1. **[3 marks] Various topics.**

   (a) **[1 mark] Number representations.** Put an "X" in the box next to the base 2 representation of the decimal number 10.
   (No justification is required.)
   Hint: $2^0 = 1$, $2^1 = 2$, $2^2 = 4$, $2^3 = 8$, $2^4 = 16$ and $2^5 = 32$.

   ☐ 110 ☐ 1010 ☐ 1001 ☐ 10010

   Solution
   ☐ 110 ☒ 1010 ☐ 1001 ☐ 10010

   (b) **[1 mark] Asymptotic descriptions.** Consider the function $f(n) = 5n^2 - 6n + 7$, where $n \in \mathbb{N}$. Put an "X" in **every** box that is beside a statement that is True. (No justification is required.)

   ☐ $f(n) \in \mathcal{O}(n)$ ☐ $f(n) \in \Omega(n^2)$
   ☐ $f(n) \in \Omega(1)$ ☐ $f(n) \in \mathcal{O}(n^3)$

   Solution
   ☐ $f(n) \in \mathcal{O}(n)$ ☒ $f(n) \in \Omega(1)$ ☒ $f(n) \in \Omega(n^2)$ ☒ $f(n) \in \mathcal{O}(n^3)$

   (c) **[1 mark] Choosing the base case.** Consider the statement:

   $$\forall n \in \mathbb{N}, n \geq n_0 \Rightarrow (2n+1) \leq 2^n$$

   Put an "X" in the box next to the **smallest** base case number n_0 for which the statement is True.
   (No justification is required.)

   ☐ 0 ☐ 1 ☐ 2 ☐ 3

   Solution
   ☐ 0 ☐ 1 ☐ 2 ☒ 3

2. **[5 marks] Induction.** Prove the following statement using induction on n:

$$\forall n \in \mathbb{N},\ \exists k \in \mathbb{N},\ 5k = 6^n + 4$$

Solution

Proof. Let $n \in \mathbb{N}$, and let $P(n)$ be the predicate "$\exists k \in \mathbb{N},\ 5k = 6^n + 4$."

Base case: Let $n = 0$. We want to prove $P(0)$. That is, we want to prove $\exists k \in \mathbb{N},\ 5k = 6^0 + 4$.

We can calculate:

$$\begin{aligned} 6^0 + 4 &= 5 \\ &= 1 \cdot 5 \end{aligned}$$

Let $k_0 = 1$. Then $k_0 \cdot 5 = 6^0 + 4$, and $P(0)$, as required.

Induction step: Let $m \in \mathbb{N}$, and assume that $P(m)$. That is, assume that $\exists k \in \mathbb{N},\ 5k = 6^m + 4$. We want to prove $P(m+1)$. That is, we want to prove that $\exists k \in \mathbb{N},\ 5k = 6^{m+1} + 4$.

Let $k_m \in \mathbb{N}$ be such that $5k_m = 6^m + 4$. Since $6^m + 4 > 0$ we know that $k_m \geq 1$. Multiplying both sides by 6 gives $30k_m = 6^{m+1} + 24$. We can then deduce:

$$\begin{aligned} 30k_m &= 6^{m+1} + 24 \\ 30k_m &= 6^{m+1} + 4 + 20 \\ 30k_m - 20 &= 6^{m+1} + 4 \\ 5(6k_m - 4) &= 6^{m+1} + 4 \end{aligned}$$

Letting $k_{m+1} = 6k_m - 4$, we have $5 \cdot k_{m+1} = 6^{m+1} + 4$. (Since $k_m \geq 1$, $6k_m - 4 \geq 0$, and so $k_{m+1} \in \mathbb{N}$.) And so, $\exists k \in \mathbb{N},\ 5k = 6^{m+1} + 4$, as required.

Alternatively, we can calculate:

$$\begin{aligned} 6^{m+1} + 4 &= 6(6^m) + 4 \\ &= 6(5k_m - 4) + 4 \\ &= 5 \cdot 6k_m - 24 + 4 \\ &= 5 \cdot 6k_m - 20 \\ &= 5(6k_m - 4) \end{aligned}$$

and the same conclusion follows. □

3. [**6 marks**] **Worst-case running time.** Consider the following algorithm, which takes as input a list of positive numbers.

```
def alg(A):
    m = len(A)
    count = 0
    i = 1
    j = m
    while i < 2^m and j > 1:   # 2^m means "2 to the power of m."
        if count < i:
            count = count + A[i]
            i = 2 * i
        else:
            j = j // 2         # Integer division; rounds down.
            print('Count exceeded limit')
```

Prove that the **worst-case running time** of the above algorithm is $\Theta(m)$, where m is the length of the input list. Note that this requires two proofs: that the worst-case runtime is $\mathcal{O}(m)$, and that the worst-case runtime is $\Omega(m)$. Be sure to state exactly what you're proving in each part of your solution.

Guidelines: you can assume that any line or block of code within the loop takes constant time. To save time, you do not need to use ceilings and floor function to round your expressions.

Note: If you run out of space on this page, continue your solution on the next page.

Solution

Part 1: Proving that the worst-case runtime is $\mathcal{O}(m)$.

The initialization lines before the `while` loop take 1 step (i.e., have runtime independent of the input size). At each iteration of the `while` loop, two things could happen: either `i` increases by a factor of 2, or `j` decreases by a factor of 2.

We claim that `i` can increase *at most* m times: since `i` starts at 1, after k doublings its value is 2^k, and the loop stops when $k = m$.

We also claim that `j` can decrease *at most* $\log m$ times:[*] since `j` starts at m, after k decreases its value is $\dfrac{m}{2^k}$, and the loop stops when $2^k = m$, or $k = \log m$.

Therefore the total number of loop iterations is at most $m + \log m$ (since during one iteration either `i` has to increase or `j` has to decrease). Each iteration takes constant time, so the cost for the `while` loop is *at most* $m + \log m$ steps.

So our total cost is *at most* $1 + m + \log m$ steps, which is $\mathcal{O}(m)$.

Part 2: Proving that the worst-case runtime is $\Omega(m)$.

To see that the worst-case runtime is also $\Omega(m)$, let $m \in \mathbb{N}$ and consider the input A of length m that consists of all 1's.

Then every time through the `while` loop, `count` increases by 1 and `i` doubles in value, leaving `count < i`.[†] In this case, the `if` branch will always execute, and `i` will double exactly m times, from 1 to 2^m.

Therefore, the loop will execute m times (with each time taking a single step), and thus the runtime is $\Omega(m)$.

---

*We're ignoring floors and ceilings here!

†Formally, we're using the fact that $\forall n \in \mathbb{N},\ n < 2^n$. We didn't expect you to prove this during the test, and didn't penalize it, either.

4. **[6 marks] Best-case running time.** Prove that the **best-case running time** of the algorithm from Question 3 is $\Theta(\log m)$, where m is the length of the input list. Note that this again requires two proofs, similar to the previous question. The same guidelines from that question apply here as well.

Solution

Part 1: Proving that the best-case runtime is $\Omega(\log m)$.

The initialization lines before the `while` loop take 1 step (i.e., have runtime independent of the input size). At each iteration of the `while` loop, two things could happen: either `i` increases by a factor of 2, or `j` decreases by a factor of 2.

Similar to the previous question, `i` must increase m times before $i \geq 2^m$ is true, and `j` must decrease $\log m$ times before $j \leq 1$ is true. So then *at least* $\log m$ loop iterations must occur,* and each iteration takes 1 step.

This gives us a lower bound on the number of steps as $\log m + 1 \in \Omega(\log m)$.

Part 2: Proving that the best-case runtime is $\mathcal{O}(\log m)$.

To see that the best-case runtime is also $\mathcal{O}(\log m)$, let $m \in \mathbb{N}$ and consider the input A of length m where every element is 3.

The initialization lines before the `while` loop take 1 step (i.e., have runtime independent of the input size). Then in the first loop iteration, `count` is 0 and so less than `i`, and the `if` branch executes. After this, `count` is 3 and `i` is 2, and so the condition `count < i` is false, and `else` branch will execute.

Since the `else` branch doesn't change `count` or `i`, it continues to execute for all remaining loop iterations, until `j` reaches 1; as we previously discussed, this takes $\log m$ iterations.

So the number of iterations of the loop on this input A is $1 + \log m$, and since each iteration takes 1 step, the cost of the loop is $1 + \log m$ steps. So the total cost is $1 + (1 + \log m) \in \mathcal{O}(\log m)$.

*Note that considering both `i` and `j` is necessary here: even if we know how long `j` takes to reach its bound, that doesn't immediately give us a lower bound on the number of iterations since `i` might reach its bound faster!