Computer Science 3600 (Winter 2027):
Problem-set #1
Answers

**1.** For the algorithm below, derive a **worst-case** time complexity function $T(n)$.

**Answer:** $T(n) = 3n^2 \log_2 n + 9n^2 + 9n + 5$ (see Figure 1)

**2.** For the algorithm below, derive an **asymptotic worst-case**, *i.e.*, Big-Oh, complexity function $O(f(n))$. Briefly explain the reasoning behind each derivation.

```
sum = 42
for i = 1 to n * log(n) do
    j = 1
    finished = false
    for k = 1 to  n do
        if COND(sum)
            sum = sum / (k * i) + j
        while ((j <= n) and (not finished)) do
            finished = true
```

Note that method `COND()` runs in $(n + 13)$ timesteps.

**Answer:** Note that the `while`-loop executes exactly once for each iteration of the innermost `for`-loop as the variable `finished` is set to `true` during the first loop iteration and hence causes the `while`-loop condition to evaluate to false on the second iteration. As method `COND()` is evaluated each time the innermost `for`-loop executes and the outermost and innermost `for`-loops execute $n \log_2 n$ and $n$ times, respectively, the algorithm runs in $O(n \log_2 n \times n \times (n + 13)) = O(n^3 \log_2 n)$ time.

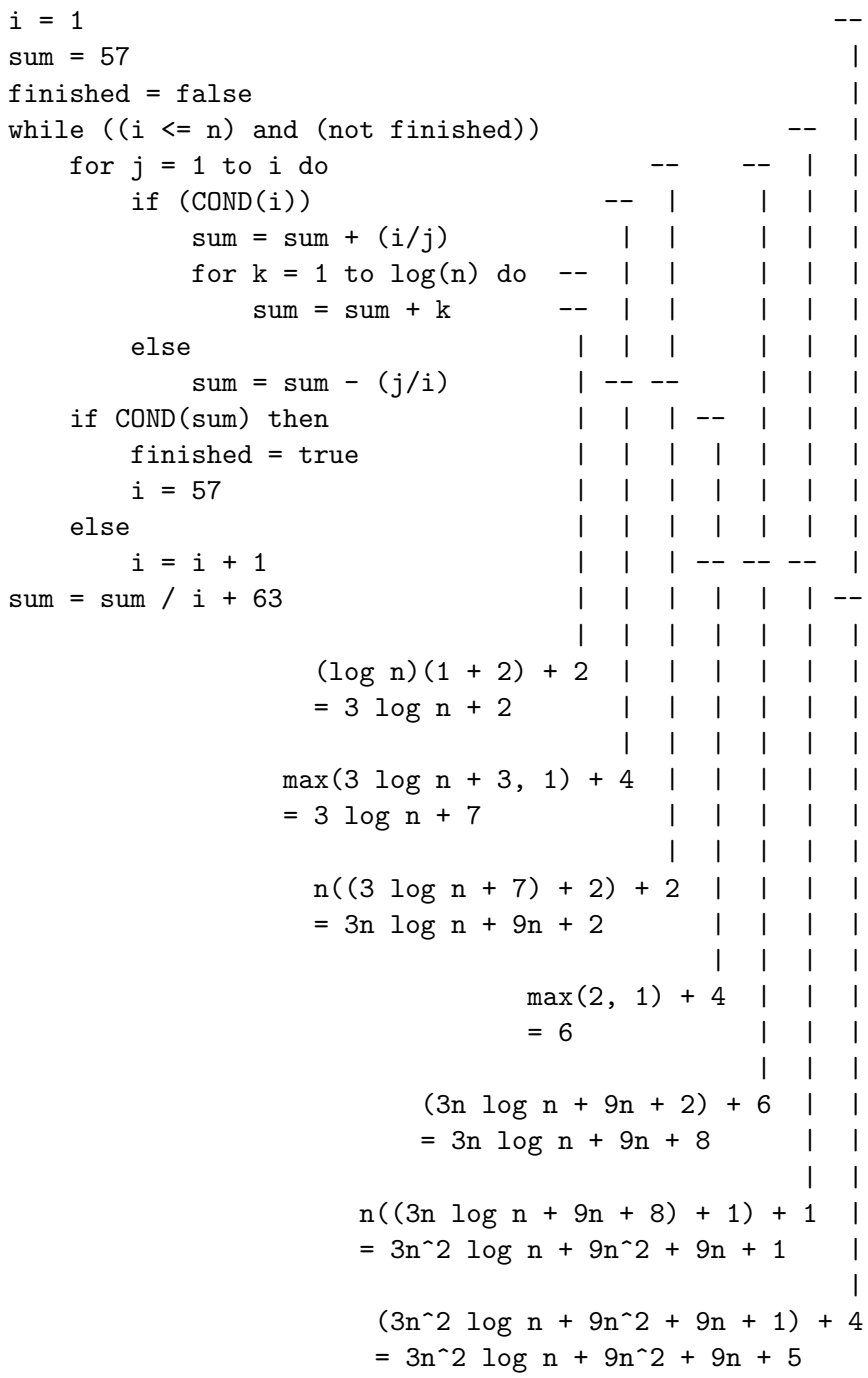



Figure 1: Worst-case Time Complexity Derivation, Question #1. Note that method `COND()` runs in 4 timesteps.

**3.** For the algorithm below, derive a parameterized asymptotic worst-case time complexity function.

```
sum = 0
tsum = -15
for i = 1 to n do
    x = P1(n)
    sum = sum - x + 5
    for j = 1 to n * n do
        y = x / (P2(n) + P1(n))
    if (P3(n))
        if (P4(n))
            tsum = tsum + tsum
    else
        for j = 1 to log(n) do
    if (P4(n))
       y = y * i - j
    tsum = tsum / y
sum = sum - tsum * tsum
```

**Answer:**

$n(T(P1) + n^2(T(P1) + T(P2)) + T(P3) + \max(T(P4), \log_2 n) + T(P4))$
$= O(n^3)T(P1) + O(n^3)T(P2) + O(n)T(P3) + O(n \max(T(P4), \log_2 n))$
$= O(n^3 T(P1) + n^3 T(P2) + nT(P3) + (n \times \max(T(P4), \log_2 n)))$

**4.** Prove or disprove the following:

**a)** $f(n) = (n-2)(n-6)$ is not $\Theta(n^2)$.

**b)** $f(n) = n^d + 10n^2$, where $d$ is some integer constant greater than or equal to 2, is $O(n^d)$.

**c)** $f(n) = 10^{127}2^n$ is $\Omega(3^n)$.

**Answer:** In each case below, we will work from the definitions for $O(g(n))$ and $\Omega(g(n))$, either to show that the inequalities in these definitions hold or that we can derive a contradiction.

- Proof that $f(n) = (n-2)(n-6)$ is $\Theta(n^2)$: As $(n-6)(n-2) = n^2 - 8n + 12$, this can be rewritten as $n^2 - 8n + 12 \leq c_1 n^2$ (the big-Oh part) and $n^2 - 8n + 12 \geq c_2 n^2$ (the big-Omega part). The first inequality holds for $c_1 = 1$ and $n_{0,1} = 8$ and the second inequality holds for $c_2 = \frac{1}{8}$ and $n_{0,2} = 8$.
- Proof that $f(n) = n^d + 10n^2$, where $d$ is some integer constant greater than or equal to 2, is $O(n^d)$: This can be rewritten as $n^d + 10n^2 \leq cn^d$. This inequality holds for $c = 11$ and $n_0 = 1$ when $d \geq 2$.

- Proof that $f(n) = 10^{127}2^n$ is not $\Omega(3^n)$: This can be rewritten as follows:

$$\begin{aligned} 10^{127}2^n &\geq c3^n \\ \log_2(10^{127}2^n) &\geq \log_2(c3^n) \\ \log_2 10^{127} + \log_2 2^n &\geq \log_2 c + \log_2 3^n \\ 127\log_2 10 + n &\geq \log_2 c + n\log_2 3 \\ 127\log_2 10 + (n - n\log_2 3) &\geq \log_2 c \end{aligned}$$

  As $\log_2 3 > 1$, the quantity $n - n\log_2 3$ is negative for positive values of $n$; moreover, this quantity goes to negative infinity as $n$ goes to infinity. Therefore, this inequality is false for any $c$ for sufficiently large values of $n$.

**5.** Determine the longest common subsequence (LCS) of the strings GAAGCCTA and TATCGA using the algorithms given on pages 394 and 395 of the textbook. Show the filled-in dynamic programming matrix, all matrix-cell backpointers, the backpointer path that gives an optimal LCS, and the LCS associated with that path.

**Answer:** The requested table is given in Table 1. Note that the algorithm on page 395 has a rigid order for selecting backpointers (diagonal-up-across) that assigns only one backpointer per cell, regardless of how many of the three associated previously-computed cells yield an optimal cost. This algorithm, by preferring "up" backpointers to "across" backpointers, effectively restricted the derivable LCS to the one that ended furthest to the right in the the given sequence labeling the top of the table.

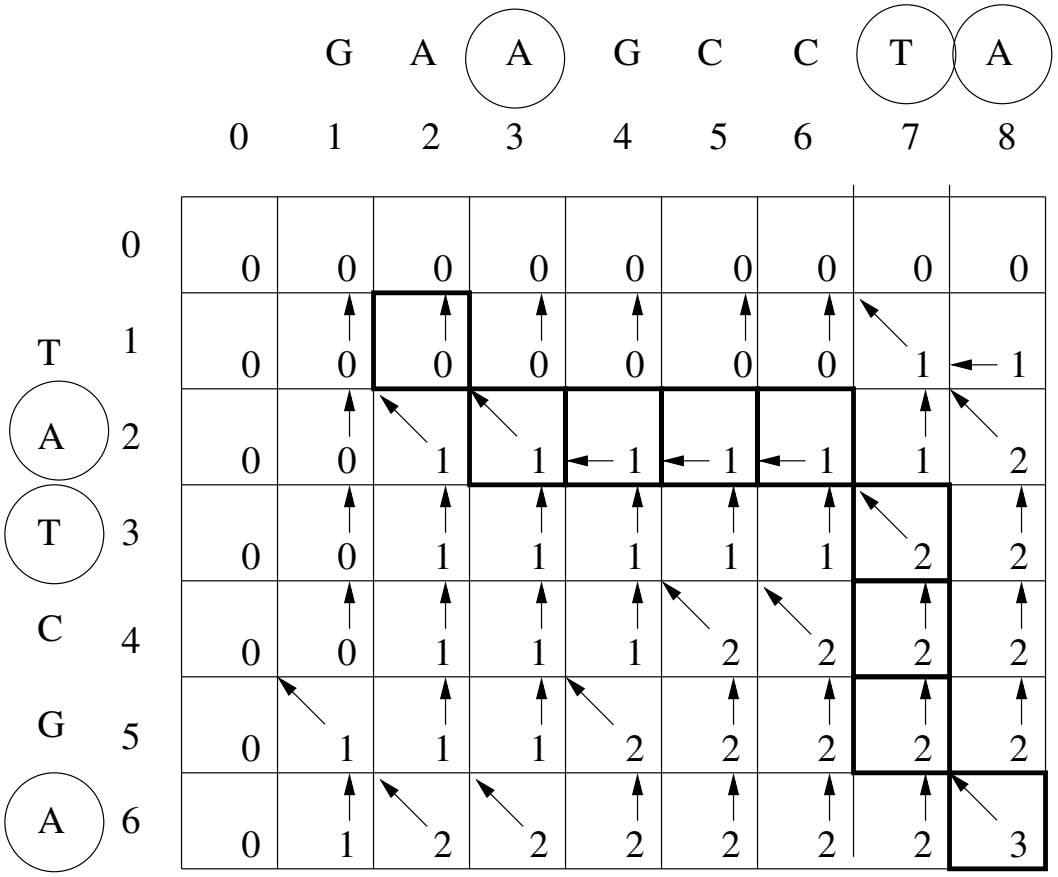



Table 1: Answer for Question #5 (textbook-specified backpointers).