/4) \log(n/4)$
.		.	.
.		.	.
.		.	.
i		2^i	$2^i(n/2^i) \log(n/2^i)$
h		.	.