Review Activity 8 Solutions

More on Algorithmic Analysis

1) Explain how the **`flip_bits`** function is mapped to the recurrence relationship given below.

```
void flip_bits(bool* A, int first, int last) {
   cout << "flip_bits called  with [" << first << ", " << last << "]" << endl;

   int partition_size = (last - first) / 3;

   if (last <= first + 1) {
         A[first] = 1;
         return;
   }

   flip_bits(A, first + 1, first + partition_size);
   flip_bits(A, first + partition_size + 1, first + partition_size * 2);
   flip_bits(A, first + partition_size * 2 + 1, last);
}
```

Let n = last - first. Recurrence Relationship: $T(1) = a,\ T(n) = 3T\left(\frac{n}{3}\right) + b$ for $n > 1$

Explanation:

For base case, the function runs up to the "return;" line, and performs constant number of steps (a). This happens as soon as last – first = 1. Hence, T(1) = a.

For general case, each function call above results in an invocation of flip_bits on a smaller set. To compute the number of elements, use (<u>end – begin + 1</u>) formula.

The first function call operates on:
first + partition_size - first – 1 + 1 = partition_size = n/3 elements.

The second function call operates on:
first + partition_size * 2 - first - partition_size – 1 + 1 =
partition_size = n/3 elements.

The third function call operates on:
last - first - partition_size * 2 – 1 + 1 = partition_size = n/3 elements.

Hence, in general case, it takes a constant number of steps to start executing the recursive calls, there are three recursive calls made, and each of them operates on n/3 elements, so the run-time performance can be defined as $\mathbf{T(n) = 3T\left(\frac{n}{3}\right) + b}$

```
int main() {
    bool A[100] = {0};
    flip_bits(A, 25, 28);
    cout << A[25] << A[26] << A[27] << A[28] << endl;
    return 0;
}
```

Draw the call tree for **`flip_bits(A, 25, 28);`**

Call Tree:

`flip_bits called  with [25, 28]`

`flip_bits called  with [26, 26]` **`flip_bits called  with [27, 27]`** **`flip_bits called  with [28, 28]`**

`Additional Output: 0111`

2) Solve the recurrence relationship given above by unrolling the recurrence (use back substitution), and give T(n)'s order of growth in terms of the Big-O notation as a function of n. Show all steps in deriving your solution. Note that $k^{\log_k n} = n$.

$$T(1) = a, \quad T(n) = 3T\left(\frac{n}{3}\right) + b \text{ for } n > 1$$

Solution:

$$\boldsymbol{T(n) = 3T\left(\frac{n}{3}\right) + b = 3\,3\,T\left(\frac{n}{3^2}\right) + 3b + b = 3\,3\,3\,T\left(\frac{n}{3^3}\right) + 3^2b + 3b + b = \cdots}$$

$$\boldsymbol{= 3^iT\left(\frac{n}{3^i}\right) + \sum_{j=0}^{i-1} 3^j b = 3^iT\left(\frac{n}{3^i}\right) + \frac{3^i-1}{2}b}$$

When $\frac{n}{3^i} = 1$, let i = c. It follows that $\frac{n}{3^c} = 1$ and $n = 3^c$, so $c = \log_3 n$

From there,

$$\boldsymbol{T(n) = 3^iT\left(\frac{n}{3^i}\right) + \frac{3^i-1}{2}b = 3^{\log_3 n}T(1) + \frac{3^{\log_3 n}-1}{2}b = n\,a + \frac{1}{2}n\,b - \frac{1}{2}b = O(n)}$$