Due: Friday 8 April, *before* 17:00

Note: **solutions may be incomplete, and meant to be used as guidelines only**. We encourage you to ask follow-up questions on the course forum or during office hours.

1. [**8 marks**] **Nested loops.**

Consider the following algorithm. (It doesn't **do** anything, but it sure wastes a bunch of time doing it!)

```
1  def twisty_too(n: int) -> None:
2      ''' Precondition: n > 0. '''
3      i = n
4      while i > 0:          # Loop 1
5          s = -1
6          j = 0
7          while j < i:      # Loop 2
8              s = s + 2
9              j = j + s
10         i = i - 1
11         while i % 4 > 0:  # Loop 3
12             i = i - 1
```

(a) [**2 marks**] Find a *lower bound* on **the number of iterations** of Loop 1, as a function of the input n, *without using Omega or Theta notation*. Show your work (in other words, explain how you obtained your answer and show your calculations). *Hint*: Don't try to find the exact number of iterations.

> **Solution:** The value of i decreases by *at most* 4 after each iteration (because Loop 3 iterates at most 3 times), so it will take *at least* $\lceil n/4 \rceil$ iterations for i to decrease from n to some value ≤ 0.

(b) [**2 marks**] Find an *upper bound* on **the number of iterations** of Loop 1, as a function of the input n, *without using Big-O or Theta notation*. Show your work (in other words, explain how you obtained your answer and show your calculations). *Hint*: Don't try to find the exact number of iterations.

> **Solution:** The value of i decreases by *at least* 1 after each iteration (because of line 10), so it will take *at most* n iterations for i to decrease from n to some value ≤ 0.

(c) [**4 marks**] Give a Theta bound on the running time function $RT(n)$ for algorithm `twisty_too`. Show your work (in other words, explain how you obtained your answer and show your calculations).

HINT: $\displaystyle\sum_{i=0}^{n-1}(2i+1) = n^2$.

> **Solution:** During the execution of Loop 2, $s_k = 2k - 1$ and $j_k = k^2$ (from the hint). So Loop 2 performs $\lceil \sqrt{i} \rceil$ iterations, each one taking constant time. In addition, Loop 3 performs between 0 and 3 constant-time iterations, so the body of Loop 1 takes time $\Theta(\sqrt{i})$. From the bounds proved in the previous two parts, Loop 1 iterates over $\Theta(n)$ many different values of i between 0 and n. So $RT(n) \in \Theta(n\sqrt{n}) = \Theta(n^{3/2})$.

More formally,

$$RT(n) \leq \sum_{k=1}^{n} \left\lceil \sqrt{k} \right\rceil \leq \sum_{k=1}^{n} \lceil \sqrt{n} \rceil = n\sqrt{n} \in \mathcal{O}\big(n\sqrt{n}\,\big)$$

and

$$RT(n) \geq \sum_{k=1}^{\left\lfloor \frac{n}{4} \right\rfloor} \big(\sqrt{4k}\,\big) \geq \sum_{k=\left\lceil \frac{n}{8} \right\rceil}^{\left\lfloor \frac{n}{4} \right\rfloor} \sqrt{\frac{n}{2}} \geq \left(\frac{n}{8} - 1\right) \sqrt{\frac{n}{2}} \in \Omega\big(n\sqrt{n}\,\big)$$

so $RT(n) \in \Theta\big(n\sqrt{n}\,\big)$.

2. [13 marks] Worst-case and Best-case.

Consider the following algorithm.

```python
def long_prod(lst: list, t: int) -> int:
    ''' Return the maximum length of any slice of lst whose product is at most t.
        Preconditions: t > 0; lst is non-empty; every element of lst is positive.
    '''
    m = 0  # max length found so far
    for i in range(1, len(lst) + 1):       # Loop 1
        j = i - 1
        p = 1  # product of lst[j+1:i]
        while j >= 0 and p * lst[j] <= t:  # Loop 2
            p = p * lst[j]
            j = j - 1
        j = j + 1
        if i - j > m:
            m = i - j
    return m
```

(a) [**2 marks**] Find, with proof, an input family for which the running time of `long_prod` is $\Theta(n^{4/3})$. Show your work.

> **<u>Solution:</u>** Let $n \in \mathbb{Z}^+$ and `lst` $= [2, \ldots, 2]$ (n copies of 2) and $t = 2^{\lfloor \sqrt[3]{n} \rfloor}$. When executing `long_prod(lst,t)`, Loop 2 iterates for each value of $j = i-1, i-2, \ldots, i - \lfloor \sqrt[3]{n} \rfloor - 2$, because the value of p after k complete iterations is equal to $p_k = 2^k$ (since p is multiplied by $\texttt{lst}[j] = 2$ at each iteration), and the loop stops when $p_k \cdot \texttt{lst}[j] > t \Leftrightarrow k \geq \lfloor \sqrt[3]{n} \rfloor$ (or when $j = -1$). So the body of Loop 1 takes time $\Theta(\sqrt[3]{n})$ for each value of $i = 0, 1, \ldots, n - \lfloor \sqrt[3]{n} \rfloor - 1$, and time $\Theta(n-i)$ for $i = n - \lfloor \sqrt[3]{n} \rfloor, \ldots, n-1$. Over every iteration for Loop 1, this means that
>
> $$n\sqrt[3]{n} - (\sqrt[3]{n})^2 \leq \sum_{i=0}^{n - \lfloor \sqrt[3]{n} \rfloor - 1} \sqrt[3]{n} \leq RT(\texttt{lst}, t) \leq \sum_{i=0}^{n-1} \sqrt[3]{n} = n\sqrt[3]{n}$$
>
> so $RT(\texttt{lst}, t) \in \Theta(n^{4/3})$.

(b) [**3 marks**] Find, with proof, an **upper bound** on the **worst-case** running time of `long_prod`. Show your work. For full marks, your upper bound must match the lower bound from the next part.

> **<u>Solution:</u>** Let $n \in \mathbb{Z}^+$ and `lst,t` be arbitrary inputs of size n. When `long_prod(lst,t)` runs, Loop 2 iterates *at most* $i \leq n$ times, and each iteration takes constant time, so the body of Loop 1 takes time $\leq n$. Loop 1 iterates n times, so $RT(\texttt{lst}, t) \leq n^2$. Hence, $WC(n) \in \mathcal{O}(n^2)$.

(c) [**3 marks**] Find, with proof, a **lower bound** on the **worst-case** running time of `long_prod`. Show your work. For full marks, your lower bound must match the upper bound from the previous part.

Solution: Let $n \in \mathbb{Z}^+$ and $\texttt{lst} = [1, \ldots, 1]$ (n copies of 1) and $t = 2$. When executing `long_prod(lst,t)`, Loop 2 iterates for each value of $j = i-1, i+1, \ldots, 0$ (because $p * \texttt{lst}[j] \leq t$ is always true since p is always equal to 1), so the body of Loop 1 takes time $\geq i$. Over every iteration for Loop 1, this means that $RT(\texttt{lst}, t) \geq \sum_{i=0}^{n-1} i = n(n-1)/2$. Hence, $WC(n) \in \Omega(n^2)$.

(d) [**5 marks**] Recall that the **best-case running time** of an algorithm is defined as follows:

$$BC(n) = \min\{RT(x) \mid x \in \mathcal{I}_n\}$$

where $RT(x)$ is the running time of the algorithm on input x, and $\mathcal{I}_n$ is the set of all inputs of size n. (As discussed in lecture, this is similar to the definition of worst-case running time, but with *min* in place of *max*.)

Find, with proof, a **tight bound** on the **best-case** running time of `long_prod`. Your analysis should consist of two separate proofs for matching upper and lower bounds on the best-case running time.

Solution: *Upper bound*: Let $n \in \mathbb{Z}^+$ and $\texttt{lst} = [2, \ldots, 2]$ (n copies of 2) and $t = 1$. When executing `long_prod(lst,t)`, Loop 2 iterates 0 times because the loop condition is False the first time it is encountered: $p * \texttt{lst}[j] = 1 \cdot 2 \not\leq 1 = t$. so the body of Loop 1 takes time ≤ 1. Since Loop 1 iterates n times, $RT(lst, t) \leq n$. Hence, $BC(n) \in \mathcal{O}(n)$.

Lower bound: Let $n \in \mathbb{Z}^+$ and `lst,t` be arbitrary inputs of size n. When `long_prod(lst,t)` runs, the body of Loop 1 executes *at least* a constant number of steps, and since Loop 1 iterates n times, $RT(lst, t) \geq n$. Hence, $BC(n) \in \Omega(n)$.

Therefore, $BC(n) \in \Theta(n)$.

3. **[8 marks] Average-case analysis.**

Consider the following algorithm.

```
1 def alpha_min(s: str) -> int:
2     ''' Return the smallest index k such that s[k:len(s)] is sorted.
3         Precondition: s is non-empty. '''
4     i = len(s) - 1
5     while i > 0 and s[i-1] <= s[i]:
6         i = i - 1
7     return i
```

For each $n \in \mathbb{N}$ with $n \geq 2$, let $\mathcal{I}_n$ be the set that contains all strings of length n with 2 b's and $(n-2)$ a's, in any order. (For example, $\mathcal{I}_4 = \{aabb, abab, abba, baab, baba, bbaa\}$.)

Note that $|I_n| = \binom{n}{2} = \frac{n(n-1)}{2}$ because each element of I_n is made up of n individual characters, all but two of which are equal to a, and there are exactly $\binom{n}{2}$ many different ways to choose the 2 positions that will contain b.

(a) [**1 mark**] Let $n \in \mathbb{N}$ with $n \geq 2$, and let k be the value returned by `alpha_min`(s), for some input $s \in \mathcal{I}_n$. Write an expression for the "exact" number of steps executed by `alpha_min`(s), *in terms of n and k*.

Show your work (explain how you count your steps and how you arrive at your answer).

> **Solution:** If `alpha_min` returns k, then the loop must have performed exactly $n-1-k$ iterations (because each iteration subtracts 1 from i and the initial value of i is $n-1$). If we count 1 step for each iteration of the loop, and 1 extra step for lines 4 and 7 (initialization and return), the exact number of steps executed is therefore equal to $n-k$.

(b) [**1 mark**] What is the exact average-case running time of `alpha_min` over the set of inputs $\mathcal{I}_4$? Give your answer in the form of a simplified, concrete rational number (like 17/5).

Show your work (explain what you are calculating at each step).

> **Solution:**
>
> $$\begin{aligned} AC(4) &= \frac{1}{|\mathcal{I}_4|} \sum_{s \in \mathcal{I}_4} RT(s) \\ &= \frac{1}{6}\big(RT(aabb) + RT(abab) + RT(abba) + RT(baab) + RT(baba) + RT(bbaa)\big) \\ &= \frac{4+2+1+3+1+2}{6} = \frac{13}{6} \end{aligned}$$

(c) [**3 marks**] For each $n \in \mathbb{N}$ such that $\mathcal{I}_n$ is defined, and each possible return value k for `alpha_min`, give an exact expression for **the number of inputs** $s \in \mathcal{I}_n$ for which `alpha_min`(s) returns k. In other words, calculate $\big|\{s \in \mathcal{I}_n \mid \texttt{alpha\_min}(s) \text{ returns } k\}\big|$.

Show your work (explain how you obtain your expression, and how it relates to the algorithm).

> **Solution:** `alpha_min`(s) returns k for all strings s such that $s[k] \leq \cdots \leq s[n-1]$, but $s[k-1] > s[k]$ (or $k = 0$).
>
> For each value of $n \in \mathbb{N}$ with $n \geq 2$, and each $k \in \{0, \ldots, n-1\}$:

- If $k = 0$, there is exactly 1 ($= k + 1$) possible input: $\overbrace{a \cdots a}^{n-2} bb$.
- If $k = n - 1$, there are $n - 2$ ($= k - 1$) inputs that end with ba: one for each of the possible positions for the second b among the first $n - 2$ characters.
- If $0 < k < n - 1$, there are exactly k many inputs possible:
  - there are $k - 1$ inputs s with $s[k - 1] = b$ and $s[k] = a$, followed by $n - 1 - k$ a's: one for each of the possible positions for the second b among the first $k - 1$ characters, and
  - there is one more input $s = \underbrace{a \cdots a}_{k-1} ba \underbrace{a \cdots a}_{n-2-k} b$.

(d) **[3 marks]** Perform an average-case analysis of `alpha_min`, for the input set $\mathcal{I}_n$ defined above. **Give an exact expression** (without using Big-O / Omega / Theta).

Show your work. In particular, your answer should be expressed in the form of a sum before you simplify it to a closed-form expression.

Hint: You may use the following fact.

$$\sum_{i=1}^{n} i^2 = \frac{1}{6} n(n + 1)(2n + 1) \tag{1}$$

Solution:

$$\begin{aligned}
AC(n) &= \frac{1}{|\mathcal{I}_n|} \sum_{s \in \mathcal{I}_n} RT(s) \\
&= \frac{1}{\binom{n}{2}} \sum_{k=0}^{n-1} \sum_{\substack{s \in \mathcal{I}_n \\ \texttt{alpha_min(s)} = k}} (n - k) && \text{(from part (a))} \\
&= \frac{2}{n(n-1)} \sum_{k=0}^{n-1} (n - k) \cdot \big|\{s \in \mathcal{I}_n \mid \texttt{alpha_min(s)} = k\}\big| \\
&= \frac{2}{n(n-1)} \cdot (n - 0) \cdot 1 + \frac{2}{n(n-1)} \sum_{k=1}^{n-2} (n - k) \cdot k \\
&\quad + \frac{2}{n(n-1)} \cdot (n - (n - 1)) \cdot (n - 2) && \text{(from part (c))} \\
&= \frac{2}{n-1} + \frac{2}{n-1} \left(\sum_{k=1}^{n-2} k \right) - \frac{2}{n(n-1)} \left(\sum_{k=1}^{n-2} k^2 \right) + \frac{2(n-2)}{n(n-1)} \\
&= \frac{2}{n-1} + \frac{2}{n-1} \cdot \frac{(n-1)(n-2)}{2} \\
&\quad - \frac{2}{n(n-1)} \cdot \frac{(n-2)(n-1)(2n-3)}{6} + \frac{2(n-2)}{n(n-1)} \\
&= \frac{n^2 + n + 6}{3n}
\end{aligned}$$