Problem 1. Asymptotic Growth

Consider the functions

$$f_1(n) = \log_{10}(3^{\log_3(n)}),\ f_2(n) = \log_2(n!),\ f_3(n) = 10^{(100n + \log_3(n) + 2)},\ f_4(n) = (\ln(n))^3,$$
$$f_5(n) = 3n^3 + 2n^{\frac{3}{2}} + n + 50,\ f_6(n) = \binom{n}{8}.$$

Q1 (8 points): Compute the asymptotic complexity of $f_1, f_2, f_3, f_4, f_5, f_6$.

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 $3^{(\log_3(n))} = n^{(\log_3(3))} = n$, then $f_1 = \log_{10}(n)$, and then, since every function is Θ of itself, which results in $f_1(n) = \Theta(\log_{10}(n))$.
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. The polynomial function n grows faster than the logarithmic function $\log_3 n$ and constant 2, hence, by dropping the low-order terms, $f_3 = \Theta(10^n)$.
4. We cannot simplify 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. $f_6(n) = \Theta(n^8)$

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. Merge Sort

Harry Potter, like the students of SUTD's 50.004 has been taught merge sort. Professor Snape poses a challenge to Harry to merge k sorted lists of size n/k ($n > k, k > 3$) to form a single sorted list of size n.

Harry uses the following algorithm for merging k sorted lists, each having n/k elements. He takes the first list and merges it with the second list using a linear-time algorithm for merging two sorted lists, such as the merging algorithm used in merge sort. Then, he merges the resulting list of $2n/k$ elements with the third list, merges the list of $3n/k$ elements that results with the fourth list, and so forth, until he ends up with a single sorted list of all elements.

Q1 (8 points): Analyse the worst-case running time of Harry's algorithm in terms of n and k

Solution:

Merging the first two lists, each of n/k elements, takes $2n/k$ time. Merging the resulting $2n/k$ elements with the third list of n/k elements takes $3n/k$ time, and so on. Thus for a total of list, we have:

$$\begin{aligned} Time &= \frac{2n}{k} + \frac{3n}{k} + \cdots + \frac{kn}{k} \\ &= \sum_{i=2}^{k} \frac{in}{k} \\ &= \frac{n}{k} \sum_{i=2}^{k} i \\ &= \frac{n}{k} \frac{(k+2)(k-1)}{2} \\ &= \Theta(nk) \end{aligned}$$

Q2 (6 points): Briefly describe an algorithm for merging k sorted lists, each of length n/k, whose worst-case running time is $O(n \lg k)$. Briefly justify the running time of your algorithm. (There are several possible answers, only one is required)

Solution:

One method is to repeatedly pair up the lists, and merge each pair. This method can also be seen as a tail component of the execution merge sort, where the analysis is clear. Finally, a conceptually simple method is to store a min priority queue of the minimum elements of each of the k lists. At each step, we output the extracted minimum of the priority queue, and using satellite data determine from which of the k lists it came, and insert the next element from that list into the priority queue.

Problem 3. Master Theorem

Q1 (6 points): $T(n) = 2T(n/4) + n^{0.51}$

Solution:

$T(n) = \Theta(n^{0.51})$ (Case 3)

Q2 (6 points): $T(n) = 3T(n/3) + n/2$

Solution:

$T(n) = \Theta(n \log n)$ (Case 2)

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, $T(n) = \Theta(\log \log n)$

Problem 4. Heap

For each of the following questions, answer either True or False. **Explain your choice.**

Q1 (2 points): The depths of any two leaves in a max heap differ by at most 1.

Solution:

True. A heap is derived from an array and new levels to a heap are only added once the leaf level is already full. As a result, a heap's leaves are only found in the bottom two levels of the heap and thus the maximum difference between any two leaves' depths is 1.

Q2 (3 points): A heap H has each key randomly increased or decreased by 1. The random choices are independent. We can restore the heap property on H in $\Theta(n \log n)$ time

Solution:

False. Simply call BUILD-HEAP at the root, which runs in $\Theta(n)$ time.