Please read the instructions carefully:

1. There are two stages to each assignment:

   (a) For the first stage, you need to submit your homework solution as a single pdf file to NYU Classes. It can be either a scanned copy of your written solution or photos joined into a pdf. You are also allowed to typeset the solutions in MS Word/\LaTeX. In case of scanned copies it is your responsibility that the solution is readable. There are no late submissions accepted, as sample solution outlines (to be used for the second stage of the homework) will be given out on the day of the submission of the first stage of the homework. You should leave around 5 to 6 cm of space between each part of the answers for the second stage of the homework.

   (b) For the second stage of the assignment, you are required to critique your own solutions and analyze the mistakes that you made. Based on the outlines provided, you need to write the self criticism in the empty space that you left in your homework. Do not rewrite your solution, you need to understand and point out the mistakes that you made when writing the original solutions. A good analysis of your solution should summarize the key steps, ideas or techniques that you previously missed. It should prompt you to reorganize your thoughts and think about what you can do to improve the solutions. Even in the case where you completely missed the solution it is insightful to point out why the approach adopted by you does not work and identify the step where it fails. This will allow you to get full credit on the self evaluation part of the homeworks and of course help you in understanding the question. Mark correct solutions in green, incorrect solutions in red and if you are unsure about any solutions write it in blue.

2. For each question where you are required to design an algorithm, you need to clearly explain the algorithm and prove the running time complexity as asked in the question.

3. You can assume $O(1)$ complexity of arithmetic of numbers (e.g., multiplication/division of numbers and calculations of $n!$) unless otherwise stated in the question.
Question 1. (10 points) Scrutinizing a test
Let the random variable \( T \) denote the result of the test being \(+ (\text{ve}) / - (\text{ve})\) and the random variable \( C \) denote the status of having cancer as \( t(\text{true}) / f(\text{false})\).
We know that \( \Pr(T = +|C = t) = 0.95, \Pr(C = t) = 0.02, \Pr(T = -|C = f) = 0.9 \). Therefore by Bayes rule
\[
\Pr(C = t|T = +) = \frac{\Pr(T = +|C = t) \Pr(C = t)}{\Pr(T = +|C = t) \Pr(C = t) + \Pr(T = +|C = f) \Pr(C = f)} = \frac{0.95 \times 0.02}{0.95 \times 0.02 + 0.10 \times 0.98} \approx 0.1624
\]

Question 2. (10 points) Fair game
Let the amount won be denoted by a random variable \( W \).
The expected winnings of the game are
\[
E[W] = 5 \cdot \binom{3}{3} \cdot \left(\frac{1}{6}\right)^3 \quad \text{if all 3 are our number}
+ 3 \cdot \binom{3}{2} \cdot \left(\frac{1}{6}\right)^2 \cdot \frac{5}{6} \quad \text{if only 2 show our number}
+ 1 \cdot \binom{3}{1} \cdot \frac{1}{6} \cdot \left(\frac{5}{6}\right)^2 \quad \text{if only 1 shows our number}
- 1 \cdot \binom{3}{0} \cdot \left(\frac{5}{6}\right)^3 \quad \text{if none of them show our number}
= 0
\]
Therefore the average winnings on the game are 0, which makes it a fair game.

Question 3. (15 points) Nuts and bolts
The idea is to emulate quicksort, but the pivot choosing step is now defined by what we pick from the opposite pile.
To split the nuts into two parts
1. Choose a random bolt, \( b_i \).
2. Find its corresponding nut, \( n_i \).
3. Split the bolts into two parts, depending on the size of the nut \( n_i \), and also split the nuts into two parts when they are compared to the bolt \( b_i \).
4. Recursively sort the two piles of nuts/bolts, as we are guaranteed that two piles have the correct matches inside them.
We will show that the analysis of this algorithm is the same as that of quicksort, which will give us the same expected running time.

Let $i$ be the even that the $i$'th bolt is chosen as the pivot. This probability is $1/n$, as we are choosing uniformly. Also because we are finding the corresponding nut $n_i$ associated with $b_i$, we are splitting the problem into two, of sizes $i-1, n-i$, which is the same in quicksort. So if we have that $T(n)$ is the running time of the algorithm

$$E[T(n)] = (n-1) + \frac{1}{n} E[T(n-i) + T(i-1)]$$

which we have solved in class to give

$$E[T(n)] = O(n \log n)$$

Question 4. (15 points) Unfair coins

We solve this problem by mathematical induction, on the hypothesis that the probability of getting an odd number of heads for $n$ coins $= \frac{2^n}{2^n+1}$.

For $n = 1$, the probability of getting 1 heads $= \frac{1}{3}$, which proves the base case.

Now suppose that the induction hypothesis is true for $n$ and we want to prove it for $n+1$. Let $1_n$ be the indicator variable for the event of having an odd number of heads when these $n$ biased coins are tossed, and $X_n$ be the random variable denoting the result of the coin toss of the $n$'th coin ($H/T$). We have that $Pr(1_n = 1) = \frac{n}{2^n+1}$, from our induction hypothesis.

Suppose now that we have $n+1$ coins to be tossed. $1_{n+1} = 1$, if

1. if the first $n$ coins have an odd number of heads and the $n+1$'th coin is tails OR
2. the first $n$ coins have an even number of heads and the $n+1$'th coin is heads

By conditional probability and independence of the coin tosses we have that

$$Pr(1_{n+1} = 1) = Pr(1_n = 1) \cdot Pr(X_{n+1} = T) + Pr(1_n = 0) \cdot Pr(X_{n+1} = H)$$

$$= \frac{n}{2^n+1} \cdot \frac{2n+2}{2n+3} + \frac{n+1}{2^n+1} \cdot \frac{1}{2n+3}$$

$$= \frac{2n^2 + 2n + n + 1}{(2n+1)(2n+3)}$$

$$= \frac{(2n+1)(n+1)}{(2n+1)(2n+3)}$$

$$= \frac{n+1}{2(n+1)+1}$$

Question 5. (15 points) Picking a stream

This is a simplified version of a more generalized class of problems called as reservoir sampling.

The general idea of this algorithm is that, as we should be ready for the stream to stop at any point in time, we should start picking new items with a decaying probability.

The generalized problem is to sample $k$ items from a stream uniformly at random, where we don’t know the length of the stream. The algorithm is given by

1. Store the first $k$ items in an array $[a_1, \ldots, a_k]$. 
2. When we see the new (i’th) item $x_i, i > k$

(a) Randomly choose a number, $s$, between 1 and $i$ (which is done in constant time, as mentioned in the problem).

(b) If $s \leq k$, replace $a_s$ by $x_i$.

We want to show that at any point in the stream, when we have viewed some number of items, say $n$, the probability for any of the items, $x_1, \ldots, x_n$ to be in the $k$ selected items is $\frac{k}{n}$.

Suppose that we see a new item $x_i$. We replace one of the items $a_1, \ldots, a_k$ with $x_i$ only when $s \leq k$. As $s$ is sampled uniformly from $1, \ldots, i$, the probability that one of the items is replaced is equal to $\frac{k}{i}$. Hence when we see a new item, we only choose is with probability $\frac{k}{i}$.

Now we will show by induction that if an item $a_j$, is in the $k$ selected items, after seeing the $i$’th item, $x_i$, it will stay in the selected items with probability $\frac{k}{i}$. We will prove this by induction on $i$.

The base case is $i = k$, in which case all the first $k$ items are chosen with probability $1$.

Now suppose that at the end of the $(i-1)$’th item, the probability that $a_j$ is still in the samples is $\frac{k}{i-1}$. We will replace $a_j$ by $x_i$ with probability $\frac{1}{i}$. Hence $a_j$ does not get replaced with probability $1 - \frac{1}{i} = \frac{i-1}{i}$. Hence the probability that $a_j$ remains in the sample after seeing the $i$’th item $= \frac{k}{i-1} \cdot \frac{i-1}{i} = \frac{k}{i}$.

Hence proving our induction hypothesis.

**Question 6. (20 points - Extra credit) Valleys in a permutation**

Let $1_i$ be the indicator variable for the event that index $i$ is a valley.

Then $V = \sum_{i=1}^{n} 1_i$, is the number of valleys in the permutation.

Therefore $E[V] = \sum_{i=1}^{n} E[1_i] = E[1_1] + E[1_n] + \sum_{i=2}^{n-1} E[1_i]$.

$$Pr(1_1 = 1) = Pr(1_n) = \frac{1}{2} \implies E[1_1] = E[1_n] = \frac{1}{2}$$

This is because the first and last positions are valleys for half the permutations and not a valley in the other half, where you switch the order of the first/last two numbers.

Now let us focus on an inner index $i$, which has two neighbours $i-1, i+1$. We are only concerned about the orders of the numbers which means that we can think of the numbers in these positions to be $1, 2, 3$, in some order. There are 6 permutations of $1, 2, 3$

$$1, 2, 3$$

$$1, 3, 2$$

$$2, 1, 3$$

$$2, 3, 1$$

$$3, 1, 2$$

$$3, 2, 1$$

Out of which, only 2 permutations have the middle index as a valley.

Hence the probability that index $i$ is a valley $= \frac{1}{3}$, if $i$ is an inner index.

$$E[1_i] = \frac{1}{3} \quad \forall \ i \in \{2, \ldots, n-1\}$$
\[ E[V] = 2 \cdot \frac{1}{2} + (n-2) \cdot \frac{1}{3} = \frac{n+1}{3} \]

**Question 7. (30 points - Extra credit) Querying strings**

1. (a) There are \(2^n\) total binary strings, out of which \(\binom{n}{\frac{n}{2}}\) have reduced similarity = \(\frac{n}{2}\). Hence the probability of a randomly generated string having reduced similarity = \(\frac{n}{2}\) is

\[
\frac{\binom{n}{\frac{n}{2}}}{2^n} = \frac{\binom{n!}{\frac{n}{2}!\cdot \left(n-\frac{n}{2}\right)!}}{2^n} \approx \frac{\sqrt{2\pi\frac{n}{2} \left(\frac{n}{2}\right) \left(\frac{n}{2}\right)^n}}{2^n} \approx \frac{\sqrt{2\pi\frac{n}{2} \left(\frac{n}{2}\right)^n}}{2^n} = \frac{\sqrt{2\pi\frac{n}{2} \left(\frac{n}{2}\right)^n}}{2^n}
\]

(b) The algorithm is to keep on randomly generating strings till we get a string with reduced similarity \(\frac{n}{2}\).

As we are generating each string independently, the expected number of strings we have to generate are \(\frac{1}{\sqrt{\frac{2\pi\frac{n}{2} \left(\frac{n}{2}\right)^n}}}
\]

To generate each string we need \(O(n)\) time, which gives us an overall time complexity of \(O(n\sqrt{n}) = O(n^{\frac{3}{2}})\).

2. Now that we have a string \(T\) which has half the bits same as that of \(S\), we need to find which of the bits are incorrect.

For each bit, \(i\), we have a variable \(1_i\), which is either \texttt{true}, if the \(i\)'th bit in \(T\) is the same as the \(i\)'th bit in \(S\) and \texttt{false}, otherwise.

Assume that \(1_1 = \texttt{true}\). Now flip the 1'st and the 2'nd bit to get \(T^2\) and query the reduced similarity of \(T^2\). If we get back \(\frac{n}{2}\), it means that \(1_2 = \texttt{false}\), else it is \texttt{true}. Similarly for each bit \(i\), create \(T^i\), by flipping the first and the \(i\)'th bit together from \(T\) and query the reduced similarity, to get \(1_i\).

Now even when we don't know \(1_1\), we do know that all indices \(i\) which give back reduced similarity \(\frac{n}{2}\) are in the opposite state of the first bit. So we can query two final strings \(T'\) and \(T''\), which assume the two different states of the first bit, \(1_1\), and we will finally get hidden string.

This takes \(O(n^2)\) time as we have to construct and query \(O(n)\) strings, each of which take \(O(n)\) time.