# CS2383 Fall 2021 Assignment 2 (110 marks)
Due Tuesday., Oct. 12, by 5pm.

- Assignments in MS Word format should be handed in via D2L.

1. **(15 marks)** Prove or disprove each of the following statements:

   (a) $n^3 + 8n^2 + 6n + 2$ is $\Theta(n^3)$.

   **Proof: (5 marks)**

$$\begin{aligned} n^3 + 8n^2 + 6n + 2 &\leq n^3 + 8n^3 + 6n^3 + 2n^3 \\ &= 17n^3 \end{aligned} \tag{1}$$

   Let $C = 17$ and $n_0 = 1$. We have $n^3 + 8n^2 + 6n + 2 \leq Cn^3$ for all $n \geq n_0$. Therefore, $n^3 + 8n^2 + 6n + 2$ is $O(n^3)$.

$$n^3 + 8n^2 + 6n + 2 \geq n^3 \tag{2}$$

   Let $C = 1$ and $n_0 = 1$. We have $n^3 + 8n^2 + 6n + 2 \geq Cn^3$ for all $n \geq n_0$. Therefore, $n^3 + 8n^2 + 6n + 2$ is $\Omega(n^3)$.

   So $n^3 + 8n^2 + 6n + 2$ is $\Theta(n^3)$.

   (b) $8n^2 - 6n + 2$ is $\Theta(n^2)$.

   **Proof: (5 marks)**

$$\begin{aligned} 8n^2 - 6n + 2 &\leq 8n^2 + 2 \\ &\leq 8n^2 + 2n^2 \\ &\leq 10n^2 \end{aligned} \tag{3}$$

   Let $C = 10$ and $n_0 = 1$. We have $8n^2 - 6n + 2 \leq Cn^2$ for all $n \geq n_0$. Therefore, $8n^2 - 6n + 2$ is $O(n^2)$.

$$\begin{aligned} 8n^2 - 6n + 2 &\geq 8n^2 - 6n = 2n^2 + (6n^2 - 6n) \\ &\geq 2n^2 \end{aligned} \tag{4}$$

Let $C = 2$ and $n_0 = 1$. We have $8n^2 - 6n + 2 \geq Cn^2$ for all $n \geq n_0$. Therefore, $8n^2 - 6n + 2$ is $\Omega(n^2)$.

So $8n^2 - 6n + 2$ is $\Theta(n^2)$.

(c) $2n^2 - 3n + 50$ is $\Theta(n^2)$.

**Proof: (5 marks)**

$$
\begin{aligned}
2n^2 - 3n + 50 &\leq 2n^2 + 50 \\
&\leq 2n^2 + 50n^2 \\
&\leq 52n^2
\end{aligned}
\tag{5}
$$

Let $C = 52$ and $n_0 = 1$. We have $2n^2 - 3n + 50 \leq Cn^2$ for all $n \geq n_0$. Therefore, $2n^2 - 3n + 50$ is $O(n^2)$.

$$
\begin{aligned}
2n^2 - 3n + 50 &\geq 2n^2 - 3n = n^2 + (n^2 - 3n) \\
&\geq n^2 \text{ when } n \geq 3
\end{aligned}
\tag{6}
$$

Let $C = 1$ and $n_0 = 3$. We have $2n^2 - 3n + 50 \geq Cn^2$ for all $n \geq n_0$. Therefore, $2n^2 - 3n + 50$ is $\Omega(n^2)$.

So $2n^2 - 3n + 50$ is $\Theta(n^2)$.

2. **(total 30 marks, 5 marks per question)** Analyze the running time of the following algorithms asymptotically.

(a) **Algorithm** *for-loop1*($n$):

```
p ← 1
for i ← 1 to n^2 do
    p ← p × i
return p
```

It is $\Theta(n^2)$.

(b) **Algorithm** *for-loop2*($n$):

```
s ← 0
for i ← 1 to n do
    for j ← i to n do
        s ← s + i
return s
```

It is $\Theta(n^2)$.

(c) **Algorithm** Algorithm *WhileLoop1*($n$):

```
x ← 0;
j ← 1;
while (j^3 <= n){
    x ← x + 1;
    j ← j + 1;
}
```

It is $\Theta(n^{\frac{1}{3}})$.

(d) **Algorithm** *WhileLoop2*($n$):

```
x ← 0;
j ← n;
while (j >= 1){
    x ← x + 1;
    j ← 2j/3;
}
```

It is $\Theta(\log n)$.

(e) **Algorithm** *WhileLoop3*($n$):

```
x ← 0;
j ← 2;
while (j <= n){
    x ← x + 1;
    j ← j^3;
}
```

It is $\Theta(\log \log n)$.

(f) **Algorithm** *WhileLoop4*($n$):

```
x ← 0
j ← n
while (j ≥ 1)
    for i ← 1 to j do
        x ← x + 1
    j ← j − 2
return x
```

It is $\Theta(n^2)$.

3. **(total 15 marks, 5 marks per question)** What does each of the following recursive algorithms do? Analyze their running time asymptotically using recursion trees.

(a) **Algorithm** fun1($n$, $m$)

```
if (n = 0)
    return m;
else
    return fun1(n - 1, n + m);
```

The function fun1($n$, $m$) calculates and returns ((1 + 2 ... + $n$-1 + $n$) + $m$). Time complexity: $\Theta(n)$.

(b) **Algorithm** fun2($n$)

```
if (n = 1)
    return 0;
else
    return 1 + fun2(n/2);
```

The function fun2($n$) calculates and returns $\log_2 n$. Time complexity: $\Theta(\log n)$.

(c) **Algorithm** fun3($A$, $l$, $h$)

**Input:** $A$ is an array, $l$ and $h$ are two integers.

```
if (l ≥ h)
    return;
```

```
minindex ← l
minivalue ← A[l]
for ( i ← l + 1; i ≤ h; i++)
        if (minivalue > A[i])
                minivalue ← A[i];
                minindex ← i;
swap(A[l], A[minindex]);
fun3(A, l + 1, h);
```

The function fun3($A$, $l$, $h$) sorts a sub-array from $A[l]$ to $A[h]$ using selection sort. The initial call is: fun3($A$, 0, $n-1$) with time complexity: $\Theta(n^2)$.

4. **(10 marks)** Given a stack that includes $n$ numbers, write a recursive algorithm to sort the elements in the stack. For example, if the contents of the input stack is: 3 (top), 5, 2, 1, 4, the sorted stack should be 1 (top), 2, 3, 4, 5. Assume that the size of the stack is $n$, what is the time complexity of your algorithm.

**Algorithm** stackSort($S$)
*Input*: a stack $S$
*Output*: $S$ with the elements sorted.

```
if (!S.isEmpty())
   temp ← S.pop();
   stackSort(S);
   stackInsert(S, temp);
```

**Algorithm** stackInsert($S$, $e$)
*Input*: a sorted stack $S$ and an element $e$
*Output*: $S$ with $e$ inserted, and $S$ is sorted.

```
if (S.isEmpty() OR e > S.top())
   S.push(e);
else
   temp ← S.pop();
   stackInsert(S, e);
   S.push(temp);
```

Time complexity: $\Theta(n^2)$.

5. **(20 marks)** Write a recursive algorithm that reverses a given integer. For example, if the given number is 12345, the output of your algorithm should be 54321. Analyze its time complexity using a recursion tree. Then describe an algorithm for determining a given number $w$ is palindrome or not. A number is called palindrome if it is equal to its reverse. For example, 1221 is palindrome. Implement your algorithm in Java and hand in the source code via D2L.

**Algorithm** *reverse*($n$, $rev$):
    **Input:** an integer $n$.
    **Output:** reversed $n$.

```
if (n=0) return rev;
return reverse(n/10, rev × 10 + n % 10);
```

Draw a recursion tree. The time complexity is $\Theta(\log n)$ (or the number of digits of $n$).

Algorithm: 10 marks; Time complexity: 3 marks; Java implementation: 7 marks.

6. **(20 marks)** Write a recursive Insertion Sort algorithm that takes an array $A$ of $n$ numbers as input. Analyze its time complexity using a recursion tree. Implement your algorithm in Java and hand in the source code via D2L.

**Algorithm** *insertionSort*($A$, $n$):
    **Input:** Array $A$ of $n$ real numbers.
    **Output:** Sorted $A$.

```
if (n=1) return;
insertionSort(A, n − 1);
temp ← A[n − 1];
for (i ← (n − 2) to 0)
    if (A[i] > temp)
        A[i + 1] ← A[i];
A[i + 1] ← temp;
```

Draw a recursion tree. The time complexity is $\Theta(n^2)$.

Algorithm: 10 marks; Time complexity: 3 marks; Java implementation: 7 marks.