Algorithm HW1 Sanford

Problem 3

To arrange these functions, we need to understand how each of them grows as n becomes very large. Big O notation can be used to classify their growth rates

- $n^2 \log_2 n$ (a): The function grows faster than n^2 but slower than any exponential function.
- 2^n (b): This is an exponential function and grows very fast.
- 2^{2^n} (c): This is a double expoential function, which grows even faster than an expoential function
- $n^{\log_2 n}$ (d): This grows slower than n^2 but faster than $n \log n$ since $\log_2 n$ is less than n for $n > 2$
- n^2 (e): Is a polynomial function, which grows faster than any polynomial function of degree less than 2 but slower than $n^2 \log_2 n$

arranging from slowest to fastest

1. n^2
2. $n^2 \log_2 n$
3. $n^{\log_2 n}$
4. 2^n
5. 2^{2n}

Samford

Algorithm HW1

problem 5

The given recurrence is:

$T(n) \leq 7 \cdot T(\frac{n}{3}) + O(n^2)$

we need to identify the variables a, b, and $f(n)$ from the general form of the Master theorem:

$T(n) = a \cdot T(\frac{n}{b}) + f(n)$

$a = 7$ (the number of sub problems)

$b = 3$ (the facton by which the subproblem size is reduced)

$f(n) = n^2$ (the cost of dividing the problem and combining the results)

The Master theorm provides the asymptotic behavior of $T(n)$ based on the comparison between $f(n)$ and $n^{\log_b a}$:

1: If $f(n) = O(n^{\log_b a - \epsilon})$ for some $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.

2: If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$.

3: If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some $\epsilon > 0$ and if $af(\frac{n}{b}) \leq cf(n)$ for some $c < 1$ and sufficiently large n, then $T(n) = \Theta(f(n))$.

Calculate $n^{\log_b a} = n^{\log_3 7}$ (used Chat GPT)

The value of $\log_3 7$ is approximately 1.77, which is less than 2. So we can see that $f(n) = n^2$ grows faster than $n^{\log_3 7}$, which implies we're in the 3rd case for Master Theorm.

The third case dicates that if $f(n)$ is polynomially larger than $n^{\log_b a}$ which is because n^2 grows faster than $n^{1.77}$), and if the regularity condition $af(\frac{n}{b}) \leq cf(n)$ is satisfied, then $T(n) = \Theta(f(n))$.

To check the regularity condition

$$7 \cdot f(\frac{n}{3}) = 7 \cdot (\frac{n}{3})^2 = \frac{7}{9} \cdot n^2$$

Since $\frac{7}{9} < 1$, the regularity condition $af(\frac{n}{b}) \leq cf(n)$ holds for some $c < 1$

Thus the Master theorm, the smallest correct upper bound on the asympotic running time of the algorithm is:

$$T(n) = \Theta(n)^2$$

Thus the correct answer is (b) $O(n^2)$

Algorithms HW1 Sanford

problem 6

given: $T(n) \leq T(\sqrt{n})+1$

If you apply iteratively to see how recurrence unfolds

1. $T(n) \leq T(\sqrt{n})+1$
2. $T(\sqrt{n}) \leq T(n^{1/4})+1$
3. $T(n^{1/4}) \leq T(n^{1/8})+1$
4. $T(n^{1/8}) \leq T(n^{1/16})+1$
5. ....

- each step the exponent is halved,
- but how many times do we halved until we get to 1 which is the definition of log base 2
- since we're taking square roots, we're halving the exponent at each step, which is equivalent to taking a log of a log

After taking K steps, the size of the problem is $n^{1/2^k}$, and we want this to be 1.

$n^{1/2^k} = 1$

$n = 2^{2^k}$

$\log_n(n) = 2^k$

$k = \log_2(\log_2(n))$

Adding up all the costs at each level of the recursion until the base case $T(1)$ is reached, we get: $T(n) \leq \log_2(\log_2(n)) + c$

for some constant c, where each level contributes a cost of 1 since at each recusive call we add 1).

This gives us a total running time of:

$O(\log_2(\log_2(n)))$

Thus the smallest correct upper bound on the asymptotic running time of the algorithm is: $O(\log \log n)$