Due before 17:00 (EDT) on Tuesday 6 April

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. [10 marks] Analyzing nested loops.

(a) [**4 marks**] Analyze the running time of function `nested3` below, in terms of its input n, concluding with a Theta bound on the running time.

```
1  def nested3(n: int) -> None:
2      i = 1
3      while i < n:            # Loop 1
4          j = i
5          while j > 1:        # Loop 2
6              k = 0
7              while k < n:    # Loop 3
8                  k = k + 2
9              j = j // 2
10         i = i * 2
```

Solution

Approach A: calculate "exact" steps and derive Theta bound at the end.

- Loop 3 performs $\lceil n/2 \rceil$ iterations, each one taking time 1, for a total of $\lceil n/2 \rceil$ steps.
- Loop 2 performs $\lfloor \log_2 i \rfloor$ iterations, each one taking time $\lceil n/2 \rceil + 1$ (the "$+1$" is for lines 6 and 9), for a total of $\lfloor \log_2 i \rfloor (\lceil n/2 \rceil + 1)$ steps.
- Loop 1 performs $\lceil \log_2 n \rceil$ iterations, where the values of i go from $1 = 2^0$ up to $2^{\lceil \log_2 n \rceil - 1}$, and each iteration takes time $\lfloor \log_2 i \rfloor (\lceil n/2 \rceil + 1) + 1$ (the "$+1$" is for lines 4 and 10). With the addition of one more step for line 2, this gives the following total number of steps

$$
\begin{aligned}
& 1 + \sum_{k=0}^{\lceil \log_2 n \rceil - 1} \left(\left\lfloor \log_2(2^k) \right\rfloor (\lceil n/2 \rceil + 1) + 1 \right) \\
&= 1 + \sum_{k=0}^{\lceil \log_2 n \rceil - 1} \left(k(\lceil n/2 \rceil + 1) + 1 \right) \\
&= 1 + (\lceil n/2 \rceil + 1) \sum_{k=0}^{\lceil \log_2 n \rceil - 1} k + \sum_{k=0}^{\lceil \log_2 n \rceil - 1} 1 \\
&= 1 + (\lceil n/2 \rceil + 1) \frac{\lceil \log_2 n \rceil (\lceil \log_2 n \rceil - 1)}{2} + \lceil \log_2 n \rceil \\
&\in \Theta(n (\log n)^2)
\end{aligned}
$$

Approach B: prove separate, but matching upper and lower bounds.

Upper bound:

- Loop 3 performs no more than n iterations, each one taking constant time, for a total time $\leq n$.

- Loop 2 performs no more than $\log_2 n$ iterations, each one taking time $\leq n$, for a total number of steps $\leq n \log_2 n$.
- Loop 1 performs no more than $\log_2 n$ iterations, each one taking time $\leq n \log_2 n$, for a total number of steps $\leq n(\log_2 n)^2$.
- Therefore, the running time is $\mathcal{O}(n(\log n)^2)$.

Lower bound:

- Loop 3 performs at least $n/2$ iterations, each one taking constant time, for a total time $\geq n/2$.
- Loop 2 performs exactly $\log_2 i$ iterations, each one taking time $\geq n/2$, for a total number of steps $\geq (n/2)\log_2 i$.
- Loop 1 performs at least $\log_2 n$ iterations for values of i from $1 = 2^0$ up to $2^{\lceil \log_2 n \rceil - 1}$, and each iteration takes time $\geq (n/2)\log_2 i$, for a total number of steps that is

$$
\begin{aligned}
&\geq \sum_{k=0}^{\log_2 n - 1} \frac{n}{2} \log_2(2^k) \\
&= \frac{n}{2} \sum_{k=0}^{\log_2 n - 1} k \\
&= \frac{n}{2} \frac{\log_2 n(\log_2 n - 1)}{2}
\end{aligned}
$$

- Therefore, the running time is $\Omega(n(\log n)^2)$.

(b) [**4 marks**] Analyze the running time of function `up_and_down` below, in terms of its input n, concluding with a Theta bound on the running time.

```python
def up_and_down(n: int) -> None:
    i = 0
    while i < n:            # Loop 1
        j = i
        if i % 2 == 1:
            while j > 0:    # Loop 2
                j = j - 1
                print(j)
        else:
            while j < n:    # Loop 3
                j = j + 1
                print(j)
        i = i + 1
```

Solution

Approach A: calculate "exact" steps and derive Theta bound at the end.

- When it executes, Loop 2 performs i iterations, each one taking time 1, for a total of i steps.
- When it executes, Loop 3 performs $n - i$ iterations, each one taking time 1, for a total of $n - i$ steps.
- Loop 1 performs n iterations, where the iterations for $i = 0, 2, 4, \ldots$ take time $n - i + 1$ and

the iterations for $i = 1, 3, 5, \ldots$ take time $i+1$ (the "$+1$" accounts for lines 4, 5, and 13). To make the final expression easier to express, we split up the proof into subcases, depending on whether n is odd or even.

– If n is even, then the total time taken by the algorithm is given by the following expression (where the initial "$1+$" accounts for line 2):

$$
\begin{aligned}
&1 + \left(\sum_{k=0}^{(n-2)/2} 2k+1+1\right) + \left(\sum_{k=0}^{(n-2)/2} n-2k+1\right) \\
&= 1 + \sum_{k=0}^{(n-2)/2} (2k+2+n-2k+1) \\
&= 1 + \sum_{k=0}^{(n-2)/2} (n+3) \\
&= 1 + \frac{n(n+3)}{2}
\end{aligned}
$$

– If n is odd, then the total time taken by the algorithm is given by the following expression (where the initial "$1+$" accounts for line 2):

$$
\begin{aligned}
&1 + \left(\sum_{k=0}^{(n-3)/2} 2k+1+1\right) + \left(\sum_{k=0}^{(n-1)/2} n-2k+1\right) \\
&= 1 + \left(\sum_{k=0}^{(n-3)/2} (2k+2+n-2k+1)\right) + (n-(n-1)+1) \\
&= 1 + \left(\sum_{k=0}^{(n-3)/2} (n+3)\right) + 2 \\
&= 3 + \frac{(n-1)(n+3)}{2}
\end{aligned}
$$

In both cases, the running time is $\Theta(n^2)$.

Approach B: prove separate, but matching upper and lower bounds.

Upper Bound:

- Loop 2 iterates $i \le n$ times and each iteration takes time 1, for a total time $\le n$.
- Loop 3 iterates $n - i \le n$ times and each iteration takes time 1, for a total time $\le n$.
- Loop 1 iterates n times and each iteration takes time $\le n$, for a total time $\le n^2$.
- So the running time is $\mathcal{O}(n^2)$.

Lower Bound:

- There are at least $n/4$ values of $i \ge n/2$ that are odd. For each of these values, Loop 2 performs at least $n/2$ iterations, each one taking time 1, for a total number of steps $\ge n/2$.
- There are at least $n/4$ values of $i \le n/2$ that are even. For each of these values, Loop 3 performs at least $n - i \ge n/2$ iterations, each one taking time 1, for a total number of steps $\ge n/2$.

- Therefore, there are at least $n/4 + n/4 = n/2$ iterations of Loop 1 that each take time $\geq n/2$, so the total runtime for the algorithm is $\geq (n/2)^2 = n^2/4$.
- So the running time is $\Omega(n^2)$.

(c) [**2 marks**] Find, with proof, an **exact** expression for the number of `print` statements executed by function `up_and_down` from the previous part, in terms of its input n.

(*Hint*: You may want to introduce cases for n.)

Solution

- When it executes, Loop 2 performs i iterations, each one calling `print` exactly once.
- When it executes, Loop 3 performs $n - i$ iterations, each one calling `print` exactly once.
- Loop 1 performs n iterations, where the iterations for $i = 0, 2, 4, \dots$ call `print` $n - i$ times and the iterations for $i = 1, 3, 5, \dots$ call `print` i times. To make the final expression easier to express, we split up the proof into subcases, depending on whether n is odd or even.
  - If n is even, then the total number of calls to `print` is given by:

$$\begin{aligned}
&\left(\sum_{k=0}^{(n-2)/2} 2k+1\right) + \left(\sum_{k=0}^{(n-2)/2} n-2k\right) \\
&= \sum_{k=0}^{(n-2)/2} (2k+1+n-2k) \\
&= \sum_{k=0}^{(n-2)/2} (n+1) \\
&= \frac{n(n+1)}{2} = \frac{n^2+n}{2}.
\end{aligned}$$

  - If n is odd, then the total number of calls to `print` is given by:

$$\begin{aligned}
&\left(\sum_{k=0}^{(n-3)/2} 2k+1\right) + \left(\sum_{k=0}^{(n-1)/2} n-2k\right) \\
&= (n-(n-1)) + \sum_{k=0}^{(n-3)/2} (2k+1+n-2k) \\
&= 1 + \sum_{k=0}^{(n-3)/2} (n+1) \\
&= \frac{2+(n-1)(n+1)}{2} = \frac{n^2+1}{2}.
\end{aligned}$$

2. [10 marks] Worst-case analysis.

Consider the following function:

```
1 def some(lst: list, s: int) -> bool:
2     """Precondition: lst is a non-empty list of integers."""
3     for i in range(len(lst)):            # Loop 1
4         for j in range(i):               # Loop 2
5             if lst[i] + lst[j] == s:
6                 return True
7     return False
```

(a) [**2 marks**] Find, with proof, an asymptotic upper bound (Big-O) on the worst-case running time of `some`.

Solution

Let $n \in \mathbb{Z}^+$ and `lst,s` be any input with `len(lst)` $= n$.

- The body of Loop 2 takes time 1 (constant time), and Loop 2 iterates $i \leq n$ times, so Loop 2 takes time $\leq n$.
- Loop 1 iterates n times, and each iteration takes time $\leq n$ for a total number of steps $\leq n^2$.

Since this applies no matter the input, the worst-case running time is in $\mathcal{O}(n^2)$.

(b) [**3 marks**] Find, with proof, an asymptotic lower bound (Omega) on the worst-case running time of `some`, **that matches your upper bound**.

Solution

Let $n \in \mathbb{Z}^+$. Let `lst` $= [0, 0, \ldots, 0]$ (n elements all equal to 0) and `s` $= 1$.

- By our choice of `lst` and `s`, the condition of the if statement will never evaluate to True: `lst[i]` $+$ `lst[j]` $= 0 \neq 1 =$ `s` for all `i` and `j`.
- So Loop 2 always performs i iterations, each one taking constant time.
- So the body of Loop 1 takes time i, and Loop 1 iterates over every value of $i = 0, 1, \ldots, n-1$, for total time equal to $\displaystyle\sum_{i=0}^{n-1} i = \frac{n(n-1)}{2}$.

Since there is at least one input of size n for which the algorithm takes time $\Omega(n^2)$, the worst-case running time is in $\Omega(n^2)$.

(c) [**5 marks**] Find, with proof, an input family for which the running time of `some` is $\Theta($`len(lst)`$)$.

Solution

Let $n \in \mathbb{Z}^+$. Let `lst` contain $\lfloor\sqrt{n}\rfloor - 1$ many 0's followed by $n - \lfloor\sqrt{n}\rfloor + 1$ many 1's, and let `s` $= 2$. In other words, `lst[k]` $= 0$ for $k = 0, 1, \ldots, \lfloor\sqrt{n}\rfloor - 2$ (if $n \geq 4$) and `lst[k]` $= 1$ for $k = \lfloor\sqrt{n}\rfloor - 1, \lfloor\sqrt{n}\rfloor, \ldots, n-1$.

- For each value of $i = 0, 1, \ldots, \lfloor\sqrt{n}\rfloor - 1$, and all values of $j = 0, 1, \ldots, i-1$, the if condition on line 5 evaluates to False (because `lst[i]` $+$ `lst[j]` $=$ `lst[i]` $+ 0 < 2$). So Loop 2 performs i iterations, each one taking time 1, for each $i = 0, 1, \ldots, \lfloor\sqrt{n}\rfloor - 1$.
- For $i = \lfloor\sqrt{n}\rfloor$, the if condition on line 5 evaluates to False for $j = 0, 1, \ldots, i-2$ (because then `lst[j]` $= 0$) and the if condition evaluates to True for $j = i-1$ (when `lst[i]` $= 1$

and `lst[j]` $= 1$). So Loop 2 also performs i iterations for $i = \lfloor\sqrt{n}\rfloor$, each one taking time 1. Then the entire function returns: Loop 1 performs no more iteration.

- The total number of steps taken by the algorithm is therefore equal to:

$$\sum_{i=0}^{\lfloor\sqrt{n}\rfloor} i = \frac{\lfloor\sqrt{n}\rfloor\left(\lfloor\sqrt{n}\rfloor + 1\right)}{2} \in \Theta(n) \qquad\qquad \text{as desired.}$$

3. [10 marks] Worst-case and Best-case analysis.

Consider the following function:

```python
def loopy(lst: list) -> None:
    """Precondition: lst is a non-empty list of integers."""
    n = len(lst)
    for i in range(n-1):                        # Loop 1
        if lst[i] % 2 == 0:
            d = lst[i+1] - lst[i]
            for j in range(i+1, n):             # Loop 2
                for k in range(i, j):           # Loop 3
                    if lst[j] - lst[k] < d:
                        d = lst[j] - lst[k]
            for j in range(i+1, n):             # Loop 4
                lst[j] = lst[j] + d
        else:
            j = i + 1
            while j < n and lst[j] > 0:         # Loop 5
                lst[j] = lst[j] + 1
                j = j + 1
```

(a) [**5 marks**] Find, with proof, an asymptotic tight bound (Theta) on the worst-case running time of `loopy`. Your analysis should consist of two separate proofs for matching upper and lower bounds on the worst-case running time.

Solution

Upper Bound: Let $n \in \mathbb{Z}^+$ and let `lst` contain n arbitrary integers.

- Loop 1 performs $n-1 \leq n$ iterations.
- For each iteration of Loop 1, Loop 2 performs $n-i-1 \leq n$ iterations.
- For each iteration of Loop 2, Loop 3 performs $j-i \leq n$ iterations.
- Each iteration of Loop 3 takes time 1, so Loop 3 takes total time $\leq n$.
- Each iteration of Loop 2 takes time $\leq n$ so Loop 2 takes total time $\leq n^2$.
- Loop 4 performs $n-i-1 \leq n$ iterations, each one taking time 1, so Loop 4 takes total time $\leq n$.
- Loop 5 performs at most $n-i-1 \leq n$ iterations, each taking time 1, so Loop 5 takes total time $\leq n$.
- So each iteration of Loop 1 takes time $\leq \max(n^2+n, n) \leq 2n^2$.
- So the total time for `loopy(lst)` is $\leq 2n^3$, i.e., the worst-case time for `loopy` is in $\mathcal{O}(n^3)$.

Lower Bound: Let $n \in \mathbb{Z}^+$. Let `lst` $= [0, 0, \ldots, 0]$, with n copies of the integer 0.

- *Lemma*: Every element of `lst` remains equal to 0 during the entire execution of `loopy`.
  Proof: During the first iteration of Loop 1, `lst[0]` $\% 2 == 0$ evaluates to True, so the algorithm executes the if block. During Loops 2 and 3, because d is always set to the difference between two list elements, and all list elements are equal to 0, $d = 0$ is always true. So during Loop 4, none of the list elements change ($d = 0$ is added to each one).

Since this happens for every iteration, the elements remain the same during the entire execution of `loopy`.

- For each iteration of Loop 1, `lst[i]` is equal to 0 by the Lemma above, so the if block executes.
- Loop 1 performs $n-1$ iterations. There are $\lfloor n/3 \rfloor$ of these iterations for which $i \leq \lfloor n/3 \rfloor - 1$ (when $i = 0, 1, \ldots, \lfloor n/3 \rfloor - 1$).
- For each of these values of i, Loop 2 performs $n - i - 1$ iterations. There are at least $\lfloor n/3 \rfloor$ of these iterations for which $j \geq \lfloor 2n/3 \rfloor$ (when $j = \lfloor 2n/3 \rfloor, \lfloor 2n/3 \rfloor + 1, \ldots, n-1$).
- For each of these values of j and i, Loop 3 performs $j - i \geq \lfloor n/3 \rfloor$ iterations.
- All together, this means Loops 1, 2, 3 perform more than $\lfloor n/3 \rfloor^3$ steps (at least $\lfloor n/3 \rfloor$ steps, at least $\lfloor n/3 \rfloor$ times, at least $\lfloor n/3 \rfloor$ times).
- So the running time of `loopy` is $\geq \lfloor n/3 \rfloor^3$ for each input in our input family. This means that the worst-case running time of `loopy` is in $\Omega(n^3)$.

All together, this proves that the worst-case running time of `loopy` is in $\Theta(n^3)$.

(b) **[5 marks]** In general, we define the **best-case running time** of an algorithm `func` as follows (where $\mathcal{I}_n$ is the set of all inputs of size n):

$$BC_{\texttt{func}}(n) = \min\{\text{running time of executing } \texttt{func(x)} \mid x \in \mathcal{I}_n\}$$

Note that this is analogous to the definition of worst-case running time, except we use **min** instead of max.

Analyse the **best-case** running time of `loopy` to find a Theta bound for it. Your analysis should consist of two separate proofs for matching upper and lower bounds on the best-case running time.

HINT: You should review the definitions of what it means for a function $f : \mathbb{N} \to \mathbb{R}^{\geq 0}$ to be an upper bound or lower bound on the worst-case running time of an algorithm, and take the time to write down the corresponding definitions for the best-case running time, to ensure that you know exactly what you are trying to prove. Your definitions are likely to be very similar to the ones for the worst-case, but *they should NOT be identical* (else you are doing it wrong)!

SUB-HINT: You may find it helpful to first translate the simpler statements "M is an upper bound on the minimum of set S" and "M is a lower bound on the minimum of set S" to make sure you have the right idea. Compare this with what it means for a value to be an upper or lower bound on the *maximum* of a set. Finally, coming back to algorithms, remember that upper and lower bounds can be proved separately on both the worst-case and the best-case running times (since these are two different functions).

Solution

Upper Bound: Let $n \in \mathbb{Z}^+$. Let `lst` $= [-1, -1, \ldots, -1]$, with n copies of the integer -1.

- *Lemma*: Every element of `lst` remains equal to -1 during the entire execution of `loopy`.
  Proof: During the first iteration of Loop 1, `lst[0]` $= -1$ so the else block executes. Because `lst[1]` $= -1$, Loop 5 stops immediately (it performs NO iteration), so the values of `lst` remain unchanged for the next iteration. Since this happens at every iteration, none of the entries of `lst` change.
- For every iteration of Loop 1, the else block will be executed because `lst[i]` $= -1$ so the if condition is False.
- Also because `lst[i]` $= -1$, the condition of Loop 5 is False the first time it is evaluated, so Loop 5 performs NO iteration. This means lines 14–17 take constant time to execute.

- In addition to line 3, and lines 4–5 that execute for each iteration of Loop 1, this means the total number of steps executed is at most $1+(n-1)=n$ for each input in our input family.
- Hence, the best-case running time of `loopy` is in $\mathcal{O}(n)$.

Lower Bound: Let $n \in \mathbb{Z}^+$ and let `lst` be any list of integers of length n.

- Loop 1 executes $n-1$ iterations and each one takes at least constant time.
- So `loopy` executes at least $n-1$ steps for every input.
- This shows that the best-case running time of `loopy` is in $\Omega(n)$.

Hence, the best-case running time of `loopy` is in $\Theta(n)$.