Problem 1. (20 points) Prove or disprove the following conjectures:

(a) $n^{\log c} = O(c^{\log n})$.
(b) $2^{n+1} = O(2^n)$.
(c) $2^{2^n} = O(2^n)$.
(d) $T(n) = T(n-1) + n = O(n^2)$.
(e) $T(n) = 16T(n/4) + n^2 = O(n^2)$.

Solution:

(a) $n^{\log c} = O(c^{\log n})$ is true, as:

$$n^{\log c} = (2^{\log n})^{\log c} = 2^{\log n \cdot \log c} = (2^{\log c})^{\log n} = c^{\log n}.$$

(b) $2^{n+1} = O(2^n)$ is true, as:

$$2^{n+1} = 2 \cdot 2^n = O(2^n)$$

is satisfied for any $c \geq 2$.

(c) $2^{2^n} = O(2^n)$ is false, as:

$$2^{2^n} = 2^n \cdot 2^n = O(2^n)$$

requires $c \geq 2^n$.

(d) $T(n) = T(n-1) + n = O(n^2)$ is true, as for the guess $T(k) \leq ck^2$, $0 < k < n$, we have:

$$T(n) = T(n-1) + n$$
$$\leq c(n-1)^2 + n$$
$$= c(n^2 - 2n + 1) + n$$
$$= cn^2 - (2cn - c - n)$$
$$\leq cn^2$$

that holds when:

$$2cn - c - n \geq 0$$
$$c(2n - 1) \geq n$$
$$c \geq \frac{n}{2n - 1}$$
$$c \geq 1 \text{ (as } n \geq 1).$$
(e) $T(n) = 16T(n/4) + n^2 = O(n^2)$ is false, as $n^{\log_b a} = n^2 = \Theta(f(n))$ and by the master method $T(n) = \Theta(n^2 \log n)$.

**Problem 2.** (20 points) Integer division $n/d$ produces quotient $q$ and remainder $r$, and can be implemented using repeated subtraction of $d$ from $n$. Give pseudocode for $\text{DIVIDE}(n, d)$ that uses this method to compute $q$ and $r$. Prove your algorithm correct using a loop invariant.

**Solution:** For $n \geq 0$ and $d > 0$:

```
DIVIDE(n, d)
1  q = 0
2  r = n
3  while r ≥ d
4      q = q + 1
5      r = r - d
6  return q, r
```

We prove the algorithm correct using the following loop invariant: before any iteration of the `while` loop, $q \cdot d + r = n$.

*Initialization.* Before the first iteration, $q = 0$, $r = n$, so $0 \cdot d + n = n$ and the invariant holds.

*Maintenance.* The new values are $q' = q + 1$ and $r' = r - d$, and the invariant reads

$$q' \cdot d + r' = (q + 1) \cdot d + (r - d) = q \cdot d + r.$$

We know that the invariant was true for previous values $q$ and $r$ (that is, $q \cdot d + r = n$), so $q' \cdot d + r' = n$ and the invariant holds.

*Termination.* At termination, $q \cdot d + r = n$ by the loop invariant. Under these circumstances, for $q$ and $r$ to not be the quotient and remainder of $n/d$ respectively, one of two conditions must be true:

(a) $r \geq d$, so we can have $q$ smaller than the actual quotient. However, the loop terminates only when $r < d$.

(b) $r < 0$, so we can have $q$ greater than the actual quotient. However, in the one before last iteration $r \geq d$ and in the last iteration $r$ becomes $r - d$, so $r \geq 0$.

Thus, $q$ and $r$ are the quotient and remainder of $n/d$ respectively, and the algorithm is correct.

**Problem 3.** (20 points) Suppose that you want to output 0 or 1, each with probability $1/2$. At your disposal is a procedure $\text{BIASED-RANDOM}$ that outputs 0 or 1 with probability $p$ and $1 - p$ respectively ($0 < p < 1$), but you don't know what $p$ is. Give an algorithm that uses $\text{BIASED-RANDOM}$ as a subroutine and returns an unbiased answer. State the expected running time of your algorithm as a function of $p$.

**Solution:** We can exploit the fact that $p(1-p) = (1-p)p$ and write:
Unbiased-Random

1 while TRUE
2     x = Biased-Random
3     y = Biased-Random
4     if x ≠ y
5         return x

Viewing each iteration as a Bernoulli trial, where success means that the iteration returns a value, the probability of success is the probability that 0 is returned plus the probability that 1 is returned, or \( p(1 - p) + (1 - p)p = 2p(1 - p) \).

The number of trials until a success occurs is given by the geometric distribution \( \Pr\{X = k\} = q^{k-1}p \), where \( q = 1 - p \) is the probability of failure (so the formula captures the probability of \( k - 1 \) failures before the one success). We can find its expectation as:

\[
E[X] = \sum_{k=1}^{\infty} kq^{k-1}p \\
= \frac{p}{q} \sum_{k=0}^{\infty} kq^k \\
= \frac{p}{q} \cdot \frac{q}{(1 - q)^2} \\
= \frac{p}{q} \cdot \frac{q}{q} \cdot \frac{1}{p} \\
= \frac{1}{p}.
\]

In our case, \( E[X] = \frac{1}{2p(1-p)} \) and the expected running time of Unbiased-Random is \( \Theta\left(\frac{1}{2p(1-p)}\right) \).

Problem 4. (20 points) Use heaps to design an \( O(n \lg k) \) algorithm to merge \( k \) sorted lists into one sorted list, where \( n \) is the total number of elements in all input lists. (You don’t need to implement basic heap operations.)

Solution:

We can use a min-heap to repeatedly select next smallest element among \( k \) candidates. Once we know that the smallest element \( x \) comes from list \( i \), we add \( x \) to the output array and add the next element from list \( i \) to the heap.
MERGE\((L, n, k)\)
1 let \(M[1..n]\) be a new array
2 let \(H[1..k]\) be a new min-heap
3 for \(i = 1\) to \(k\)
4 \(H\).insert\((L[i].\text{head})\)
5 for \(i = 1\) to \(n\)
6 \(\text{min} = H\).extract-min()
7 \(M[i] = \text{min}\)
8 if \(\text{min}.\text{next} \neq \text{NULL}\)
9 \(H\).insert\((\text{min}.\text{next})\)
10 return \(M\)

Heap \(H\) here has at most \(k\) elements, so both \text{EXTRACT-MIN} and \text{INSERT} take \(O(\lg k)\). The enclosing loop executes exactly \(n\) times, so the overall running time of the algorithm is \(O(n \lg k)\).

**Problem 5.** (20 points) Describe an algorithm that, given \(n\) integers in the range 0 to \(k\), preprocesses its input in \(\Theta(n + k)\) time and then answers any query about how many of the \(n\) integers fall into a range \([a..b]\) in \(O(1)\) time. Be mindful of edge cases. (Recall how counting sort computes the number of elements smaller than \(x\) for every input element \(x\).)

**Solution:** We can reuse the idea of counting sort that computes \(C[i]\) to be the number of input elements smaller than or equal to \(i\). Answering queries about the integers falling into a range \([a..b]\) is as easy as computing \(C[b] - C[a - 1]\) with \(C[-1] = 0\):

PREPROCESS\((A, n, k)\)
1 let \(C[-1..k]\) be a new array
2 for \(i = -1\) to \(k\)
3 \(C[i] = 0\)
4 for \(j = 1\) to \(n\)
5 \(C[A[j]] = C[A[j]] + 1\)
6 for \(i = 1\) to \(k\)
7 \(C[i] = C[i - 1]\)
8 return \(C\)

QUERY\((C, a, b)\)
1 return \(C[b] - C[a - 1]\)

**Problem 6.** (20 points) Recall the quicksort algorithm:

QUICKSORT\((A, p, r)\)
1 if \(p < r\)
2 \(q = \text{PARTITION}(A, p, r)\)
3 QUICKSORT\((A, p, q - 1)\)
4 QUICKSORT\((A, q + 1, r)\)
**Partition**($A, p, r$)

1. $x = A[r]$
2. $i = p - 1$
3. **for** $j = p$ **to** $r - 1$
4. 
   - **if** $A[j] \leq x$
   - $i = i + 1$
5. exchange $A[i]$ with $A[j]$
6. exchange $A[i + 1]$ with $A[r]$
7. **return** $i + 1$

And answer the following questions:

(a) What is the running time of **Partition**?

Each line of **Partition** takes constant time, so the overall running time is linear in the number of iterations, or $\Theta(r - p)$.

(b) What is the running time of **QuickSort** when all elements of $A$ have the same value?

With all elements having the same value, condition on line 4 of **Partition** always evaluates to true, and $i$ gets incremented $(r - p)$ times to become $(r - 1)$ at the end of the procedure. Thus, **Partition** always returns $i + 1 = r$, producing a split $(n - 1) : 0$ and resulting in a recurrence $T(n) = T(n - 1) + T(0) + \Theta(n)$ with a solution $T(n) = \Theta(n^2)$.

(c) What is the running time of **QuickSort** when $A$ contains distinct elements sorted in decreasing order?

- On the first partition, the pivot is the smallest element in $A[p..r]$ and the resulting split is $0 : (n - 1)$. Note that partition ends with exchanging first and last elements of $A[p..r]$, which are the largest and smallest elements respectively.
- On the second partition, the pivot is the largest element in $A[p..r]$ and the split becomes $(n - 1) : 0$.
- On the third partition, we are back to $A[p..r]$ containing elements in decreasing order, the split becomes $0 : (n - 1)$, and the pattern recurs.

This pattern results in a recurrence $T(n) = T(n-1) + T(0) + \Theta(n)$ and gives $T(n) = \Theta(n^2)$. 
