**CS433 HW1**Jed Pulley & Matthew Star **Complexity Analysis**Q1)  
There are n/k groups because dividing the total elements in the array over the size of the groups gives us the number of groups. We are running mergesort n/k times. Mergesort runs in O(n log n) time.  
Therefore, O(n/k \* n log n) => **O(n2/k \* log n)**

Q2)

1. **O(n2) in the worst case**. First loop runs n times. Second loop runs n times.
2. Mergesort A => O(n log n). binary search (log n) each in B until n. n \* log n + n \* log n = 2(n \* log n). Drop constant => **O(n log n) in the worst case**

**Growth of Functions**

Q3)

1. n3 + 4n2 log n + 16n = Θ(n3)   
   *Proof.* Let f(n) = n3 + 4n2 log n + 16n and g(n) = n3. Choose n0 = 1.  
   If C0 = 20, then C0 \* g(n) = 20n3 >= n3 + 4n2 log n + 16n = f(n) for any n >= 1.  
   If C0 = 16, then C0 \* g(n) = 16n3<= n3 + 4n2 log n + 16n = f(n) for any n >= 1.
2. If f1(n) = O(g1(n)) and f2(n) = O(g2(n)), then f1(n) ∗ f2(n) = O(g1(n) ∗ g2(n)), where all the functions are positive  
   *Proof.* By definition, f1(n) = O(g(n)) implies that there exists two constants c1 and n1 such that f1(n) ≤ c1 ∗ g1(n) for every n ≥ n1. By definition, f2(n) = O(g(n)) implies that there exists two constants c2 and n2 such that f2(n) ≤ c2 ∗ g(n) for every n ≥ n2.

Since f1(n) ≤ c1 ∗ g(n) and f2(n) ≤ c2 ∗ g2(n) hold for n ≥ n1 and n ≥ n2 respectively, both must hold for n ≥ max{n1, n2}. Then,

f1(n) ∗ f2(n) ≤ c1 ∗ g1(n) ∗ c2 ∗ g2(n)

f1(n) ∗ f2(n) ≤ c1 ∗ gmax(n) ∗ c2 ∗ gmax(n)  
f1(n) ∗ f2(n) ≤ (c1 ∗ c2) ∗ gmax(n)

for every n ≥ max{n1, n2}.

1. If f(n) = Θ(g(n)) and g(n) = Θ(h(n)), then f(n) = Θ(h(n)) *Proof.* By definition, f(n) = Θ(g(n)) implies the existence of three constants, c1, c2, and n1 such that c1 ∗ g(n) ≥ f(n) ≥ c2 ∗ g(n) for every n ≥ n1. By definition, g(n) = Θ(h(n)) implies the existence of three constants c3, c4, and n2 such that c3 ∗ h(n) ≥ g(n) ≥ c4 ∗ h(n) for every n ≥ n2.

Since c1 ∗ g(n) ≥ f(n) ≥ c2 ∗ g(n) and c3 ∗ h(n) ≥ g(n) ≥ c4 ∗ h(n) hold for n ≥ n1 and n ≥ n2 respectively, both must hold for n ≥ max{n1, n2}. Then,

c1 ∗ g(n) ≥ f(n) ≥ c2 ∗ g(n)

c1 ∗ c3 ∗ h(n) ≥ f(n) ≥ c2 ∗ c4 ∗ h(n)

for every n ≥ max {n1, n2}. Hence, f(n) = Θ(h(n))

**Master Theorem**

Q4)

1. T(n) = 625 ∗ T(n/25) + Θ(n1.75)   
   a = 625  
   b = 25  
   C = log25 625 = log25 252 = 2 log25 25 = 2  
   E = 1.75  
   C > E so Case 1: T(n) = Θ(nC) => **Θ(n2)**
2. T(n) = 43.5 ∗ T(n/4) + Θ(n3.5 log2 n)  
   a = 43.5  
   b = 4  
   C = log443.5 = 3.5 log4 4 = 3.5  
   E = 3.5  
   C = E so Case 3: T(n) = Θ(f(n) log n) => **Θ(n3.5 log3 n)**
3. T(n) = 5 ∗ T(n/25) + Θ(n)  
   a = 5  
   b = 25  
   C = log25 5 = 0.5  
   E = 1  
   C < E so Case 2: T(n) = Θ(f(n)) => **Θ(n)**

**Analyzing Recursive Algorithms**

Q5) T(n) = 3 \* T(2/3n) + Θ(1)

a = 3  
b = 3/2 = 1.5  
C = log1.5 3 = 2.7 (rounded)  
E = 0  
C > E so Case 1: T(n) = Θ(nC) => **Θ(n2.7)**

Q6) T(n) = T(n/3) + Θ(1)

a = 1  
b = 3  
C = log31 = 0  
E = 0  
C = E so Case 3: T(n) = Θ(f(n) log n) => **Θ(log n)**

Q7) T(n) = T(2/3n) + Θ(n)  
  
a = 1  
b = 3/2 = 1.5  
C = log1.5 1 = 0  
E = 1  
C < E so Case 2: T(n) = Θ(f(n)) => **Θ(n)**