## 1: Big Oh

**(a)** $O(n^2)$

**(b)** $O(\log n)$

**(c)** $O(1/n)$

**(d)** $O(1)$

**(e)** $O(\log n)$

## 2: Removing Duplicates

**Algorithm:** : First, sort the array A (using merge sort), we will call this sorted array C which is of size $n$. Next, iterate through C, in order, while keeping track of the previous element. If the current element is not equal to the previous element (or it's the first element), add it to B. Else, continue to the next element.

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

**Running time:** : The step of sorting A with merge sort take $O(n \log n)$ time and the step of iterating B takes $O(n)$ time. Thus the overall running time of the algorithm is $O(n \log n)$

## 3: Square vs Multiply

Given an algorithm that can square an $n$ digit number in $O(n \log n)$ time (let it be A()), we can find the product of any two $n$ digit numbers $a$,$b$ also in $O(n \log n)$ time.

Observe that $ab = \frac{(a+b)^2 - a^2 - b^2}{2}$. To compute $ab$ can we compute $A(a+b) - A(a) - A(b)$ and divide by 2.

The running time consists of computing $(a+b)$ ($O(n)$ time), three calls to A() ($O(n \log n)$ time) and division by two ($O(1)$). Thus the overall time is $O(n \log n)$.

## 4: Basic Probability

**(a)** When tossing a fair coin $k$ times, there are $2^k$ possible outcomes. A sequence of $k$ coin tosses can be written as a string of length $k$ (e.g. HTTTHHTTTTT...). The possible outcomes are of the form HTTT..., THTT..., TTHT..., ..., and there are $k$. Thus the probability of seeing heads precisely once is $k/2^k$.

**(b)** Given $k$ different boxes and $k$ different colors, there are $k^k$ possible outcomes (similar to rolling a $k$ sided dice $k$ times). All have equal probability.

The number of different ways to paint the boxes so they all have a distinct color is $k!$. When painting the first box we have $k$ colors to choose from, $(k-1)$ for the second box and so on to one color for the final box. $k! = k(k-1)(k-2)....1$.

Thus the probability of all the boxes getting distinct colors is $k!/k^k$

## 5: Array Sums

**Algorithm:** : Let us describe an $O(n^2 \log n)$ time algorithm to determine: if given an array $A[1...n]$ of integers, there exists indices $i, j, k$ such that $A[i] = A[j] + A[k]$.

First, sort the elements of A (using merge sort) to obtain a new array B. Next, for each combination of indices $j, k$ compute $A[j] + A[k]$ and search B (using binary search) to see if it contains this sum. If for any combination of indices $j, k$ there exists an element in B that equals $A[j] + A[k]$, return true. If all combinations of indices are tried and no match is found, return false.

**Correctness:** : Array B contains identical elements as array A. All combinations of indices $j, k$ are tested and we determine if $A[j] + A[k]$ is an element of B. So if there exists indices $i, j, k$ such that $A[i] = A[j] + A[k]$ they will be found. If no combination of indices is found, a valid combination doesn't exist.

**Running time:** : The initial merge sort takes $O(n \log n)$ time. There are $n^2$ binary searches performed on B which each take $O(\log n)$ time. So the running time is $O(n^2 \log n) + O(n \log n)$. Thus the time complexity is $O(n^2 \log n)$.