Fall 2014 CS372 Assignment 2 solutions

1)

a)

Given: $T(n) = 3\mathrm{T}(n/3) + O(1)$. By the definition of Big-O notation we know that for some c and *n0*, $T(n) \le 3\mathrm{T}(n/3) + c$ for every $n \ge n0$. Therefore,

$$T(n) \le 3(3\mathrm{T}(n/3^2) + c) + c$$
$$= 3^2 T(n/3^2) + c(3+1)$$
$$\le 3^2(3\mathrm{T}(n/3^3) + c) + c(3+1)$$
$$= 3^3 T(n/3^3) + c(3^2+3+1)$$

following this pattern we can see that the *k*th iteration is given by,

$T(n) \le 3^k T(n/3^k) + c(\sum_{i=0}^{k-1} 3^i)$. This is the general term k^th term.

The last iteration has T(1) in it. Therefore, the value of k in the last iteration is such that $n/3^k = 1$, or $k = \log_3(n)$. Plugging in $k = \log_3(n)$, we get

$$T(n) \le 3^{\log_3 n} T(1) + c\left(\sum_{i=0}^{k-1} 3^i\right) = nT(1) + \frac{c(3^k-1)}{(3-1)} = nT(1) + c(n-1)/2$$

Since we assume $T(1) = O(1)$ (constant), then $T(n) = O(n)$

b)

Given: $T(n) = T(n-3) + c$. Therefore,

$$T(n) = T(n-6) + 2\mathrm{c}$$
$$= T(n-9) + 3\mathrm{c}$$

Following this pattern we can see that on the *k*th iteration,

$T(n) = T(n-3k) + kc$. This is the general term k^th term.

The last iteration has T(0) in it. Thus, the k in the last iteration is k = n/3, giving us

$$T(n) = T(0) + \left(\frac{n}{3}\right)c = 1 + \left(\frac{c}{3}\right)n.$$

Therefore, T(n)=O(n).

2)

- For algorithm A we have that $T_A(n) = 5T_A(n/2) + O(n)$ so, by the Master Theorem we have that $T_A(n) = O(n^{\log_2(5)})$.
- For algorithm B we have $T_B(n) = 2T_B(n-1) + O(1)$ using the iteration method we can see that

$$T_B(n) \le 2(2T_B(n-2) + c) + c$$
$$= 2^2 T_B(n-2) + c(2^1+1)$$
$$= 2^2(2T_B(n-3) + c) + c(2^1+1)$$
$$= 2^3 T_B(n-3) + c(2^2+2^1+1)$$

so we get $T_B(n) \le 2^k T_B(n-k) + c(\sum_{i=0}^{k-1} 2^i)$, and for k = n-1 we will finally have,

$$T_B(n) \le 2^{n-1} T_B(1) + c(\sum_{i=0}^{n-2} 2^i) = 2^{n-1} T_B(1) + c(2^{n-1} - 1)$$
$$T_B(n) = O(2^n)$$

- For algorithm C we have that $T_C(n) = 9T_C(n/3) + O(n^2)$, using the Master Theorem we have that $T_C(n) = O(n^2 \log(n))$.

We will most likely want to pick the one with the best asymptotic running time, C.

3)

a) Given $T(n) = 3\mathrm{T}(n/3) + c$:

	level	number of nodes	cost per node	argument of T
c	0	3^0	c	n
/ | \\				
c c c	1	3	c	n/3
/ |\\ /|\\ / | \\				
c c c c c c c c c	2	3^2	c	n/3^2
.				
.				
c c	i	3^i	c	n/3^i
.				
.				
T(1) . . . . T(1)	k	3^k	T(1)	n/3^k

At the last level argument is 1, that is, $n/3^k = 1$. Therefore, $k = \log_3(n)$.
Total cost:

$$T(n) = 3^k T(1) + \sum_{i=0}^{k-1} c3^i$$

$$= 3^k T(1) + c\sum_{i=0}^{k-1} 3^i = 3^k T(1) + c(3^k - 1)/(3-1)$$

$$= 3^{\log_3 n} T(1) + \frac{c\left(3^{\log_3 n} - 1\right)}{2} = nT(1) + \frac{c(n-1)}{2} = O(n)$$

b) Given $T(n) = T(n-1) + cn$:

	level	argument of T
cn	0	n
|		
c(n-1)	1	n-1
|		
c(n-2)	2	n-2
.		
.		
c(n-i)	i	n-i
.		
.		
T(1)	k	n-k

Since on the last level k the argument of T is equal to 1 (T(1)), therefore, $n - k = 1$ and $k = n - 1$.
Total cost:

$$T(n) = cn + c(n-1) + c(n-2) + \cdots + c(n-k+1) + T(1)$$
$$= T(1) + c(n-k+1) + \cdots + c(n-2) + c(n-1) + cn$$
$$= T(1) + c(2 + 3 + \cdots + (n-1) + n)$$
$$= T(1) + c\frac{(n+2)(n-1)}{2} = O(n^2)$$

4)

a) Given: $T(n) = 5\mathrm{T}(n/6) + O(n)$. Master Theorem applies with a=5, b=6, d=1. Since $\log_6(5) < 1$ the Master Theorem gives us $T(n) = O(n)$.

b) Given: $T(n) = 16\mathrm{T}(n/4) + O(n^2)$. Master Theorem applies with a=16, b=4, d=2. Since $\log_4(16) = 2$ the Master Theorem gives us $T(n) = O(n^2 \log(n))$.

c) Given: $T(n) = \mathrm{nT}(n/2) + O(1)$. Master Theorem does not apply because a=n which is not a constant.

d) Given: $T(n) = 7\mathrm{T}(n/2) + O(n^3)$. Master Theorem applies with a=7, b=2, d=3. Since $\log_2(7) < 3$ the Master Theorem gives us $T(n) = O(n^3)$.

e) Given: $T(n) = 16\mathrm{T}(n/2) + O(n^4)$. Master Theorem applies with a=16, b=2, d=4. Since $\log_2(16) = 4$ the Master Theorem gives us $T(n) = O(n^4 \log(n))$.

5)

Given:

```
function f(n)
  if n > 1:
     for i=0 to n-1:
           for j=0 to n-1:
                 print_line(''still going'')
     f(n/2)
     f(n/2)
```

Let T(n) be the number of times that the function f(n) will print line “still going”. The body of the inner for loop is executed n^2 times, therefore, T(n) is given by the following recurrence:
$T(n) = n^2 + 2\mathrm{T}(n/2)$. We can use the Master Theorem with a=2, b=2, d=2 and conclude that T(n) = $O(n^2)$. The program prints $O(n^2)$ lines.