Answers to Recurrences and Asymptotic Complexity

1. Give the asymptotic solution to each recurrence, in terms of n, using Θ; except for the last one, problem 2i. For that problem, Use O instead of Θ.

   (a) $F(n) = 4F\left(\frac{n}{2}\right) + 5n^2$
   $A = 4$, $B = 2$, $C = 2$. The coefficient 5 has no aysmptotic effect. $\log_B A = 2 = C$, and therefore
   $F(n) = \Theta(n^2 \log n)$

   (b) $f(n) = f(n-1) + n$
   $\frac{f(n) - f(n-1)}{1} = \frac{n}{1}$
   $f'(n) = n$
   $f(n) = \Theta(n^2)$

   (c) $f(n) = f\left(\frac{n}{2}\right) + f\left(\frac{n}{3}\right) + n$
   Akra-Brazi. $\alpha_1 = \alpha_2 = 1$, $\beta_1 = 1/2$, $\beta_2 = 1/3$, $\gamma = 1$. . $\alpha_1\beta_1^\gamma + \alpha_2\beta_2^\gamma = 5/6 < 1$
   Therefore
   $f(n) = \Theta(n^\gamma) = \Theta(n)$

   (d) $f(n) = f(\sqrt{n}) + 1$
   Substitution. Let $m = \log n$ and $g(m) = f(n)$, then $\log(\sqrt{n}) = \log n/2$.
   Substituting: $g(m) = g(m/2) + 1$. Using the Master theorem $A = 1$, $B = 2$, $C = 0$ and thus $\log_B A = C$, hence $f(n) = g(m) = \Theta(\log m) = \Theta(\log \log n)$.

   (e) $f(n) = 2f(\sqrt{n}) + \log n$
   Substitution. Let $m = \log n$ and $g(m) = f(n)$, then $\log(\sqrt{n}) = \log n/2$.
   Substituting: $g(m) = 2g(m/2) + m$, $A = 2$, $B = 2$, $C = 1$, and $\log_B A = C$. Thus
   $f(n) = g(m) = \Theta(m \log m) = \Theta(\log n \log \log n)$

   (f) $g(n) = 2g(n-1) + 1$
   Substitution. Let $n = \log m$, $m = 2^n$, $g(n) = f(m)$. Then $n - 1 = \log(m/2)$
   Substituting: $f(m) = 2f(m/2) + 1$. Master theorem: $A = 2$, $B = 2$, $C = 0$, $\log_B A = 1 > C$, therefore
   $g(n) = f(m) = \Theta(m^{\log_B A}) = \Theta(m) = \Theta(2^n)$

   (g) $G(n) = G(n-1) + \log n$
   $\frac{G(n) - G(n-1)}{1} = \log n$
   $G'(n) = \log n$
   $G(n) = \Theta(n \log n)$

(h) $H(n) \leq 2H(\sqrt{n}) + 4$ Substitution: $m = \log n$, $G(m) = H(n)$, $G(m/2) = H(surdn)$
Substituting: $G(m) = 2G(m/2) + 4$ Using the Master theorem: $A = B = 2$, $C = 0$
The constant 4 is asymptotically irrelevant. Then
$G(m) = \Theta(m^{\log_B A}) = \Theta(m)$
$H(n) = G(m) = \Theta(m) = \Theta(\log n)$

(i) $K(n) = K(n - 2\sqrt{n} + 1) + n$. Asymptotically, this recurrence is equivalent to $K(n) = K(n - \sqrt{n}) + n$, then $\dfrac{K(n) - K(n - \sqrt{n}}{\sqrt{n}} + \sqrt{n}$. HERE We get $K'(n) = \sqrt{n}$, thus‘
$K(n) = \Theta\left(n^{3/2}\right)$

(j) $F(n) \leq F\left(\frac{n}{5}\right) + F\left(\frac{7n}{10}\right) + n$
This recurrence appears naturally in the median of medians algorithm. Akra-Brazi.
$\alpha_1 = \alpha_2 = 1$, $\beta_1 = 1/5$, $\beta_2 = 7/10$, $\gamma = 1$ and $\alpha_1\beta_1^\gamma + \alpha_2\beta_2^\gamma = 9/10$ which is less than γ.
Thus $F(n) = \Theta(n^\gamma) = \Theta(n)$.

(k) $F(n) = 2F\left(\frac{2n}{3}\right) + F\left(\frac{n}{3}\right) + n$. Akra Brazi.
$\alpha_1\beta_1^\gamma + \alpha_2\beta_2^\gamma = 5/3$ which is more than 1 We need to find a number δ such that $\alpha_1\beta_1^\delta + \alpha_2\beta_2^\delta = 1$ That number is $\delta = 2$, since $2(2/3)^2 + (1/3)^2 = 1$. Thus
$F(n) = \Theta(n^\delta) = \Theta(n^2)$.

(l) $f(n) = 1 + f(\log n)$
This cannot be worked by any of the methods I've shown you. The answer is
$f(n) = \Theta(\log^* n)$. I will explain the function $\log^* n$ in class.

2. Write the asymptotic time complexity for each code fragment,

In each case, I will solve an equivalent integral.

(a)
```
for (int i=1; i < n; i++)
  for (int j=i; j > 0; j--)
    cout << "hello world" << endl;
```
$\int_1^n \int_0^x dydx = \int_0^n xdx = \Theta(n^2)$

(b)
```
for (int i=1; i < n; i++)
  for (int j=1; j < i; j++)
    cout << "hello world" << endl;
```
$\int_1^n \int_1^x dydx = \int_1^n xdx = \Theta(n^2)$

(c)
```
for (int i=1; i < n; i = 2*i)
  for (int j=1; j < i; j++)
    cout << "hello world" << endl;
```
Substitution: $k = \log i$, $m = \log n$, $\log(2 * i) = k + 1$,
$\int_0^m \int_1^{2^x} dydx = \int_0^m 2^x dx = 2^m = n$ Answer: $\Theta(n)$

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

Substitution: $k = \log j$

```
for (int i=1; i < n; i++)
  for (int k=0; k < log i; k++)
    cout << "hello world" << endl;
```

$$\int_1^n \int_1^{logx} dydx = \int_1^n \log x dx = \Theta(n \log n)$$

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

Substitution: $k = \log j$, $m = \log n$

```
for (int i=1; i < n; i++)
  for (int k=log i; k < m; k++)
    cout << "hello world" << endl;
```

$$\int_1^n \int_{\log x}^m dydx = \int_1^n (m - \log x)dx = \int_1^n (\log n - \log x)dx = \Theta(n \log n - n \log n + n) = \Theta(n)$$

(f)
```
for (int i=2; i < n; i = i*i)
  cout << "hello world" << endl;
```

Substitution: $j = \log i$, $m = \log n$

```
for (int j=1; j < m; j = 2*j)
  cout << "hello world" << endl;
```

Substitution: $k = \log j$, $p = \log m$

```
for (int k=0; k < p; k = k+1)
  cout << "hello world" << endl;
```

$\int_0^p dx = p = \log m = \log \log n$ Answer $\Theta(\log \log n)$

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

Substitution: $n = m^2$.

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

```
for (int i=1; i < m; i++)
  cout << "hello world" << endl;
```

Answer: $\Theta(m) = \Theta(\sqrt{n})$

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

Substitution: $k = \log i$, $m = \log n$, $p = \log j$

```
for (int k=m; k > 0; k = k-1)
  for (int p=0; p < k; p=p+1)
    cout << "hello world" << endl;
```

$$\int_0^m \int_0^x dydx = \int_0^m xdydx = \Theta(m^2) = \Theta(\log^2 n)$$

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

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

You can verify that m decreases by at least 1 at each iteration, thus the asymptotic time complexity is $O(n)$.