COMP 3011 Assignment 1

JAHJA Darwin, 16094501d

---

1. a)

Using master method with $a = 27$, $b = 3$, and $f(n) = n^2$, $n^{log_b a} = n^{log_3 27} = n^3$.

Thus, $f(n) = n^2 = O(n^{log_b a - \epsilon})$ with $\epsilon = 1$.

By case 1 of the master method, $T(n) = \Theta(n^{log_b a}) = \Theta(n^3)$

1. b)

The substitution method can be used to show $T(n) = \Theta(n)$:

- ***Upper bound***

  Guess: $T(n) = O(n)$

  Assume that $T(n) \leq cn$ for some constant $c > 0$ holds for all positive $m < n$. Then:

  $$T(n) = T(2n/3) + T(n^{2/3}) + n \leq 2cn/3 + cn^{2/3} + n$$

  When $c > 3$, $2c/3 + 1 < c$, so for all sufficently large n:

  $$T(n) = (2c/3 + 1)n + cn^{2/3} \leq cn$$

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

- ***Lower bound***

  Guess: $T(n) = O(n)$

  Assume that $T(n) \geq cn$ for some constant $c > 0$ holds for all positive $m < n$. Then:

  $$T(n) = T(2n/3) + T(n^{2/3}) + n \geq 2cn/3 + cn^{2/3} + n$$

  When $c \leq 3$, $2c/3 + 1 \geq c$, so for all sufficently large n:

  $$T(n) = (2c/3 + 1)n + cn^{2/3} \geq cn$$

  Therefore, $T(n) = \Omega(n)$.

As $T(n) = O(n)$ and $T(n) = \Omega(n)$, thus $T(n) = \Theta(n)$

1. c)

The Iteration method can be used to solve T(n). Through repeating substitution,

$$
\begin{aligned}
T(n) &= T(n-1) + lg\, n \\
&= T(n-2) + lg(n-1) + lg\, n \\
&= T(n-2) + lg[n \cdot (n-1)] \\
&= ... \\
&= T(n-k) + lg[n \cdot (n-1) \cdot ... \cdot (n-k)]
\end{aligned}
$$

As the base case is $T(1)$, thus $n - 1 = k \rightarrow k = n + 1$, and substituting this we can get:

$$T(n) = T(1) + lg[n \cdot (n-1) \cdot ... \cdot 1]$$

As $n \cdot (n-1) \cdot ... \cdot 1 = n!$, and by Stirling's approximation, $\Theta(lgn!) = \Theta(n \cdot lgn)$. Therefore:

$$
\begin{aligned}
T(n) &= T(1) + lg\, n! \\
&= T(1) + \Theta(n \cdot lg\, n) \\
&= \Theta(n \cdot lg\, n)
\end{aligned}
$$

---

2. Algorithm 1: Find sum of 2018

Input: An unsorted list L of negative and positive integers

Output: "YES" if there exist three elements $a, b, c \in L$ (with repetitions allowed) such that $a + b + c = 2018$; "NO" otherwise.

```
MergeSort(L)                          // O(nlogn)
n = length of L                       // O(1)
for i = 0 to n do                     // O(n)
  start = i
  end = n
  while end > start do                // O(n^2)
    sum = L[i] + L[start] + L[end]
    if sum = 2018 return "YES"
    if sum > 2018 then
      end = end - 1
    else
      start = start + 1
return "NO"                           // O(1)
```

To analyze the running time, Merge Sort takes $O(nlogn)$ time. Operations inside *for* loop takes $O(n)$ while those inside the nested *while* loop takes $O(n^2)$. Other operations outside the loop take $O(1)$ time.

Therefore, by summing up the cost of every line, the time complexity of this algorithm is $O(n^2)$.

3.

The recurrence relation of multiplying two (4x4) matrices using Divide and Conquer Algorithm with M submatrix multiplications can be shown by this equation:

$$T(n) = MT(n/4) + \Theta(n^2)$$

To find the largest value of constant integer M to get asymptotic improvement over Le Gall's algorightm, which runs in $o(n^{2.37287})$, the recurrence relation can be shown as:

$$T(n) = MT(n/4) + o(n^{2.37287})$$

Using master method with $a = M$ and $b = 4$, $f(n) = n^{2.37287}$, $n^{log_b a} = n^{log_4 M}$.

To satisfy case 1 of the master method, $log_4 M < 2.37287$ such that $f(n) = n^{2.37287} = O(n^{log_4 M - \epsilon})$ with $\epsilon > 0$.

$$\begin{aligned} log_4 M &< 2.37287 \\ M &< 26.83 \\ M &= 26 (\text{to nearest integer}) \end{aligned}$$

Therefore, the largest value of M is 26.

4.

Using greedy algorithm, this problem can be solved with the following steps:

1. Set L' as an empty array, [ ].
2. Apply Breath-First Search to travel over all nodes in the tree. Then, for each leaf $x \notin L'$, find the unique length of the path from root to leaf x. *i.e.* $ulen(x) = |E(L' + x)| - |E(L')|$.
3. For all leaves that is not in L', *i.e.* $e \notin L'$, find leaf x' which have maximum $ulen(x')$.
4. Append x' to L'.

5. If $|L'| = l$, return L' as the result. Otherwise, go back to *step 2* and continue the process.

Prove the algorithm runs in polynomial time

The number of iterations in the Breath-First Travelsal is $O(n)$, where n is the size of the tree. And we need to perform Breath-First Travelsal for l times to acquire L' with the largest $|E(L')|$ for $|L'| = l$. Therefore, $O(ln)$ is the upper bound on this algorithm's time complexity, which is polynomial in the input size.

Prove the correctness of the algorithm

Assume that we have got a leaf set L' based on the above greedy algorithm, and A is one of the possible results such that $|E(A)|$ is the largest. If A can be changed to L' by adding and deleting one leaf correspondingly while remaining $|E(A)|$ unchanged, then we can prove that the algorithm is correct as $|E(A)| = |E(L')|$.

To show this process, let's define $u(x)$ as the number of unique edges contributed by leaf x to $E(A)$, $i.e. u(x) = |E(A)| - |E(A - x)|$. Thus, if $L' \subseteq A, ulen(x) \geq u(x)$ holds for any x where $x \notin L' \text{and} x \in A$.

Starting from $L' = []$, each time when we found a new leaf x which has the largest value of $ulen(x)$, if $x \in A$, then we append x to L'. Otherwise, we need to find a leaf x' in A that has the lowest common ancestor(LCA) with x.

Then, let d and d' be the distance between x and LCA, and the distance between x' and LCA respectively. If $x' \notin L'$, it can be proven that $d' \leq d$ as $ulen(x') < ulen(x)$, thus $|E(A - x' + x)| = |E(A)| - x' + x \geq |E(A)|$. And x' in A can be replaced by x while maintaining $|E(A)|$ unchanged. Otherwise, we need to find an arbitrary leaf a such that $a \in A$ but $a \notin L'$. Similarly, as $u(a) \leq ulen(a) \leq ulen(x) = d, |E(A - a + x)| = |E(a)| - u(a) + ulen(x) \geq |E(A)|$. Thus, we can replace a in A with x and append x into L'.

As $x \in A$ and $L' \subseteq A$ still holds, repeat the process until A is totally changed to L'.