Question 1 (K&T 2.2)**.** Suppose you have algorithms with the six running times listed below. (Assume these are the *exact* number of operations performed as a function of the input size n.) Suppose you have a computer that can perform 10^{10} operations per second, and you need to compute a result in at most an hour of computation. For each of the algorithms, what is the largest input size n for which you would be able to get the result within an hour?

1. n^2
2. n^3
3. $100n^2$
4. $n \log_2 n$
5. 2^n
6. 2^{2^n}

Question 2 (K&T 2.3)**.** Take the following list of functions and arrange them in ascending order of growth rate. That is, if function $g(n)$ immediately follows function $f(n)$ in your list, then it should be the case that $f(n)$ is $O(g(n))$. Please prove your claims.

1. $f_1(n) = n^{2.5}$
2. $f_2(n) = \sqrt{2n}$
3. $f_3(n) = n + 10$
4. $f_4(n) = 10^n$
5. $f_5(n) = 100^n$
6. $f_6(n) = n^2 \log n$

Question 3. For each of the following, answer with the tightest upper bound from this list : $O(1)$, $O(\log n)$, $O(n)$, $O(n \log n)$, $O(n^2)$, $O(n^3)$, $O(2^n)$, $O(n!)$. Prove/justify your answers.

- Total number of nodes in a balanced binary tree with n leaves.
- Number of edges in a graph with n nodes, where every node is connected to every other node.
- Time to find a given element in a sorted array.
- Number of three-element subsets of a set of size n.
- Number of ways to seat n people around a circular table.
- Time to find a given element in a linked list.
- Biggest number that can be represented with n bits.
- Best-case run time to sort n items using merge sort.
- Time to sort a list of n items if you are only allowed to swap adjacent elements.

Question 4 (K&T 2.8)**.** You're doing some stress-testing on various models of glass jars to determine the height from which they can be dropped and still not break. The setup for this experiment, on a particular type of jar, is as follows. You have a ladder with n rungs, and you want to find the highest rung from which you can drop a copy of the jar and not have it break. We call this the *highest safe rung*.

It might be natural to try binary search: drop a jar from the middle rung, see if it breaks, and then recursively try from rung $n/4$ or $3n/4$ depending on the outcome. But this has the drawback that you could break a lot of jars in finding the answer.

If your primary goal were to conserve jars, on the other hand, you could try the following strategy. Start by dropping a jar from the first rung, then the second rung, and so forth, climbing one higher each time until the jar breaks. In this way, you only need a single jar—at the moment it breaks, you have the correct answer—but you may have to drop it n times (rather than $\log n$ as in the binary search solution).

So here is the trade-off: it seems you can perform fewer drops if you're willing to break more jars. To understand better how this trade-off works at a quantitative level, let's consider how to run this experiment given a fixed "budget" of $k \geq 1$ jars. In other words, you have to determine the correct answer—the highest safe rung—and can use at most k jars in doing so.

(a) Suppose you are given a budget of $k = 2$ jars. Describe a strategy for finding the highest safe rung that requires you to drop a jar at most $f(n)$ times, for some function $f(n)$ that grows slower than linearly. (In other words, it should be the case that $\lim_{n\to\infty} f(n)/n = 0$.)

(b) Now suppose you have a budget of $k > 2$ jars, for some given k. Describe a strategy for finding the highest safe rung using at most k jars. If $f_k(n)$ denotes the number of times you need to drop a jar according to your strategy, then the functions $f_1, f_2, f_3, \ldots$ should have the property that each grows asymptotically slower than the previous one: $\lim_{n\to\infty} f_k(n)/f_{k-1}(n) = 0$ for each k.

Question 5 (Subinterval sums, K&T 2.6)**.** Consider the following basic problem. You're given an array A consisting of n integers $A[1], A[2], \ldots, A[n]$. You'd like to output a two-dimensional $n \times n$ array B in which $B[i, j]$ (for $i \leq j$) contains the sum of array entries $A[i]$ through $A[j]$—that is, the subinterval sum $A[i] + A[i+1] + \cdots + A[j]$. (The value of array entry $B[i, j]$ is left unspecified whenever $i > j$, so it doesn't matter what is output for these values.)

For example, given the array $A = [1, 5, 7, 2]$, the desired output B would be

$$\begin{bmatrix} 1 & 6 & 13 & 15 \\ & 5 & 12 & 14 \\ & & 7 & 9 \\ & & & 2 \end{bmatrix}.$$

(Note this is formulated slightly differently than K&T 2.6, which only requires $B[i, j]$ to be defined when $i < j$, but that is silly. The sum $A[i] + A[i+1] + \cdots + A[j]$ is perfectly well-defined when $i = j$; it is equal to $A[i]$. (For that matter, the sum is perfectly well-defined when $i > j$ too (it is 0), but let's ignore that.))

Here's a simple algorithm to solve this problem.

```
1: for i ← 1 ... n do
2:   for j ← i ... n do
3:     Add up array entries A[i] through A[j]
4:     Store the result in B[i, j]
5:   end for
6: end for
```

Note that j starts at i, not at 1.

(a) For some function f that you should choose, give a bound of the form $O(f(n))$ on the running time of this algorithm on an input of size n (i.e., a bound on the number of operations performed by the algorithm).

(b) For this same function f, show that the running time of the algorithm on an input of size n is also $\Omega(f(n))$. (This shows an asymptotically tight bound of $\Theta(f(n))$ on the running time.)

(c) Although the algorithm you analyzed in parts (a) and (b) is the most natural way to solve the problem—after all, it just iterates through the relevant entries of the array B, filling in a value for each—it contains some highly unnecessary sources of inefficiency. Give a different algorithm to solve this problem, with an asymptotically better running time. In other words, you should design an algorithm with running time $O(g(n))$, where $\lim_{n\to\infty} g(n)/f(n) = 0$.

Question 6. On a scale of 1 to 10, how difficult was this assignment? How many hours would you estimate that you spent on it? (Note that these are two separate questions, though of course they are probably correlated. It is quite possible to have a difficult but short or easy but long assignment.)