Clues for Solving Recurrences and Asymptotic Complexity Problems

1. Solve each recurrence, using $O$, $\Omega$, or $\Theta$, whichever is appropriate. Throughout, we assume the base of the logarithm is 2.

   (a) $F(n) = 4F\left(\frac{n}{2}\right) + 5n^2$

       Use the master theorem.

   (b) $f(n) = f(n - 1) + n$

       Anti-derivative method.

   (c) $f(n) = f\left(\frac{n}{2}\right) + f\left(\frac{n}{3}\right) + n$

       Use the Akra Bazzi method (generalized master theorem) Note that $(1/2)^1 + (1/3)^1 = 1/2 + 1/3 = 5/6 < 1$

   (d) $f(n) = f(\sqrt{n}) + 1$

       Substitute $m = \log n$, i.e., $n = 2^m$. Then $\log \sqrt{n} = \frac{1}{2} \log n = m/2$. Let $f(n) = G(m) = G(\log n)$. Then $f(\sqrt{n}) = G(\log \sqrt{n}) = G(\frac{1}{2} \log n) = G(m/2)$. We have the recurrence $G(m) = G(m/2) + 1$

       By the master theorem

       $f(n) = G(m) = \Theta(\log m) = \Theta(\log \log n)$.

   (e) $f(n) = 2f(\sqrt{n}) + \log n$

       Substitute $m = \log n$, then use the master theorem.

   (f) $H(n) = 2H\left(\frac{n}{2}\right) + O(n)$

       Master theorem.

   (g) $g(n) = 2g(n - 1) + 1$

       Let $n = \log m$, and $F(m) = g(n)$. The answer will be exponential.

   (h) $G(n) \geq G(n - 1) + \log n$

       Anti-derivative
(i) \( H(n) \leq 2H(\sqrt{n}) + 4. \)

Substitute \( m = \log n \) which makes \( n = 2^m \). Let \( G(m) = H(n) \). Then \( G(m) = H(2^m) \), and \( G(m/2) = H(2^{m/2}) = H((2^m)^{1/2}) = H(\sqrt{2^m}) = H(\sqrt{n}) \). Finally, \( G(m) \leq 2G(m/2) + 4 \). By the master theorem, \( H(n) = G(m) = O(m) = O(\log n) \).

(j) \( K(n) = K(n - 2\sqrt{n} + 1) + n. \)

Let \( n = m^2 \), i.e., \( m = \sqrt{n} \), and \( G(m) = K(n) = K(m^2) \). Then \( G(m-1) = G(\sqrt{n-1}) = K((\sqrt{n-1})^2) = K(n - 2\sqrt{n} + 1) \). We then have the recurrence \( G(m) = G(m-1) + m^2 \).

Finish up by using the anti-derivative method.

(k) \( F(n) \leq F(\frac{n}{3}) + F(\frac{7n}{10}) + n \)

This is from the BFPRT algorithm.

(l) \( F(n) = 2F(\frac{2n}{3}) + F(\frac{n}{3}) + n \)

The Akra Bazzi method. The exponent you need to find is an integer.

(m) \( f(n) = 1 + f(\log n) \)

None of the methods we’ve discussed cover this one. But I expect you to know it.

2. Write the asymptotic time complexity for each code fragment, giving the answer in terms of \( n \), using \( O, \Omega, \Theta \), whichever is appropriate.

For the first five, replace the inner loop by a statement that increments the counter by the appropriate amount.

(a) for (int i=1; i < n; i++)
   for (int j=i; j > 0; j--)
      cout << "hello world" << endl;

(b) for (int i=1; i < n; i++)
   for (int j=1; j < i; j++)
      cout << "hello world" << endl;

(c) for (int i=1; i < n; i = 2*i)
   for (int j=1; j < i; j++)
      cout << "hello world" << endl;

(d) for (int i=1; i < n; i++)
   for (int j=1; j < i; j = j*2)
      cout << "hello world" << endl;

(e) for (int i=1; i < n; i++)
   for (int j=1; j < n; j = j*2)
      cout << "hello world" << endl;
(f) for (int i=2; i < n; i = i*i)
    cout << "hello world" << endl;

You will need a substitution.

(g) for (int i=1; i*i < n; i++)
    cout << "hello world" << endl;

(h) for (int i=n; i > 1; i = i/2)
    for (int j=1; j < i; j=2*j)
        cout << "hello world" << endl;

A little complicated, but don’t get scared. Hint: substitute $k = \log i$ and $\ell = \log j$.

(i) For this problem, $\text{george}$ is a function which returns an integer. You have no idea what that integer will be.

```cpp
int m = n;
while(m > 0){
    int g = george(m);
    if (g > 0) m = m - g;
    else m = m - 1;
    cout << "hello world" << endl;
}
```

It is common in practice to not know in advance what an input will be, even asymptotically.