Algorithms – Assignment 1 (Solution)

(Complexity)

Prof. Eunwoo Kim

1) Show directly that $f(n) = n^2 + 3n^3 \in O(n^3)$ and $f(n) = n^2 + 3n^3 \in \Omega(n^3)$.

Solution:

We show that $n^2 + 3n^3 \in O(n^3)$ because for $n \geq 1$,

$$n^2 + 3n^3 \leq 4n^3,$$

we can take $C = 4, k = 1$ to obtain our result.

We show that $n^2 + 3n^3 \in \Omega(n^3)$ because for $n \geq 1$,

$$n^2 + 3n^3 \geq 3n^3,$$

we can take $C = 3, k = 1$ to obtain our result.

Thus, since $n^2 + 3n^3 \in O(n^3)$ and $n^2 + 3n^3 \in \Omega(n^3)$, $n^2 + 3n^3 \in \Theta(n^3)$.

2) Using the definitions of O and Ω, show that

$$6n^2 + 20n \in O(n^3), \text{ but } 6n^2 + 20n \notin \Omega(n^3).$$

Solution:

Starting at $k = 9$, $6n^2 + 20n < n^3$, so we can take $C = 1, k = 9$ in the definition of O.

On the other hand, no matter how large C were chosen, the limit $C(6n^2 + 20n)/n^3$ is zero, so $C(6n^2 + 20n)$ cannot stay $> n^3$ which contradicts the definition of Ω.

3) The function $f(n) = 3n^2 + 10n \log n + 1000n + 4 \log n + 9999$ belongs in which of the following complexity categories:

(a) $\Theta(\lg n)$ (b) $\Theta(n^2 \log n)$ (c) $\Theta(n)$ (d) $\Theta(n \lg n)$ (e) $\Theta(n^2)$ (f) None of these

Solution:

(e), $f(n) \in \Theta(n^2)$, by "throwing out" all the lower-order terms.

4) The function $f(n) = (\log n)^2 + 2n + 4n + \log n + 50$ belongs in which of the following complexity categories:

(a) $\Theta(\lg n)$ (b) $\Theta((\log n)^2)$ (c) $\Theta(n)$ (d) $\Theta(n \lg n)$ (e) $\Theta(n(\lg n)^2)$ (f) None of these

Solution:

(c), $f(n) \in \Theta(n)$, "throwing out" all the lower-order terms.

Note: n is only less than $(\log n)^2$ for small values of n.

5) The function $f(n) = n + n^2 + 2^n + n^4$ belongs in which of the following complexity categories:

(a) $\Theta(n)$ (b) $\Theta(n^2)$ (c) $\Theta(n^3)$ (d) $\Theta(n \lg n)$ (e) $\Theta(n^4)$ (f) None of these

Solution:

(f), None of these; it is actually $\Theta(2^n)$.

6) What is the runtime (time complexity) of the below code?

```
def printUnorderedPairs(array):
    for i in range(0,len(array)):
        for j in range(i+1,len(array)):
            print(array[i] + "," + array[j])
```

Solution: $O(n^2)$

Counting the iterations: $(n-1) + (n-2) + (n-3) + \cdots + 2 + 1 = (n^2 - n)/2$.

7) What is the runtime of the below code?

```
def printUnorderedPairs(arrayA, arrayB):
    for i in range(len(arrayA)):
        for j in range(len(arrayB)):
            for k in range(0,100000):
                print(str(arrayA[i]) + "," + str(arrayB[j]))
```

Solution: $O(mn)$ where the size of arrayA is n and the size of arrayB is m.
100,000 units of work is still constant.

8) What is the runtime of the below code?

```
def powersOf2(n):
    # print("n:"+str(n))
    if n < 1:
        return 0
    elif n == 1:
        print(1)
        return 1
    else:
        prev = powersOf2(int(n/2))
        # print("prev:"+str(prev))
        print(prev)
        curr = prev*2
        print(curr)
        return curr
```

Solution: $O(\log n)$
The recursive function powersOf2 does not cover all the positive integer numbers to n and the input number is reduced by 2.