

**Computer Science I – Spring 2012**  
**Lab: Algorithm Analysis (Solutions)**

1) For an  $O(n^3)$  algorithm, one data set with  $n = 3$  takes 54 seconds. How long will it take for a data set with  $n = 5$ ?

Solution

Let  $T(n)$  be the function for the run time of the algorithm. Then,  $T(n) = cn^3$  for some constant  $c$ .

$$\begin{aligned} T(3) &= c3^3 = 54 \\ 27c &= 54, \text{ so } c = 2 \end{aligned}$$

$$T(5) = c5^3 = 2(125) = 250 \text{ seconds.}$$

2) For an  $O(2^n)$  algorithm, a friend tells you that it took 17 seconds to run on her data set on a  $O(2^n)$  algorithm. You run the same program, on the same machine, and your data set with  $n = 7$  takes 68 seconds. What size was her data set?

Solution

Let  $T(n)$  be the function for the run time of the algorithm. Then,  $T(n) = c2^n$  for some constant  $c$ .

$$\begin{aligned} T(7) &= c2^7 = 68 \\ 128c &= 68, \text{ so } c = 68/128 = 17/32. \end{aligned}$$

$$T(n) = c2^n = 17(2^n)/32 = 17, \text{ so } 2^n = 32 \text{ and } n = 5.$$

3) For an  $O(N^k)$  algorithm, where  $k$  is a positive integer, an instance of size  $M$  takes 32 seconds to run. Suppose you run an instance of size  $2M$  and find that it takes 512 seconds to run. What is the value of  $k$ ?

Solution

Let  $T(n)$  be the function for the run time of the algorithm. Then,  $T(n) = cn^k$  for some constant  $c$ .

$$T(m) = cm^k = 32$$

$$\begin{aligned} T(2m) &= c(2m)^k = c2^k m^k = 512, \text{ but since } cm^k = 32, \text{ substituting, we have:} \\ 32(2^k) &= 512 \\ 2^k &= 16 \\ k &= 4 \end{aligned}$$

4) Assume that an  $O(\log_2 N)$  algorithm runs for 10 milliseconds when the input size ( $N$ ) is 32. What is input size makes the algorithm run for 14 milliseconds?

Solution

Let  $T(n)$  be the function for the run time of the algorithm. Then,  $T(n) = c \log_2 n$  for some constant  $c$ .

$$T(32) = c \log_2 32 = 10$$
$$5c = 10, \text{ so } c = 2$$

$$T(n) = 2 \log_2 n = 14, \text{ so } \log_2 n = 7 \text{ and } n = 2^7 = 128.$$

5)

```
int function5(int A[], int B[], int n) {  
  
    int i, j, sum = 0;  
    for (i=0; i<n; i++)  
        for (j=0; j<n; j++)  
            if (A[i] == B[j])  
                sum++;  
    return sum;  
}
```

Solution

The if statement gets executed  $n^2$  times. Namely, it get executed for each ordered pair  $(i,j)$ , ranging from  $(0,0)$  to  $(n-1,n-1)$ . This is the portion that dominates the code, so this function runs in  $O(n^2)$  time.

6)

```
int function6(int A[], int B[], int n) {  
    int i=0, j=0;  
  
    while (i < n) {  
        while (j < n && A[i] > B[j]) j++;  
        i++;  
    }  
    return j;  
}
```

Solution

This one's tricky! On initial observation, we might think that we have two nested loops that could each run  $n$  times. But, on closer inspection, we see that  $j$  can only be incremented  $n$  times and  $i$  can only be incremented  $n$  times and each loop iteration must have one or the other increment. Thus, at most,  $2n$  steps can run before both loops exit. Thus, the run time is  $O(n)$ .

7)

```
int function4(int A[], int B[], int n) {
    int i=0,j;

    while (i < n) {
        j=0;
        while (j < n && A[i] > B[j]) j++;
        i++;
    }
    return j;
}
```

#### Solution

The key difference here is that j is reset to 0 each time, so that inner loop really can run n times every single time. This changes our run-time to  $O(n^2)$ , since that j++ statement can run  $n \times n = n^2$  times.

8)

```
void function8(int n) {

    while (n > 0) {
        printf("%d\n", n);
        n = n/2;
    }
}
```

#### Solution

Each loop iteration, n is divided by 2 and we stop the step after  $n = 1$  (because of integer division). After k loop iterations, the value of n is  $\text{oldn}/2^k$ , where oldn represents the original value of n. Thus, we must have  $n/2^k = 1$ . Our goal is to find k, the number of iterations this code runs. Multiplying, we get  $2^k = n$ . solving, by definition of log, we have  $k = \log_2 n$ . Thus, the run-time is  $O(\lg n)$ .

9)

```
int function8(int n) {  
    int i,j;  
    for (i=0; i<n; i++)  
        for (j=0; j<n; j++)  
            if (j == 1)  
                break;  
    return j;  
}
```

Solution

The j loop never runs more than twice for each value of I, because it gets stopped at  $j = 1$ . Thus, the total number of times it runs is at most  $2 \times n$ . Thus, the function runs in  $O(n)$  time.