Susmitha Shailesh

pg 67

4a) This algorithm computes the sum of all the squares of the numbers from 1 to n.

b) multiplication

c) n times → once for each iteration of the for-loop

d) $\Theta(n)$

e) replace the algorithm with

$$S = (n * (n+1) * (2n+1)) / 6$$

This has an efficiency of $\Theta(1)$

pg 76

1a) $x(n) = x(n-1) + 5$ for $n>1$, $x(1) = 0$

$$x(n) = x(n-1) + 5$$

1. $x(n-1) = x(n-2) + 5$

$$x(n) = [x(n-2) + 5] + 5 = x(n-2) + 10$$

2. $x(n-2) = x(n-3) + 5$

$$x(n) = [x(n-3) + 5] + 10 = x(n-3) + 15$$

3. $x(n) = x(n-i) + 5i$ 4. $n-i = 1 \rightarrow i = n-1$

5. $x(n) = x(n-n+1) + 5(n-1) \rightarrow x(1) + 5(n-1) \rightarrow 5(n-1)$

$\boxed{\Theta(n)}$

b) $x(n) = 3x(n-1)$ for $n>1$, $x(1) = 4$

$$x(n) = 3x(n-1)$$

1. $x(n-1) = 3x(n-2)$

$$x(n) = 3[3x(n-2)] = 9x(n-2)$$

2. $x(n-2) = 3x(n-3)$

$$x(n) = 9[3x(n-3)] = 27x(n-3)$$

3. $x(n) = 3^i x(n-i)$ 4. $n-i = 1 \rightarrow i = n-1$

5. $x(n) = 3^{n-1} x(n-n+1) \rightarrow 3^{n-1} x(1) \rightarrow 4 \cdot 3^{n-1}$

$\boxed{\Theta(3^n)}$

c) $x(n) = x(n-1) + n$ for $n > 0$, $x(0) = 0$

$x(n) = x(n-1) + n$

1. $x(n-1) = x(n-2) + (n-1)$

   $x(n) = [x(n-2) + (n-1)] + n = x(n-2) + (n-1) + n$

2. $x(n-2) = x(n-3) + (n-2)$

   $x(n) = [x(n-3) + (n-2)] + (n-1) + n$

3. $x(n) = x(n-i) + (n-(i-1)) + (n-(i-2)) + \dots + n$

4. $n - i = 0 \rightarrow i = n$

5. $x(n) = x(n-n) + (n-(n-1)) + (n-(n-2)) + \dots + n$

   $x(n) = x(0) + 1 + 2 + \dots + n \rightarrow \frac{n(n+1)}{2}$

$\boxed{\Theta(n^2)}$

d) $x(n) = x(\frac{n}{2}) + n$ for $n > 1$, $x(1) = 1$

0. $x(2^k) = x(2^{k-1}) + 2^k$

1. $x(2^{k-1}) = x(2^{k-2}) + 2^{k-1}$

   $x(2^k) = [x(2^{k-2}) + 2^{k-1}] + 2^k$

2. $x(2^{k-2}) = x(2^{k-3}) + 2^{k-2}$

   $x(2^k) = [x(2^{k-3}) + 2^{k-2}] + 2^{k-1} + 2^k$

3. $x(2^k) = x(2^{k-i}) + 2^{k-(i-1)} + 2^{k-(i-2)} + \dots + 2^k$

4. $2^{k-i} = 1 \rightarrow k - i = 0 \rightarrow i = k$

5. $x(2^k) = x(2^{k-k}) + 2^{k-k+1} + 2^{k-k+2} + \dots + 2^k$

   $= 1 + 2^1 + 2^2 + 2^3 + \dots + 2^k$

   $= 2^{k+1} - 1 = 2 \cdot 2^k - 1 = 2n - 1$

$\boxed{\Theta(n)}$

Scanned with CamScanner

e) $x(n) = x(\frac{n}{3}) + 1$ for $n>1$, $x(1) = 1$

0. $x(3^k) = x(3^{k-1}) + 1$
1. $x(3^{k-1}) = x(3^{k-2}) + 1$

   $x(3^k) = [x(3^{k-2})+1] + 1 = x(3^{k-2}) + 2$
2. $x(3^{k-2}) = x(3^{k-3}) + 1$

   $x(3^k) = [x(3^{k-3}) + 1] + 2 = x(3^{k-3}) + 3$
3. $x(3^k) = x(3^{k-i}) + i$ 4. $3^{k-i} = 1 \rightarrow k-i=0 \rightarrow k=i$
5. $x(3^k) = x(3^{k-k}) + k = 1 + k$ $k = \log_3 n$

$\hookrightarrow 1 + \log_3 n$

$\boxed{\Theta(\log n)}$

3a) Basic operation → multiplication

$M(n) = M(n-1) + 2$ for $n>1$, $M(1) = 0$

$M(n) = M(n-1) + 2$
1. $M(n-1) = M(n-2) + 2$

   $M(n) = [M(n-2) + 2] + 2 = M(n-2) + 4$
2. $M(n-2) = M(n-3) + 2$

   $M(n) = [M(n-3) + 2] + 4 = M(n-3) + 6$
3. $M(n) = M(n-i) + 2i$ 4. $n-i=1 \rightarrow i = n-1$
5. $M(n) = M(n-(n-1)) + 2(n-1) = M(1) + 2(n-1) = 2(n-1)$

$\boxed{\Theta(n)}$

b) The non-recursive algorithm for this function would use a for-loop & would look like:

```
for (i=1; i≤n ;i++) :
    s = s+ (i * i * i)
return s
```

This algorithm would also do $2(n-1)$ multiplications (twice for for every time it runs through the for-loop.

The non-recursive & recursive algorithms are the same.