Question 1 (Computational Complexity)
Please prove the following.

(a) Consider the factorial function, \( N! = \prod_{n=1}^{N} n \). Show that \( N! = O(N^N) \).

**Solution:** We will show that \( N! \leq 1 \times N^N \) for \( N \geq 1 \), which establishes that \( N! = O(N^N) \).

To show that \( N! \leq 1 \times N^N = N^N \) for \( N \geq 1 \), note that
\[
\frac{N!}{N^N} = \frac{1 \cdot 2 \cdots N}{N \cdot N \cdots N} = \frac{1}{N} \frac{2}{N} \cdots \frac{N}{N} \leq 1.
\]

(b) Let \( p(N) = \sum_{i=0}^{d} a_i N^i \), be a degree-\( d \) polynomial in \( N \), where \( a_d > 0 \). Show that \( p(N) = \Theta(N^d) \). (Hint: \( d \) is constant, and \( \Theta(N^d) \) implies that we are considering the rate of growth of \( p(N) \) for large \( N \).)

**Solution:** We need to show that \( c_1 N^d \leq p(N) \leq c_2 N^d \) for \( N \geq N_0 \), where \( c_1, c_2, N_0 \) are constants. To do so, we compute the following limit,
\[
\lim_{N \to \infty} \frac{p(N)}{N^d} = \lim_{N \to \infty} \frac{\sum_{i=0}^{d} a_i N^i}{N^d} = \lim_{N \to \infty} \sum_{i=0}^{d} a_i N^{i-d} = \sum_{i=0}^{d} a_i \lim_{N \to \infty} N^{i-d} = a_d,
\]

where the limit of a finite summation is equal to the sum of individual limits, and besides \( i = d \) those limits are all zero. Because the limit exists and is finite, we can choose \( c_1 = a_d - \epsilon \), \( c_2 = a_d + \epsilon \), and there exists some large \( N_0 \) for which \( c_1 \leq \frac{p(N)}{N^d} \leq c_2 \) holds for all \( N \geq N_0 \) (note that we are employing the definition of a limit).
Question 2 (Algorithms)
Consider the problem of determining whether a sequence of $N$ numbers contains $N$ distinct numbers, or instead at least one number occurs multiple times. Describe an efficient algorithm and its computational complexity. (There is no need to use pseudocode; describing in words is fine if your algorithm is simple.)

Solution: A simple solution is to sort the $N$ numbers in $O(N \log(N))$ time, and then scan the list in $O(N)$ while checking whether any numbers repeat.
**Question 3** (Data structures)

We have discussed that *linked lists* are a popular data structure that lets objects be inserted and removed from the data structure in a flexible order, in contrast to stacks and queues. On the other hand, linked lists require $O(N)$ computation for searching among $N$ objects for one with a specific key, because we may need to scan through the entire list.

**Skip lists:** We can perform faster searches through the data structure using skip lists. As illustrated above, we supplement the regular linked list (normal lane) with a skip list (express lane) that lets us skip through objects. Both lanes are maintained in sorted order, allowing us to skip through the express lane and quickly identify the approximate location to scan within the normal lane. In the illustration, both lanes are singly linked lists, where objects in the skip list also include pointers to corresponding objects in the regular linked list. Each next pointer in the express lane moves us ahead (skips) $K$ objects in the normal lane. (Although $K = 5$ in our illustration, $K$ might not be constant in general, because objects are being added and removed. To simplify the question, we assume that $K$ is constant.)

**Example search:** Suppose that we are searching for the object with key 45 (located in the regular linked list). We begin with the skip list at the object whose key is 10. Because $10 < 45$, we skip ahead to 30 in the skip list. Because $30 < 45$, we skip ahead to 57 in the skip list. However, $57 > 45$; we return to 30 in the skip list, and then move below to the 30 in the regular linked list. Because $30 < 45$, we move to 43 in the regular linked list. Because $43 < 45$, we move to 45 in the regular linked list. Our search is over.

**Search computation:** How much computation does searching with a skip list require? The number of objects in the express lane is $\Theta(N/K)$. Once we identify the approximate location to scan within the normal lane, that scan is $O(K)$. Therefore, the total computation for search is $O(N/K + K)$.

(a) To minimize total computation, $K$ can be a function of $N$. Compute the optimal $K^*$ that minimizes $N/K + K$. Using this $K^*$, what is the total computational complexity? (Hint: you may assume that $N$ is fixed.)

**Solution:** We want $K^*$ that minimizes $f(K) = N/K + K$. To do so, we compute the derivative of $f(K)$ with respect to $K$ and set it to zero, $-\frac{N}{K^2} + 1 = 0$. Therefore, $K^* = \sqrt{N}$. The total computational complexity is $O(N/K^* + K^*) = O(N/\sqrt{N} + \sqrt{N}) = O(2\sqrt{N}) = O(\sqrt{N})$.

(b) What about using two levels of skip lists? You can envision a normal lane with $N$ objects, an express lane that skips $K_e$ normal objects at a time (there are $\Theta(N/K_e)$ express lane objects), and a super lane that skips $K_s$ express objects (there are $\Theta(\frac{N}{K_eK_s})$ super lane objects). Recall that the total computation with an express lane (and without a super lane) was $O(N/K + K)$; what is the total computation with a super lane? Express your answer as a function of $K_e, K_s, N$, and make sure to justify your answer.

**Solution:** We will scan through up to $\Theta(\frac{N}{K_eK_s})$ super lane objects, $O(K_s)$ express lane objects, and $O(K_e)$ normal lane objects. The total computation is $O(\frac{N}{K_eK_s} + K_e + K_s)$. 

---