Quiz 7 – Algorithm Complexity

1. Determine whether each of the following functions is $O(x)$, $\Omega(x)$, or both (i.e., $\Theta(x)$):
   (a) $f(x) = 10$
   $O(x)$
   (b) $f(x) = 3x + 7$
   $\Theta(x)$
   (c) $f(x) = x^2 + x + 1$
   $\Omega(\mathrm{x})$

2. Using the tree method *or* the Master Theorem, solve the following recurrence relations:
   (a) $T(n) = 3T(n/3) + O(n)$

Master Theorem:

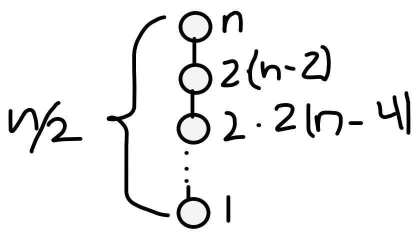
a = 3, b = 3, d = 1

$\log_b a = \log_3 3 = 1 = d$

→ $O(n \log n)$

(b) $T(n) = 2T(n-2) + O(1)$

Tree Method:



$\mathrm{T(n)} = \sum_{i=0}^{\frac{n}{2}-1} 2^{\wedge} i = 2^{n/2-1} - 2^0 = 2^{n/2-1} - 1 = (2^{1/2})^n - 1$

→ $O((\sqrt{2})^n)$

3. Give the smallest Big-O estimate for each of the following functions:
   (a) $(n! + 2^n)(n^3 + \log_2 n\,(n^2 + 1))$
   $O(n^3\, n!)$
   (b) $(n^3 + n^2 \log_2 n)(\log_2 n + 1) + (17 \log_2 n + 19)(n^3 + 2)$
   $O(n^3 \log n)$

4. Consider the following recursive algorithm:

$\textsc{Power}(x, n)$

Input: A real number x and a positive integer n
Output: x^n

if $n = 1$ **then**
  return x
else if n is even **then**
  return $\textsc{Power}(x, n/2) \cdot \textsc{Power}(x, n/2)$
else
  return $x \cdot \textsc{Power}(x, \lfloor n/2 \rfloor) \cdot \textsc{Power}(x, \lfloor n/2 \rfloor)$

Set up and solve a recurrence relation giving a Big-*O* estimate for the complexity of this algorithm.

T(n) = 2*T(n/2) + O(1)

By Master Theorem:

a = 2, b = 2, d = 0

$\log_a b = \log_2 2 = 1 > d$

→ $O(n^{\log_2 2})$

→ $O(n)$