CP 312, Fall 2025, Assignment 1

1. [4 marks] Using the definition of Θ-notation, prove that

$$(n^2 + 1)(n + 1)(\log(256n^4)) \in \Theta(n^3 \log n).$$

You can not use limit computation to establish this result!
Solution:

Step 1: Simplifying the expression

- $n^2 + 1 \leq 2n^2$ and $n^2 + 1 \geq n^2$ for $n \geq 1$
- $n + 1 \leq 2n$ and $n + 1 \geq n$ for $n \geq 1$
- $\log(256n^4) = \log(256) + \log(n^4) = \log(256) + 4\log n$

For large n, $\log(256)$ is a constant, so $\log(256n^4) \leq 5\log n$ and $\log(256n^4) \geq 4\log n$.

Step 2: Finding upper and lower bounds

Let $f(n) = (n^2 + 1)(n + 1)\log(256n^4)$.

- **Upper bound:**

$$f(n) \leq 2n^2 \cdot 2n \cdot 5\log n = 20n^3 \log n$$

- **Lower bound:**

$$f(n) \geq n^2 \cdot n \cdot 4\log n = 4n^3 \log n$$

Step 3: Apply the definition of Θ-notation

By definition, $f(n) \in \Theta(n^3 \log n)$ if there exist constants $c_1, c_2 > 0$ and n_0 such that for all $n \geq n_0$,

$$c_1 n^3 \log n \leq f(n) \leq c_2 n^3 \log n$$

For all $n \geq 1$,

$$4n^3 \log n \leq f(n) \leq 20n^3 \log n$$

So, $f(n)$ is sandwiched between constant multiples of $n^3 \log n$ for all large n.

Conclusion:

$$(n^2 + 1)(n + 1)\log(256n^4) \in \Theta(n^3 \log n)$$

by the definition of Θ-notation.

2. [7 marks] **Deciding presence of equidistant numbers.** Given an array of n integers $a_1, \ldots, a_n$, the problem is to decide if it contains a triplet of equidistant numbers. Another words, are there three different indices i, j, k such that $a_i - a_j = a_j - a_k$? For example, array $[7, 2, 3, 11, -1, 21]$ contains equidistant numbers; take $i = 4, j = 1, k = 3$ and verify that $a_i - a_j = a_j - a_k = 4$. On the other hand, array $[41, 4, 11, 21, -1]$ does not contain equidistant numbers. Straightforward algorithm (trying all possible triplets of different indices) has unacceptable running time in $\Theta(n^3)$. Design an efficient algorithm for this problem.

   The input is a list of integers. The output is *true* if it contains equidistant numbers or $false$ if it does not.

   Your algorithm must have worst case running time in $o(n^3)$.
   First find distinct indices i, j, k such that $a_i - a_j = a_j - a_k$. Rearrange: $a_i + a_k = 2a_j$.
   Algorithm Structure:

   1. Create a hash set S containing all array values.
   2. For each pair of distinct indices i, k ($i \neq k$):
      - Calculate $m = (a_i + a_k)/2$.
      - If m is integer and m is in S and $m \neq a_i$ and $m \neq a_k$, then return *true*.
   3. If no such m is found, returns $false$.

   Time Complexity:

   - There are $O(n^2)$ pairs (i, k).
   - Each lookup in S is $O(1)$.
   - Total time complexity: $O(n^2)$.
     Therefore this algorithm is much faster than the brute-force $O(n^3)$ approach and meets the $o(n^3)$ requirement.
     Psudocode:

```
function hasEquidistantNumbers(arr):
    S = set(arr)
    for i in range(len(arr)):
            for k in range(len(arr)):
                    if i == k:
                            continue
                    m = (arr[i] + arr[k]) / 2
                    if m.is_integer() and m in S and m != arr[i] and m != arr[k]:
                            return True
    return False
```

3. [10 marks] Arrange the functions given in list L by the order of growth from slowest to fastest.

   Use shorthand $f(n) \ll g(n)$ for $f(n) \in o(g(n))$ and $f(n) == g(n)$ for $f(n) \in \Theta(g(n))$.

   For example, for the list

$$2^n, n^2, \log n(n+1), n(n+1)$$

   the answer might look like

$$\log n(n+1) \ll n^2 == n(n+1) \ll 2^n$$

   or

$$\log n(n+1) \ll n(n+1) == n^2 \ll 2^n.$$

   You may use the following information:

$$1 \ll \log\log n \ll \log n \ll \log^2 n \ll \sqrt{n} \ll n \ll n \log n \ll n^2 \ll 2^n \ll n!$$

   Furthermore, for all positive real a and b the following holds:

   - $\log^a n \in o(n^b)$,
   - $n^a \in o(b^n)$ for $b > 1$,
   - and if $a < b$ then $n^a \in o(n^b)$ and $a^n \in o(b^n)$.

   Justify your choice of $\ll$ or $==$ between each pair of consecutive entries. For example, to show $\log n(n+1) \ll n^2$ first note, that $\log n < \log(n+1) \leq 2\log n$ for all $n \geq 2$, thus $\log(n+1) \in \Theta(\log n)$. Now, because $\log n(n+1) = \log n + \log(n+1) \in \Theta(\log n)$ and $\log n \in o(n^2)$ the claim follows.

   Here is the list L:

   $\log(3^n(n{+}1)n)$, $15n^2$, 3^n, $\dfrac{n}{10000}$, $\log^2(n^3)$, $\sqrt[4]{n}$, $n\log n^{1001}$, $16^{\log\sqrt{n}}$, $n2^n$, $2^{\sqrt{n}}$

Solution:

Each function in the list L:

- $\log(3^n(n+1)n) = \log(3^n) + \log(n+1) + \log(n) = n\log 3 + \log(n+1) + \log n \in \Theta(n)$
- $15n^2 \in \Theta(n^2)$
- 3^n is exponential
- $\frac{n}{10000} \in \Theta(n)$
- $\log^2(n^3) = (3\log n)^2 = 9\log^2 n \in \Theta(\log^2 n)$
- $\sqrt{\sqrt{n}} = n^{1/4}$
- $n\log n^{1001} = n \cdot 1001 \log n \in \Theta(n\log n)$
- $16^{\log\sqrt{n}} = 16^{(1/2)\log n} = 2^{2\cdot(1/2)\log n} = 2^{\log n} = n$ (since $a^{\log_b n} = n^{\log_b a}$)
- $n2^n = n \cdot 2^n$
- $2^{\sqrt{n}}$ is subexponential

Ordered from slowest to fastest:

$$\log^2(n^3) \ll \sqrt[4]{n} \ll \log(3^n(n+1)n) == \frac{n}{10000} == 16^{\log\sqrt{n}} \ll n(\log n)^{1001} \ll 15n^2 \ll 2^{\sqrt{n}} \ll n2^n \ll 3^n$$

Justification:

- $\log^2(n^3)$ is a polynomial in logarithm n, which makes it solwest slowest.
- $\sqrt[4]{n} = n^{1/4}$ which is slower than linear.
- $\log(3^n(n+1)n)$, $\frac{n}{10000}$, and $16^{\log\sqrt{n}}$ are all $\Theta(n)$.
- $n\log n^{1001}$ is $n\log n$, which grows faster than linear but slower than quadratic.
- $15n^2$ is quadratic.
- $2^{\sqrt{n}}$ is subexponential, which grows faster than any polynomial but slower than 3^n.
- $n2^n$ is exponential with a polynomial factor, which grows slower than 3^n for large n.
- 3^n is exponential.

4. [5 marks] Consider each of the following statements, assuming that all functions are non-negative:

   a) if $f_1(n) \in o(g(n))$ and $f_2(n) \in o(g(n))$, then $f_1(n)f_2(n) \in o(g(n))$;

   b) if $f_1(n) \in \Theta(g(n))$ and $f_2(n) \in \Theta(g(n))$, then $f_1(n)/f_2(n) \in \Theta(1)$;

   c) if $f(n) \in \Omega(g(n))$ then $3^{f(n)} \in \Omega(2^{g(n)})$.

   For each statement: if the statement is true then provide a proof that starts with the formal definition of the order notation utilized in the statement. If the statement is false then provide a counter example and demonstrate why the statement is false.

   Solution:

   a) if $f_1(n) \in o(g(n))$ and $f_2(n) \in o(g(n))$, then $f_1(n)f_2(n) \in o(g(n))$

   Answer: False

   Counterexample: Let $f_1(n) = f_2(n) = 1/\sqrt{n}$ and $g(n) = 1$. Then $f_1(n) \in o(1)$ and $f_2(n) \in o(1)$, but $f_1(n)f_2(n) = 1/n$, which is also $o(1)$ in this case. However, if $g(n)$ grows, for example $g(n) = n$, then $f_1(n) = f_2(n) = n^{1/3}$, $f_1(n) \in o(n)$, $f_2(n) \in o(n)$, but $f_1(n)f_2(n) = n^{2/3} \notin o(n)$ (since $n^{2/3}/n = n^{-1/3} \to 0$). But the statement is not always true for arbitrary $g(n)$ and functions. The correct counterexample is:

   Let $f_1(n) = f_2(n) = n^{0.6}$, $g(n) = n$. Then $f_1(n) \in o(n)$, $f_2(n) \in o(n)$, but $f_1(n)f_2(n) = n^{1.2} \notin o(n)$.

   b) if $f_1(n) \in \Theta(g(n))$ and $f_2(n) \in \Theta(g(n))$, then $f_1(n)/f_2(n) \in \Theta(1)$

   Answer: True

   Proof: By definition, $f_1(n) \in \Theta(g(n))$ means there exist constants $c_1, c_2 > 0$ and n_0 such that $c_1 g(n) \leq f_1(n) \leq c_2 g(n)$ for all $n \geq n_0$. Similarly, $d_1 g(n) \leq f_2(n) \leq d_2 g(n)$ for all $n \geq n_0$. So,

   $$\frac{c_1}{d_2} \leq \frac{f_1(n)}{f_2(n)} \leq \frac{c_2}{d_1}$$

   for all $n \geq n_0$. Thus, $f_1(n)/f_2(n) \in \Theta(1)$.

c) if $f(n) \in \Omega(g(n))$ **then** $3^{f(n)} \in \Omega(2^{g(n)})$

Answer: True

Proof: By definition, $f(n) \in \Omega(g(n))$ means there exist $c > 0$ and n_0 such that $f(n) \geq cg(n)$ for all $n \geq n_0$. Then,

$$3^{f(n)} \geq 3^{cg(n)} = (3^c)^{g(n)}$$

Since $3^c > 2$ for any $c > \log_3 2$, we have $3^{f(n)} \geq C \cdot 2^{g(n)}$ for some $C > 0$ and all large n. Thus, $3^{f(n)} \in \Omega(2^{g(n)})$.

5. [4 marks] Analyze the following pseudocode and give a tight (Θ) bound on the running time as a function of n. Assume all individual instructions are elementary. Show your work.

```
l := 0; s := 0; m := 1;
for i = 1 to n do
    for j = 1 to i do
        l := l + 2*i + 3*j
    od
od
while m <= n do
    for j = 1 to m do
        s := s + n - 2*m
    od
    m := 2*m
od
```

First part:

```
for i = 1 to n:
    for j = 1 to i:
        l := l + 2*i + 3*j
```

The inner loop runs i times for each i from 1 to n.
Total iterations: $\sum_{i=1}^{n} i = n(n+1)/2 \in \Theta(n^2)$.

Second part:

```
m := 1
while m <= n:
    for j = 1 to m:
        s := s + n - 2*m
    m := 2*m
```

The value of m doubles each time: $m = 1, 2, 4, 8, \ldots, n$.
Number of times the while loop runs: $\log_2 n + 1 \in \Theta(\log n)$.
For each iteration, the inner for-loop runs m times. The total work is:

$$\sum_{k=0}^{\lfloor \log_2 n \rfloor} 2^k = 2^{\log_2 n + 1} - 1 = 2n - 1 \in \Theta(n)$$

Total run time:

- First part: $\Theta(n^2)$
- Second part: $\Theta(n)$

Therefore run time is

$$\Theta(n^2)$$

since n^2 dominates n for large n.