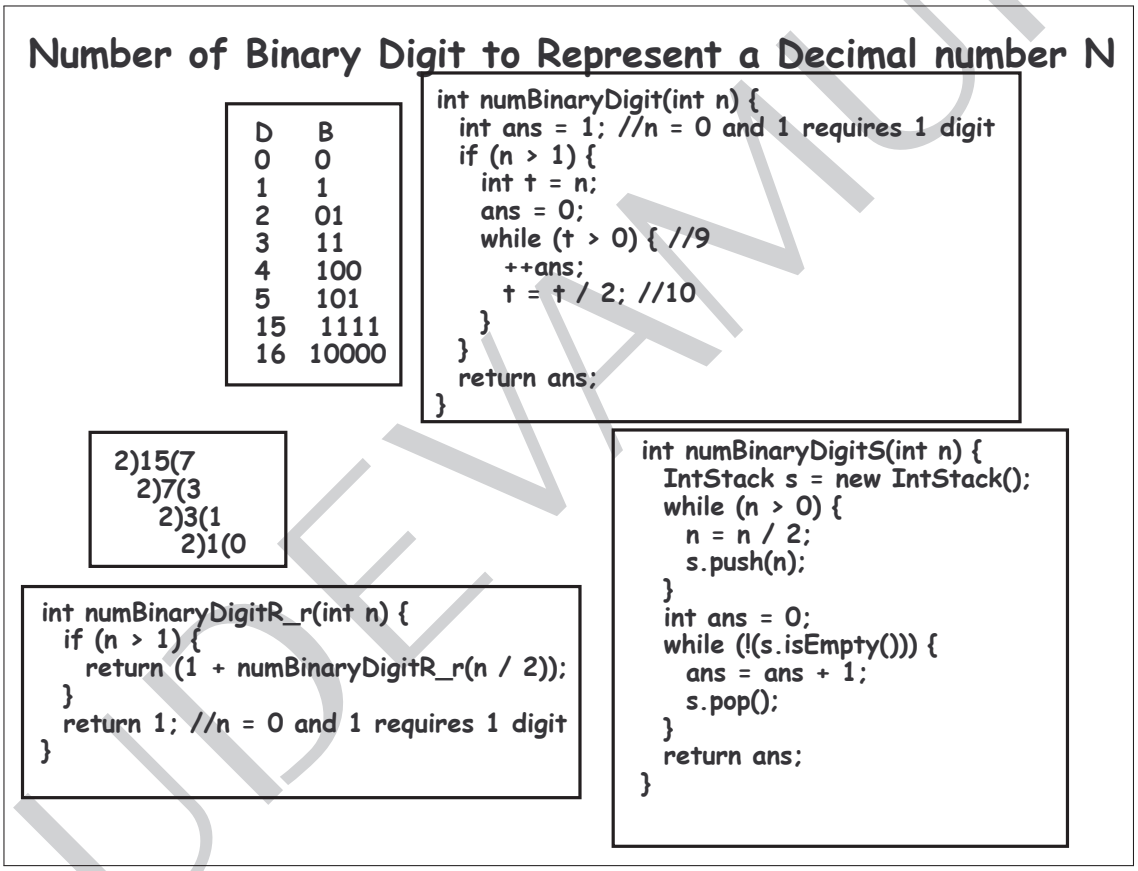

Figure 3.8: Computing number of binary digit using recursion

3.6.2 Complexity of computing number of binary digits to represent a decimal number

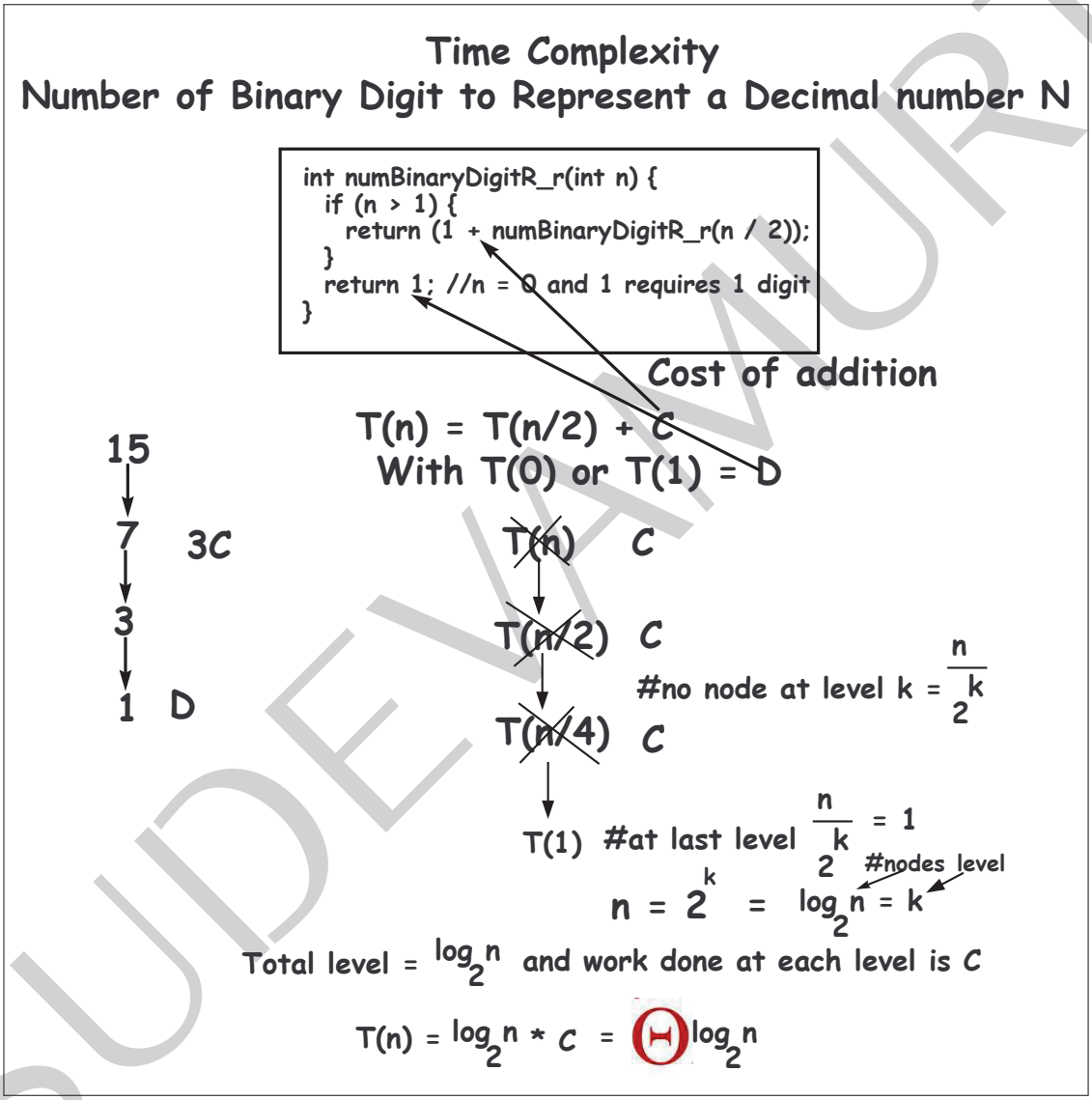

Figure 3.9: Computing T(n)

3.6.3 Computing number of binary digits to represent a decimal number using tail recursion

```
private static int numBinaryDigitTR_r(int n, int ans) {
    incKount() ;
    if (n > 1) {
      return numBinaryDigitTR_r(n/2,ans+1);
    }
    return ans ; //n = 0 and 1 requires 1 digit
}

private static int numBinaryDigitTR(int n) {
    setKount() ;
    return numBinaryDigitTR_r(n,1);
}
```

```
private static int numBinaryDigitTRS(int n) {
    setKount() ;
    IntStack s = new IntStack();
    int t = n ;
    int ans = 1 ;
    s.push(ans) ;
    while (n > 1) {
      ans = ans + 1 ;
      s.pop();
      n = n/2;
      s.push(n) ;
    }
    setKount(s.maxStackSize());
    System.out.println("numBinaryDigitTRS " + t +
        ": = " + ans + " max Stack used = " + getKount()) ;
    return ans ;
}
```

```
numBinaryDigitTRS 0: = 1 max Stack used = 1
numBinaryDigitTRS 1: = 1 max Stack used = 1
numBinaryDigitTRS 10: = 4 max Stack used = 1
numBinaryDigitTRS 15: = 4 max Stack used = 1
numBinaryDigitTRS 16: = 5 max Stack used = 1
numBinaryDigitTRS 65535: = 16 max Stack used = 1
numBinaryDigitTRS 65536: = 17 max Stack used = 1
```

Figure 3.10: Computing T(n)

