Problem 1. Asymptotic Growth

Consider the functions

$f_1(n) = \log_{10}(2^{\log_2(n)})$, $f_2(n) = \log_2(n!)$, $f_3(n) = 10^{(2n + \log_3(n) + 100)}$, $f_4(n) = (\ln(n))^3$.

Q1 (8 points): Compute the asymptotic complexity of f_1, f_2, f_3, f_4.

Solution:

1. Apply logarithm property, $x^{(\log_b(n))} = n^{(\log_b(x))}$, where b, x and n are positive real numbers and $b \neq 1$. Then $2^{(\log_2(n))} = n^{(\log_2(2))} = n$, then $f_1 = \log_{10}(n)$, and then, same as the reasoning of 1) every function is Θ of itself, hence $f_1(n) = \Theta(\log_{10}(n))$.
2. Using Stirling's approximation $n! \approx \sqrt{2\pi n}(\frac{n}{e})^n$ and the properties of logarithm

$$\begin{aligned}
\log_2(n!) &\approx \log_2(\sqrt{2\pi n}(\frac{n}{e})^n) \\
&\approx \log_2(\sqrt{2\pi n}) + \log_2(\frac{n}{e})^n \\
&\approx \tfrac{1}{2}log_2(2\pi n) + n\log_2(\frac{n}{e}) \\
&\approx \tfrac{1}{2}log_2(2\pi n) + n\log_2(n) - n\log_2(e) \\
&\approx \Theta(\log_2 n) + \Theta(n\log_2 n) - \Theta(n) \\
&\approx \Theta(n\log_2 n). \text{ (Dropping the low-order terms)}
\end{aligned}$$

3. As polynomial function n grows faster than polylogarithmic function $\log_3 n$ and constant 100, hence, by dropping the low-order term, $f_3 = \Theta(10^n)$.
4. Cannot simply f_4 more. Since every function is Θ of itself, hence, $f_4(n) = \Theta(\ln(n))^3$.
5. $f_5(n) = \Theta(n^3)$
6. $f_6(n) = \Theta(n^{10})$

Q2 (4 points): Rank the above functions by decreasing order of growth.

Solution:

$$f_3 > f_6 > f_5 > f_2 > f_4 > f_1$$

Problem 2. Sorting

Given an array of size n, $A[1, \cdots, n]$:

Q1 (6 points): There exist another method to sort called *bubble sort*. The pseudocode of bubble sort is as follows:

```
for (int i = n - 1; i >= 0; i--):
    for (int j = 0; j <= i - 1; j++):
        if (A[j] > A[j + 1]):
            swap(A[j], A j + 1])
```

We first split the array into two equal-size pieces, sort the two pieces using the above bubble sort method, and then merge them into one array. Write down the complexity of this method. (Hint: Find the $\Theta(\cdot)$ of bubble sort first)

Solution:

- Time for diving the array into two pieces: $\Theta(1)$;
- Time for sorting the two pieces using bubble sort method: $2\Theta((n/2)^2) = \Theta(n^2)$;
- Time for merging:$\Theta(n)$.

Therefore, the complexity of this method: $T(n) = \Theta(n^2) + \Theta(n) + \Theta(1) = \Theta(n^2)$

Q2 (6 points): Now try to sort the array A using a *K-Way Merge-Sort* algorithm. Similar to the merge-sort algorithm taught in class, in K-way merge-sort, we split the array into K equal-sized pieces. Then, we recursively sort each piece and merge them into one sorted array. Write and solve the recurrence related to the complexity of this K-way merge-sort. Solution:

The recurrence:

$$T(n) = KT\left(\frac{n}{K}\right) + \Theta(Kn)$$

Solve the recurrence using master theorem case 2, we have: $T(n) = \Theta(nlogn)$ if K is small.

Q3 (2 points): For what values of K will the algorithm in **Q2** be better than the algorithm in **Q1**?

Solution:

The full complexity for K-way merge algorithm (assuming non optimum) : $Kn \log_K n$. As long as $K < n$, the algorithm in **Q2** will be better than **Q1**.

Problem 3.Master Theorem

Q1 (6 points): $T(n) = T(7n/10) + n$

Solution: $T(n) = \Theta(n)$

Q7 (6 points): $T(n) = 2T(n/4) + \sqrt{n}$

Solution:

$T(n) = \sqrt{n} \log n$

Q3 (7 points): $T(n) = T(\sqrt{n}) + 1$. (Hint: Variable change: $m = \log n$.)

Solution:

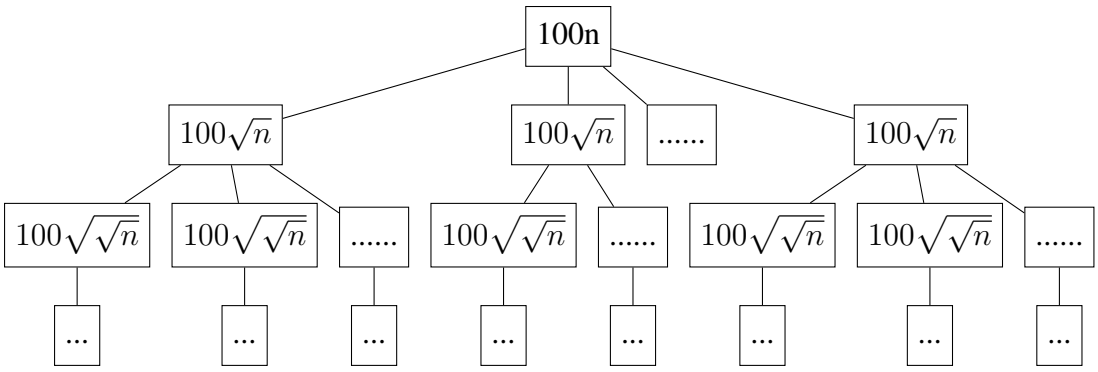
Let $m = \log n$, and$S(m) = T(2^m), T(2^m) = T(2^{m/2}) + 1$, so we have $S(m) = S(m/2) + 1$. Using the master theorem, $n^{\log_b a} = 1$ and $f(n) = 1$, Case 2 applies, and $S(m) = \Theta(\log m)$. Therefore, $T(n) = \Theta(\log \log n)$

Bonus Problem (15 points)

Q1 (7 points): $T(n) = \sqrt{n}T(\sqrt{n}) + 100n$. (Hint: draw recurrence trees).

Solution:

Draw a recursion tree:



Calculating the cost for each layer, we found that he cost of the first layer is 100n, the cost of the second layer is $c = \sqrt{n} \cdot 100\sqrt{n} = 100n$ and the cost of third layer is also $\sqrt{n} \cdot \sqrt{\sqrt{n}} \cdot 100\sqrt{\sqrt{(n)}} = 100n$ and so on.

Consider the recursion equation $T(n) = T(\sqrt{n}) + \Theta(1)$, let $k = \lg n$, then $T(2^k) = T(2^{\frac{k}{2}}) + \Theta(1)$. Let $S(k) = T(2^k)$, we have $S(k) = S(k/2) + \Theta(1)$. Based on this equation, we can get that the height of the recursion tree should be $\lg k$, which is also known as $\lg \lg n$. we can even extend this result to equations with the format of $T(n) = f(n) \cdot T(\sqrt{n}) + g(n)$. The

depth of the recursion tree is $h = \Theta(\lg \lg n)$. (You can also draw a recursion tree to find the height)

Thus, we could give the value of T(n) as $T(n) = h \cdot 100n = \Theta(n \cdot \lg \lg n)$.

Another Solution: We divide both of the equation by n and we get $\frac{T(n)}{n} = \frac{T(\sqrt{n})}{\sqrt{n}} + 100$.

Let $S(n) = \frac{T(n)}{n}$, we have $S(n) = S(\sqrt{n}) + 100$. Draw a recursion tree or let $m = \lg n$, we can get that $S(n) = \Theta(\lg \lg n)$. Thus, $T(n) = n \cdot S(n) = \Theta(n \cdot \lg \lg n)$.

Q2 (8 points): $T(n) = T(n/2 + \sqrt{n}) + \sqrt{6046}$. (Hint: find how much the problem size increase or decrease at each stage).

Solution:

By induction, $T(n)$ is a monotonically increasing function. Thus, for large enough n, $T(n/2) \leq T(n/2 + \sqrt{n}) \leq T(3n/4)$. At each stage, we incur constant cost of $\sqrt{6046}$, but we decrease the problem size to at least one half and at most three quarters. Therefore, the complexity is in the order of $\lg n$: $T(n) = \Theta(\lg n)$.