NOTE: The proofs here are (intentionally) presented in full detail. You may submit proofs that are shorter and still get full credit, as long as the idea is clear.

1: Big Oh

(a) $O(n^2)$

(b) $O(\log n)$

(c) $O(1/n)$

(d) $O(1)$

(e) It is well known that $\ln n < 1 + 1/2 + 1/3 + \cdots + 1/n < \ln n + 1$, where $\ln n$ denotes the natural logarithm. Using this, we can conclude that $f(n) = O(\log n)$.[1]

2: Removing Duplicates

Algorithm: : First sort the array A (using MergeSort), and suppose C is the sorted array (also of size n). Now, scan C from left to right, keeping track of the previous element. If the current element being scanned is not equal to the previous element (or if it is the first element), place it in B, else move on.

Correctness: : We observe that every distinct element in C is included in B, and since A and C have the same elements, we have the correctness.

Running time: The sorting step takes $O(n \log n)$ time, and the left-right scan takes time $O(n)$. Thus the overall time is $O(n \log n)$.

3: Square vs Multiply

Suppose we have two n digit numbers a, b that we wish to multiply. Let $A()$ be the algorithm for squaring that takes time $O(n \log n)$.

Now, observe that $ab = \frac{(a+b)^2 - a^2 - b^2}{2}$. Thus to compute ab, we can find $A(a+b) - A(a) - A(b)$, and divide by 2. The running time thus consists of first computing $(a+b)$ (time $O(n)$), three calls to $A()$ (time $O(n \log n)$), plus the time for division ($O(1)$ or $O(n)$, depending on how the division is done). Thus the overall time is $O(n \log n)$.

4: Basic Probability

[1]This part is meant as a test, to see if you know the fact above. The fact itself is not too difficult to prove, though I do not expect you to prove it if you haven't seen it. First, note that $\ln n$, by definition, is $\int_{t=1}^{n} \frac{dt}{t} = \sum_{j=1}^{n-1} \int_{t=j}^{j+1} \frac{dt}{t}$. Now, the term $\int_{t=j}^{j+1} \frac{dt}{t}$ lies between $\frac{1}{j}$ and $\frac{1}{j+1}$. This implies that $\frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} < \ln n < 1 + \frac{1}{2} + \dots \frac{1}{n-1}$, which implies the inequality.

(a) Any outcome of the tosses can be written as a string of length k (e.g. HTTTHHH, ...). There are 2^k total possibilities, all of which are equally likely. The ones that contain precisely one heads are HTTTTT..., THTTTT..., ..., and thus there are k of them. Thus the probability is $k/2^k$.

(b) There are k^k total colorings of the boxes (one way to see this is: a coloring is a string of length k, and each position in the string is one of k colors). All these colorings are equally likely.

Now, the number of colorings in which the boxes all get different colors is exactly $k! = k(k-1)(k-2)\dots 1$ (one way to see this is: we can color the first box with any of the k colors, and having done so, the second box can be colored with any of the remaining $(k-1)$ colors, and so on).

Thus the desired probability is $k!/k^k$. (Interestingly, this turns out to be roughly e^{-k}.)

5: Array Sums

Let us describe an $O(n^2 \log n)$ time algorithm.

Algorithm: : First, sort the elements of A, thus obtaining a new array B. We will use B for 'querying'. Now, for every choice of j, k, compute $A[j] + A[k]$, and then check (using binary search in B) if there exists an element in B equal to $A[j] + A[k]$, and output YES if the search is successful. If the search above fails for all j, k, output NO.

Correctness: : If there exist indices i, j, k with $A[i] = A[j] + A[k]$, then the search for $A[j] + A[k]$ in B must succeed (because $A[i]$ is also present in B). If there are no indices i, j, k, the search can never succeed. Thus the algorithm is always correct.

Running time: : The initial sorting takes $O(n \log n)$ time. Then, we perform n^2 binary searches. Each takes time $O(\log n)$, thus the overall run time is $O(n^2 \log n)$.

[*Bonus Exercise:* Find an algorithm with runtime $O(n^2)$.]