CS3230
Tutorial 2

1. Which of the following recurrence relations can be solved using the master theorem? If it cannot be solved, state so. If it can be solved, give the answer (in terms of Θ notation) and state which clause of the master theorem applies.

   (a) $T(n) = 15T(n/3) + n^3$.

   Ans: Yes. $a = 15$, $b = 3$, $k = 3$, $b^k = 3^3 > a$. Thus, $T(n) = \Theta(n^3)$.

   (b) $T(n) = 3T(n/2) + n$.

   Ans: $a = 3, b = 2, k = 1$. Thus, $a > b^k$.

   Thus, $T(n) = \Theta(n^{\log_2 3})$.

   (c) $T(n) = 4T(n/2) + n^2$.

   Ans: $a = 4, b = 2, k = 2$. Thus, $a = b^k$.

   Thus, $T(n) = \Theta(n^2 \log n)$.

   (d) $T(n) = T(n/3) + T(2n/3) + n$.

   Ans: No.

   (e) $T(n) = 4T(n/3) + n \log n$.

   Ans: Yes, by taking $a = 4, b = 3, k = 1 + \epsilon$, where $1 < 1 + \epsilon < log_3 4$, we see that $n^1 \leq f(n) \leq n^{1+\epsilon}$, for large enough n.

   Thus, $T(n) = O(n^{\log_3 4})$.

   Similarly, as $f(n) = \Omega(n)$, we have that $T(n) = \Omega(n^{\log_3 4})$.

   Giving us $T(n) = \Theta(n^{\log_3 4})$.

2. Use induction to detmine the upper bound for the following recurrence relations. Express your answer in big O-notation.

   (a) $T(n) = 2 + \sum_{i=1}^{n-1} T(i)$; $T(1) = 1$.

   Ans: We show by induction that $T(n) \leq 2 * 2^n$, for $n \geq 1$.

   Base case: For $n = 1$, the relation clearly holds.

   Suppose the relation holds for $1 \leq n \leq k$. We show that the relation holds for $n = k + 1$.

$$T(k+1) \quad = \quad 2 + \sum_{i=1}^{k} T(i)$$

$$\begin{aligned} &\leq 2 + 2 * [2 + 2^2 + 2^3 + \ldots + 2^k] \\ &\leq 2 * 2^{k+1} \end{aligned}$$

Therefore, $T(n) = O(2^n)$.

(b) $T(n) = T(n-1) + 5n$; $T(1) = 5$.

Ans: We show that $T(n) \leq 5 * n^2$, for $n \geq 1$.

This clearly holds for $n = 1$.

Suppose the relation holds for $1 \leq n \leq k$. We show that the relation holds for $n = k + 1$.

$$\begin{aligned} T(k+1) &= T(k) + 5(k+1) \\ &\leq 5k^2 + 5(k+1) \\ &\leq 5(k+1)^2 \end{aligned}$$

Thus $T(n) = O(n^2)$.

(c) $T(n) = T(n-1) + 3n^2$; $T(1) = 1$.

Ans: We show that $T(n) \leq 100n^3$, for $n \geq 1$.

This clearly holds for $n = 1$.

Suppose the relation holds for $1 \leq n \leq k$. We show that the relation holds for $n = k + 1$.

$$\begin{aligned} T(k+1) &= T(k) + 3(k+1)^2 \\ &\leq 100k^3 + 3(k+1)^2 \\ &\leq 100(k+1)^3 \end{aligned}$$

Thus, $T(n) = O(n^3)$.

(d) $T(n) = T(n-1) + \log n$; $T(1) = 1$.

Ans: We show that $T(n) \leq 1 + n \log n$, for $n \geq 1$

This clearly holds for $n = 1$.

Suppose the relation holds for $1 \leq n \leq k$. We show that the relation holds for $n = k + 1$.

$$\begin{aligned} T(k+1) &= T(k) + \log(k+1) \\ &\leq 1 + k \log k + \log(k+1) \\ &\leq 1 + (k+1)\log(k+1) \end{aligned}$$

Thus, $T(n) = O(n \log n)$.

(e) $T(n) = 2T(n-1) + 1$; $T(1) = 1$.

Ans: We show that $T(n) = 2^n - 1$, for $n \geq 1$.

This clearly holds for $n = 1$.

Suppose the relation holds for $1 \leq n \leq k$. We show that the relation holds for $n = k + 1$.

$$\begin{aligned} T(k+1) &= 2T(k) + 1 \\ &= 2(2^k - 1) + 1 \\ &= 2^{k+1} - 1 \end{aligned}$$

Thus $T(n) = O(2^n)$.

3. Recall Euclid's algorithm for GCD. Assume $m \geq n$.

   Then,
$$GCD(n, m) = \begin{cases} m, & \text{if } n = 0; \\ GCD(m \ \mathbf{mod}\ n, n), & \text{otherwise;} \end{cases}$$

   Give a good upper bound on the running time of the above algorithm in terms of m, n.

   Ans: Note that, if $n \leq m/2$, then $m \ \mathbf{mod}\ n < n \leq m/2$. On the other hand, if $n > m/2$ then again $m \ \mathbf{mod}\ n \leq m - n < m/2$.

   Thus, in both cases, after two iterations, the maximum of the two values (on which GCD is taken), reduces by at least half.

   Hence, one can use the recurrence

   $T(m) \leq T(\lfloor m/2 \rfloor) + 2$, giving us $T(m) \leq O(\log m)$.

4. Binary Insertion Sort:

   Binary insertion sort is similar to insertion sort, except that: instead of finding the place to insert a new element using linear search, it does a binary search.

   Please give an algorithm to do binary insertion sort.

   Do a time-complexity analysis of your algorithm.

   Ans:

```
Insertion Sort
   For $i = 2$ to $n$ {
        Let $temp = A[i]$
        $j = Bsearch(A[i], A, 1, i)$.
        For $k = i - 1$ downto $j$, Do {
              $A[k+1] = A[k]$                                    (S*)
        }
        $A[j] = temp$.
   }
End
```

```
$Bsearch(key, A, s, t)$
(* Returns least $k$ such that $s \leq k \leq t$ with $A[k] \geq key$. *)
(* Assumption: there is a number $k$ such that $s \leq k \leq t$, and $A[s] \geq key$. *)
    If $s = t$, then return $s$
    Let $w = \lfloor \frac{s+t}{2} \rfloor$.
    If $key > A[w]$, then $Bsearch(key, A, w+1, t)$.
    Else, $Bsearch(key, A, s, w)$.
End
```

Complexity:

Number of comparisons in the worst case:

$C(n) \leq n*$ complexity of binary search for upto n elements.
Thus, $C(n) = O(n * \log n)$.

Number of assignments in the worst case at step (S*): at most n for each value of i.

$$A(n) \leq n * n = O(n^2).$$

Therefore, the complexity of the algorithm is $O(n^2)$.

Note that there are inputs (inputs in reverse sorted order) on which the number of assignments in step (S*) is at least $1 + 2 + 3 + \ldots + n - 2$. Thus, the algorithm takes at least $\Omega(n^2)$ time.

Using above, the worst case complexity of the algorithm is $\Theta(n^2)$.

5. Prof Larry suggests to change the quicksort algorithm to:

```
Quicksort$(A, i, j)$
    If $i < j$, then
        $p =$Partition$(A, i, j)$
        Quicksort$(A, i, p)$
        Quicksort$(A, p, j)$
End
```

Will the above algorithm work? If so, what is the complexity of the algorithm. If not, why not?

Ans: No, as if $p = i$, then the algorithm does not make progress.