Goal: Understand properties of randomized algorithms

1) The algorithm at the right takes a biased coin (BiasedCoin returns 1 with probability p and 0 w.p. $1-p$). Argue that von Neumann's algorithm is unbiased, i.e. it outputs 0 and 1 each with probability ½.

```
VonNeumannCoin():
  x ← BiasedCoin()
  y ← BiasedCoin()
  if x ≠ y
      return x
  else
      return VonNeumannCoin()
```

You have 4 possiblities: HH, HT, TH, TT
It returns 1 for HT and 0 for TH cases, which occur with probability p (1-p) = (1-p) p.
So, conditioned on the coins being equal, you get 1 w.p. ½ and 0 w.p. ½
This means it gives a fair coin.

2) Describe an algorithm that solves the following problem: Your input is an integer n. The algorithm should output a random subset of $\{1, \ldots, n\}$ with equal probability. Hint: count the number of subsets and use the combinatorics to design the algorithm.

Flip a coin to determine whether each item is in the set. This gives 2^n possibilities with equal probabilities. In other words, each set from the power set is output with probability $1/2^n$.

```
RandomSet(n):
  S = {}
  for i = 1 to n:
    if FairCoin():
      S = Union(S, {I})
  return S
```

3) Suppose we use QuickSelect to select the minimum element (k=1) from array A=[3, 2, 9, 0, 7, 5, 4, 8, 6, 1].
Describe the sequence of partitions that results in a worst-case performance.

```
QuickSelect(A[1..n], k):
  r ← Partition(A[1..n], Random(n))
  if k < r
      return QuickSelect(A[1..r−1], k)
  else if k > r
      return QuickSelect(A[r+1..n], k−r)
  else
      return A[k]
```

Worst case is to always pick the largest element. In that case, the array only shrinks by 1.

Here, that is:
9, 8, 7, 6, 5, 4, 3, 2, 1, 0

In the “average” case, you get something in the middle of the range and it shrinks exponentially over rounds.

4) Try to argue non-rigorously, but using ideas from probability and analogy to binary search, that QuickSelect runs in $O(n)$ expected time.

See the posted notes.