1 Asymptotic Complexity Comparisons

(a) Order the following functions so that for all \( i, j \), if \( f_i \) comes before \( f_j \) in the order then \( f_i = O(f_j) \). Do not justify your answers.

- \( f_1(n) = 3^n \)
- \( f_2(n) = n^{\frac{1}{3}} \)
- \( f_3(n) = 12 \)
- \( f_4(n) = 2^{\log_2 n} \)
- \( f_5(n) = \sqrt{n} \)
- \( f_6(n) = 2^n \)
- \( f_7(n) = \log_2 n \)
- \( f_8(n) = 2^{\sqrt{n}} \)
- \( f_9(n) = \log_2 n \)

As an answer you may just write the functions as a list, e.g. \( f_8, f_9, f_1, \ldots \)

(b) In each of the following, indicate whether \( f = O(g) \), \( f = \Omega(g) \), or both (in which case \( f = \Theta(g) \)). Briefly justify each of your answers. Recall that in terms of asymptotic growth rate, logarithmic < polynomial < exponential.

<table>
<thead>
<tr>
<th>( f(n) )</th>
<th>( g(n) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \log_3 n )</td>
<td>( \log_4(n) )</td>
</tr>
<tr>
<td>( n \log(n^4) )</td>
<td>( n^2 \log(n^3) )</td>
</tr>
<tr>
<td>( \sqrt{n} )</td>
<td>( (\log n)^3 )</td>
</tr>
<tr>
<td>( n + \log n )</td>
<td>( n + (\log n)^2 )</td>
</tr>
</tbody>
</table>

2 Bit Counter

Consider an \( n \)-bit counter that counts from 0 to \( 2^n - 1 \). As it moves from \( x \) to \( x + 1 \), it tracks how many bits were flipped from \( x \).

When \( n = 5 \), the counter has the following values:

\[
\begin{array}{ccc}
\text{Step} & \text{Value} & \text{# Bit-Flips} \\
0 & 00000 & - \\
1 & 00001 & 1 \\
2 & 00010 & 2 \\
3 & 00011 & 1 \\
4 & 00100 & 3 \\
\vdots & \vdots & \vdots \\
31 & 11111 & 1 \\
\end{array}
\]

For example, the last two bits flip when the counter goes from 1 to 2. Using \( \Theta(\cdot) \) notation, find the growth of the total number of bit flips (the sum of all the numbers in the “# Bit-Flips” column) as a function of \( n \).
3 Asymptotic Bound Practice

Prove that for any $\epsilon > 0$ we have $\log x \in O(x^\epsilon)$.

4 Hadamard matrices

The Hadamard matrices $H_0, H_1, H_2, \ldots$ are defined as follows:

- $H_0$ is the $1 \times 1$ matrix $[1]$
- For $k > 0$, $H_k$ is the $2^k \times 2^k$ matrix

\[
H_k = \begin{bmatrix}
H_{k-1} & H_{k-1} \\
H_{k-1} & -H_{k-1}
\end{bmatrix}
\]

(a) Write down the Hadamard matrices $H_0$, $H_1$, and $H_2$.

(b) Compute the matrix-vector product $H_2 \cdot v$ where $H_2$ is the Hadamard matrix you found above, and

\[
v = \begin{bmatrix}
1 \\
-1 \\
-1 \\
1
\end{bmatrix}
\]

Note that since $H_2$ is a $4 \times 4$ matrix, and the vector has length 4, the result will be a vector of length 4.

(c) Now, we will compute another quantity. Take $v_1$ and $v_2$ to be the top and bottom halves of $v$ respectively. Therefore, we have that

\[
v_1 = \begin{bmatrix}
1 \\
-1
\end{bmatrix}, v_2 = \begin{bmatrix}
-1 \\
1
\end{bmatrix}, v = \begin{bmatrix}
v_1 \\
v_2
\end{bmatrix}
\]

Compute $u_1 = H_1(v_1 + v_2)$ and $u_2 = H_1(v_1 - v_2)$ to get two vectors of length 2. Stack $u_1$ above $u_2$ to get a vector $u$ of length 4. What do you notice about $u$?

(d) Suppose that

\[
v = \begin{bmatrix}
v_1 \\
v_2
\end{bmatrix}
\]

is a column vector of length $n = 2^k$. $v_1$ and $v_2$ are the top and bottom half of the vector, respectively. Therefore, they are each vectors of length $\frac{n}{2} = 2^{k-1}$. Write the matrix-vector product $H_kv$ in terms of $H_{k-1}, v_1$, and $v_2$ (note that $H_{k-1}$ is a matrix of dimension $\frac{n}{2} \times \frac{n}{2}$, or $2^{k-1} \times 2^{k-1}$). Since $H_k$ is a $n \times n$ matrix, and $v$ is a vector of length $n$, the result will be a vector of length $n$.

(e) Use your results from (c) to come up with a divide-and-conquer algorithm to calculate the matrix-vector product $H_kv$, and show that it can be calculated using $O(n \log n)$ operations. Assume that all the numbers involved are small enough that basic arithmetic operations like addition and multiplication take unit time. You do not need to prove correctness.

5 Extra Divide and Conquer Practice: Sorted Array

Given a sorted array $A$ of $n$ (possibly negative) distinct integers, you want to find out whether there is an index $i$ for which $A[i] = i$. Devise a divide-and-conquer algorithm that runs in $O(\log n)$ time.