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

a = 1, b = 2, f(n) = n

n^(logb(a)) = n^(log2(1)) = 1

**This is case 3 of the master’s theorem**

**T(n) = O(n)**

1. **T(n) = 2nT(n/2) + n**

n^(logb(a)) = n^(log2(2n)) = n^(1 + log2(n)) > 1

**This is case 1 of the master’s theorem**

**T(n) = O(n^(log22n))**

1. **T(n) = 5T(n/2) + n**

a = 5, b = 2, f(n) = n, c = 1

n^(logb(a)) > c = n^(log2(5)) > 1

**This is case 1 of the master’s theorem**

**T(n) = O(n^(log2(2n)))**

1. **T(n) = T(n) + c**

b = 1

**Master’s Theorem doesn’t apply, not a valid recursive equation in time complexity, we need b > 1**

1. **T(n) = 4T(n/4) + O(1)**

a = 4, b = 4, f(n) = O(1) which is roughly n^0, and c = 0

n^(logb(a)) = n^(log4(4)) = 1 > 0

**This is case 1 of the master’s theorem**

**T(n) = O(n)**

1. **T(n) = 3T(n/4) + 1**

a = 3, b = 4, f(n) = 1

1 = n0 c = 0

f(n/2) < c\*n therefore O(1)

n^(log43) < 0

**This is case 3 of the master’s theorem**

**T(n) = O(1)**

1. **T(n) = 4T(n/2) + n log n**a = 4, b = 2, f(n) = O(nclogxn) where c = 1 and x = 1  
   **This is case 2 of the master’s theorem**  
   **T(n) = O(n1 log2n) = O(n log2n)**
2. **T(n) = 4T(n/2) + n2**a = 4, b = 2, f(n) = n2  
   log2(4) = 2 = c  
   **This is case 2 of the master’s theorem**  
   **T(n) = O(n2 logn)**
3. **Master’s theorem cannot be applied to T(n) = aT(n/b) + f(n) f(n) = 2n and 2^n is not a polynomial function of n**
4. **T(n) = T(n/3) + T(n/4) + n**  
     
     
     
     
     
     
     
     
     
     
     
     
     
     
     
     
     
     
     
     
     
     
     
     
     
   = n + n/3 + n/4 + n/3(1/3 + 1/4) + n/13(1/3 + 1/4)  
   + n/3[1/3(1/3 + 1/4) +1/4(1/3 + 1/4)]  
   + n/4[1/3(1/3 + 1/4) +1/4(1/3 + 1/4)] …  
   = n[1 + (1/3 + 1/4) + (1/3 + 1/4) + … + infinity]  
   = n/(1 – 1/3 – 1/4) infinity with alpha = (1/3 + 1/4)

**= 12n/5  
= O(n), master’s theorem doesn’t work here**

1. **T(n) = T(n/2) + 2^n**a = 1, b = 2, f(n) = 2n (or N as I defined N = 2n), c = 1 since log2(1) = 0 < c (which = 1)starts at 2n at the top element of the tree and then branches down to 2n/2, then branches down to 2n/4 … T(n) = 2n + 2n/2 + 2n/4 + … infinity  
   T(n) = T(n/2) + 2n  
     
   Substitute N = 2n  
   T(log2N) = T((log2N)/2) + N  
   U(N) = T(log2N)  
   U(N) = U(N/2) + N  
   **This is case 3 of the master’s theorem.  
   This is O(N), where N = 2n for the given equation**
2. **T(n) = T(n-4) + O(1)**T(n) = O(1) + O(1) + … + (n/4) times  
    = n/4 O(1)  
   = **O(n), master’s theorem doesn’t work here**
3. s = 1, t = n  
   In the worst case, we keep on searching 3/4th part of the array, and the element doesn’t exist at all  
   T(n) = O(1) + T(3n/4)  
   b = 4/3, a = 1, c = 0  
   log4/31 = 0 = c  
   **O(n0logn) =** **O(logn), which is case 2 of master’s theorem**
4. **T(n) = O(1) + T(2n) + T(2n + 1)**= (n(n + 1))/2  
   = **O(n2), master’s theorem doesn’t work here**