1.)

$2, 52!, \ln^4(n^3), \ln^2(n), \frac{1}{n^3}, \ln(n), \log_2((4n)^n), n\ln(n), n^3, (\frac{3}{2})^n, 2^n, n!$

2 and 52! are constant, same Θ class. $\ln(n^2)$ and $\ln^2(n) = \Theta$.

and $n\ln(n)$ and $\log_2((4n)^n) = \Theta$

2.)

```
List<elem> SortFunction(A):
    if A.length() ≤ 1
        return A

    p = A.last()
    List<elem> L, R
    for elem e in A (besides last)
        if(e ≤ p
            L.Push(e)
        else
            R.Push(e)

    L' = Sort(L)
    R' = Sort(R)

    Return L' + [p] + R'
```

2B.) worst case is $O(n^2)$, if our pivot is an extremely large or small elem, then one list will be way larger, or contain all the elements. Then we would have each call sorting $n-1, n-2, \ldots$ elem each call.

2C) $T(n) = n + T(\frac{n}{2}) + T(\frac{n}{2})$

$$T(n) = n + 2T(\frac{n}{2})$$

$$T(n) = \Theta(n) + 2T(\frac{n}{2})$$

2D) $T(n) = aT(\frac{n}{b}) + f(n)$

$\rightarrow \log_b a \rightarrow \log_2 2 \rightarrow 1 \rightarrow n^1 \rightarrow n^1 = f(n) \rightarrow n = n$

$\rightarrow \Theta(n \log^k(n)) \rightarrow \Theta(n \log^1(n)) \rightarrow \Theta(n \log^2(n))$

$= \Theta(n \log(n))$

3A.) A is the winners essentially Since "no-one outside this group can reach them. So IF I've lost to you, You have lost to me: A. is also Someone who can Push around at a minimum one player without them winning back. Implies an Scc has an outgoing edge.

B. is the opposite kinda, These are losers the 12th place Diddy Kong Bike players. Someone can beat them (w → u) but I can't beat them (u → w). A Scc for losers.

C. Is the mid tier, They are in Some Scc and can reach other Scc's but not consistently getting 1st or 12th

3B) Each Set's size can vary on the number of players and winners, losers, etc. We could have Set A be extremely large but Set B be relatively small implying a Dynamic Data Structure necessary

3B.) Triple<int, int, int> getSetSizes(G)

H = MakeSCC(G)

Int ACount, BCount, CCount = 0

for scc in H

inDeg = scc.incoming

outDeg = scc.outgoing

cSize = scc.Size

if inDeg = 0 and outDeg > 0

ACount = cSize

elseif outDeg = 0 and InDeg > 0

BCount = cSize

else if

CCount = cSize

Return Triple<int, int, int> sizes = (ACount, BCount, CCount)

Return sizes

3C) $\Theta(|V|+|E|)$, I assume if we run a Faire Kosaraju's then we do some work it will be dependent on the size of our graph.