CS 3340 Assignment 1 - Nicholas Gardi – 250868721 – 2/1/19

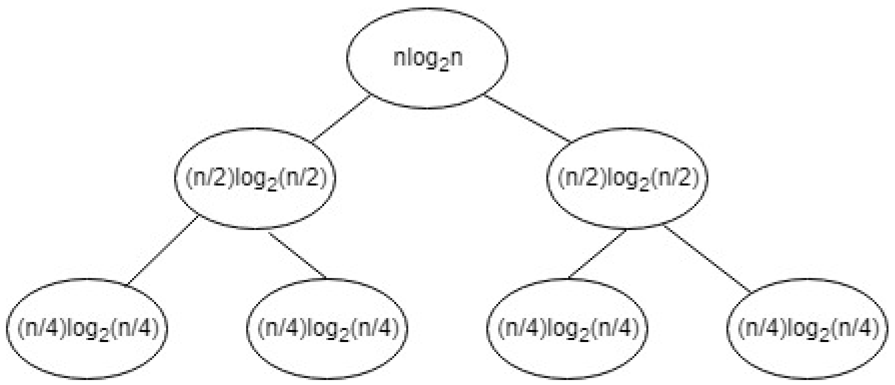
1.
   a. Time for insertion sort to sort a single list with length k is $\Theta(k^2)$. This means that n/k of them will take $\Theta(n/k * k^2) = \Theta(nk)$
   b. Coarseness = k. Start by using the usual merging procedure but start it at the level in which each array has a maximum size of k. Then the depth of the merge tree is log(n) – log(k) = log (n/k). Merging is still cn for each level to the total merging time is $\Theta(n \log(n/k))$.
   c. As long as $k(n) \in O(\lg(n))$, then for any choice of k the outcome will be the same.
   d. To find the best choice for k in practice we could try and time various values and choose the best one.

2.

A	B	O	o	Ω	ω	θ
$\lg^k n$	n^ϵ	Yes	Yes	No	No	No
n^k	c^n	Yes	Yes	No	No	No
$\sqrt{n}$	$n^{\sin n}$	No	No	No	No	No
2^n	$2^{n/2}$	No	No	Yes	Yes	No
$n^{\log c}$	$c^{\log n}$	Yes	No	Yes	No	Yes
Log(n!)	$\log(n^n)$	Yes	No	Yes	No	Yes

3.
   a.
      i. T(n) = T(n/2) + Θ(1). Solving the recursion: T(N) = Θ(lg N)
      ii. T(n) = T(n/2)+ Θ(N). Solving the recursion: T(N) = Θ(N lg N)
      iii. T(n) = T(n/2) + Θ(n/2). Solving the recursion: T(N) = Θ(N)
   b.
      i. T(n) = 2T(n/2) + cn. Solving the recursion: T(N) = Θ(N lg N)
      ii. T(n) = 2T(n/2) + cn + 2 Θ(N). Solving the recursion: T(N) = Θ(N lg N) + Θ(N2) = Θ(N2)
      iii. T(n) = 2T(n/2) + cn + 2c 0n/2. Solving the recursion: T(N) = Θ(N ln N)

4. COST:



$n\log_2(n)$

$n\log_2(n/2)$

$n\log_2(n/4)$

If there was one more row in the tree representing level i, the nodes would contain $(n/i^2)\log_2(n/i^2)$ and the cost would be $2i * n\log_2(n/i^2)$. i is a constant, therefore the time complexity is $n\log_2(n)$.

5d. The merge sort in b is much faster than the insertion sort in a. If we ran a on an input size of 20000000 then it would take a very long time to complete. On a size of 20000, b took less than a second while a took about 26 seconds. The time complexity is better for merge sort than it is for insertion sort but insertion can be faster on small data sets as it is not making additional arrays. I found the sweet spot to be a k of 32, which was about 15 seconds faster than b. Overall, c was faster than b.