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}

In addition to O, Θ, and Ω, we will also sometimes use the notation $o(g(n))$ (little-o) to mean that a function is $O(g(n))$ but *not* $\Theta(g(n))$. So, if $\lim_{n\to\infty} T(n)/g(n) = 0$, then $T(n)$ is $o(g(n))$. Intuitively, if big-O is like "less than or equal to", little-o is like "less than"; if $f(n)$ is $o(g(n))$ then f "grows strictly more slowly than" g.

Question 2 (Some asymptotic properties)**.**

(a) Show that n^j is $o(n^k)$ whenever $j < k$.

(b) Prove that $\log_a n$ is $\Theta(\log_b n)$ for any positive integer bases $a, b > 1$. Conclude that we are justified in writing simply $\Theta(\log n)$ without caring about the base of the logarithm.

Hint: recall, or look up, the change-of-base formula for logarithms.

(c) Prove that n^k is $o(b^n)$ for any positive integer k and any real number $b > 1$. For example, n^{295} is $o(1.0001^n)$. *In the long run*, any polynomial function grows more slowly than any exponential function!

Hint: use L'Hôpital's rule k times. Recall that the derivative of b^n with respect to n is $b^n \ln b$.

Question 3 (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 4. Characterize the (best possible) asymptotic behavior of each of the following in terms of n, using big-Θ. Give a *short* justification for each answer.

Please come ask for help if you are unsure about any of these!

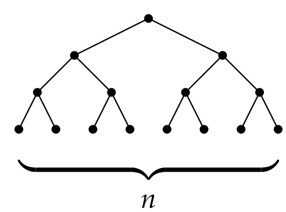
(a) $1 + 2 + 3 + 4 + \cdots + n$

(b) Average number of array lookups needed to find a given element in a sorted array of length n.

(c) Number of two-element subsets of a set of size n.

(d) $1 + 2 + 4 + 8 + \cdots + 2^n$

(e) Total number of nodes in a balanced binary tree with n leaves.



(f) Number of edges in a graph with n nodes, where every node is connected to every other node.

(g) Number of ways to seat n people around a circular table.

(h) Average number of steps needed to find a given element in a linked list of length n.

(i) Biggest integer that can be represented with n bits.

(j) Worst-case number of swaps needed to sort an array of length n if you are only allowed to swap adjacent elements.

(k) Worst-case number of steps needed to check whether two lists, each containing n integers (not necessarily sorted), have any element in common.

4(k)

(l) Number of array lookups performed by this Python code:

```
for i in range(0,n):
  for j in range (0,i):
    sum += array[i][j]
```

(m) Number of addition operations performed by this Java code:

```
for (int i = 0; i < n; i += 3) {
  sum = sum + i;
}
```

(n) Total number of calls to `println` performed by this Java code:

```
for (int i = 1; i < n; i *= 2) {
  for (j = 0; j < 20; j++) {
    System.out.println(i + j);
  }
}
```

(o) Total number of calls to `print` performed by this Python code:

```
def foo(n):
  print n
  if n > 0:
    foo(n-1)
    foo(n-1)
```

Question 5 (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 $\log n$ times 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, $f(n)$ should be $o(n)$, that is, $\lim_{n\to\infty} f(n)/n = 0$.)

(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 6.

(a) On a scale of 1 to 10, how difficult was this assignment?

(b) How many hours would you estimate that you spent on it?

(c) Which question(s) did you find most difficult? Why?