# Back Substitution

Compute the complexity of the recursive algorithms based on the recursive equation and stop condition. Show your work, not just your final answer.

1. T(n) = 2T(n-1) + 1 and T(0) = 1
   1. You can compute this complexity as a tight upper bound.

**Substitute (n-1) –> k = 2**

T(n-1) = 2T((n-1)-1) + 1

T(n-1) = 2T(n-2) + 1

T(n) = 2[2T(n-2) + 1] +1

**T(n) = 4T(n-2) + 2 + 1**

**Substitute (n-2) –> k = 3**

T(n-2) = 2T((n-2)-1) + 1

T(n-2) = 2T(n-3) + 1

T(n) = 4[2T(n-3) + 1] + 2 +1

**T(n) = 8T(n-3) + 4 + 2 + 1**

**General Formula**

**T(n) = 2kT(n-k) + 2k-1 + 2k-2 + … + 21 + 20**

**Stop Condition**

T(0) = 1 🡪 n-k = 0 🡪 **k = n**

T(n) = 2nT(n-n) + 2n-1 + 2n-2 + … + 21 + 20

T(n) = 2nT(0) + 2n-1 + 2n-2 + … + 21 + 20

**T(n) = 2n + 2n-1 + 2n-2 + … + 21 + 20 = 2n+1 – 1**

**Big-Oh Complexity**

**O(2n)**

1. T(n) = T(n-2) + n2 and T(0) = 1
   1. Hint: Assume n is even; that is, n = 2k for some integer k.

**Substitute (n-2) –> k = 2**

T(n-2) = T((n-2)-2) + (n-2)2

T(n-2) = T(n-4) + (n-2)2

**T(n) = T(n-4) + (n-2)2 + n2**

**Substitute (n-4) –> k = 3**

T(n-4) = T((n-4)-2) + (n-4)2

T(n-4) = T(n-6) + (n-4)2

**T(n) = T(n-6) + (n-4)2 + (n-2)2 + n2**

**Substitute (n-6) –> k = 4**

T(n-6) = T((n-6)-2) + (n-6)2

T(n-6) = T(n-8) + (n-6)2

**T(n) = T(n-8) + (n-6)2 + (n-4)2 + (n-2)2 + n2**

**General Formula**

**T(n) = T(n-2k) + (n-2(k-1))2 + (n-2(k-2))2 + (n-2(k-3))2 + … + (n-2)2 + n2**

**Stop Condition**

T(0) = 1 🡪 n-2k = 0 🡪 **k = n/2**

T(n) = T(n-2()) + (n-2(()-1))2 + (n-2(()-2))2 + (n-2(()-3))2 + … + (n-2)2 + n2

T(n) = T(n-n) + (n-n+2)2 + (n-n+4))2 + (n-n+6))2 + … + (n-2)2 + n2

T(n) = T(0) + (2)2 + (4)2 + (6)2 + … + (n-2)2 + n2

T(n) = 1 + 2

**T(n) = 1 + 42 = 1+ = 1 + = 1 +** (sum of squares of first n even numbers

**Big-Oh Complexity**

**O(n3)**

# 3.T(n) = T(n-1) + 1/n and T(1) = 1

a. Hint: Go online and find a formula for the sum of the first n terms of the

“harmonic series”.

**Substitute (n-1) –> k = 2**

T(n-1) = T((n-1)-1) +

T(n-1) = T(n-2) +

**T(n) = T(n-2) + +**

**Substitute (n-2) –> k = 3**

T(n-2) = T((n-2)-1) +

T(n-2) = T(n-3) +

**T(n) = T(n-3) + + +**

**General Formula**

**T(n) = T(n-k) + + + + … + +**

**Stop Condition**

T(1) = 1 🡪 n-k = 1 🡪 **k = n - 1**

T(n) = T(n-(n-1)) + + + + … + +

T(n) = T(1) + + + + … + +

T(n) = 1 + + + + … + + **= ln(n) + γ (Sum of first n terms of harmonic series)**

**Big-Oh Complexity**

* **γ = Euler-Mascheroni constant ≈ 0.58**
  + **Ignore for Big-Oh complexity**

**O(ln(n))**

## Master Method

Compute the complexity of the recursive algorithms based on the recursive equation and stop condition. Show your work, not just your final answer.

1. T(n) = 2T(n/4) + 1 and T(0) = 1
   1. Be sure to rewrite 1 as n0.

**Variables**

**a = 2**

**b = 4**

**f(n) = n0**

**Compare f(n) to nd**

**nlogba = nlog42 = n½**

**n0 < n½**

**Big-Oh Complexity**

**T(n) = O(nlogba) = O(n1/2)**

1. T(n) = 2T(n/4) + n1/2 and T(0) = 1
   1. Note that n1/2 is the square root of n.

**Variables**

**a = 2**

**b = 4**

**f(n) = n1/2**

**Compare f(n) to nd**

**nlogba = nlog42 = n½**

**n1/2 = n½**

**Big-Oh Complexity**

**T(n) = O(nlogbalog(n)) = O(n1/2logn)**

# 6.T(n) = 2T(n/4) + n2 and T(0) = 1

a. This is similar to the previous one.

**Variables**

**a = 2**

**b = 4**

**f(n) = n2**

**Compare f(n) to nd**

**nlogba = nlog42 = n½**

**n2 > n½**

**Big-Oh Complexity**

**T(n) = O(f(n)) = O(n2)**

# 7.T(n) = 10T(n/3) + n2 and T(0) = 1

1. In your answer, round the value of the logarithm to 2 decimal places.
2. Remember that the logb(a) is equal to log2 (a) / log2 (b).

**Variables**

**a = 10**

**b = 3**

**f(n) = n2**

**Compare f(n) to nd**

**nlogba = nlog310 = log2(10) / log2(3) = n2.09**

**n2 < n2.09**

**Big-Oh Complexity**

**T(n) = O(nlogba) = O(n2.09)**

# 8.T(n) = 2T(2n/3) + 1 and T(0) = 1

1. In your answer, round the value of the logarithm to 2 decimal places.
2. Be sure to rewrite 1 as n0.
3. Remember that the logb(a) is equal to log2 (a) / log2 (b).
4. Hint: rewrite 2n / 3 as n / (3/2)

**Variables**

**a = 2**

**b = 3/2**

**f(n) = n0**

**Compare f(n) to nd**

**nlogba = nlog1.52 = log2(2) / log2(1.5) = n1.71**

**n0 < n1.71**

**Big-Oh Complexity**

**T(n) = O(nlogba) = O(n 1.71)**