// FOR GRADER: Texts colored in green are importants derivations and equations.
// FOR GRADER: Texts colored in red are results and conclusions.
// In question1, I defined two additional X_i values to make the thought process more
// thorough and complete

Question1:

Step1: Declare some data structures

1. Let **F[1......n]** be an array that contains all files inside the evidence room
2. Let **k > 0** be the particular file we want to look for
3. Let **Count** be a random variable represent the number of accesses to the file pool
4. Let X_i be a random variable that denotes the "status" of each file such that
   X_i = -1 if the i^{th} file in the file pool is not yet accessed and compared
   X_i = 1 if the i^{th} file in the file pool is accessed and compared.
   X_i = 0 if the i^{th} file in the file pool should be discarded.

Step2: Algorithm Analysis (linearity of expectation analysis)

1. **RANDY**(l, u) = (i, a_i) algorithm implies that this particular algo will only compare each item i in the file pool with k only one
2. By linearity of expectation: $E(Count)$ **=** $\sum_{i=1}^{n} E(X_i)$, and since we know that, according to the principle of linearity of expectation:
   $E(randomVariable)$ **=** $r_1 * p_1 + r_2 * p_2 + \ldots\ldots\ r_n * p_n$ **⟹ (1)**
   where r_i is the value that random variable can have, and p_i is the probability corresponding to having that value. In our case, X_i can have two value = 1, -1, or 0

3. $E(X_i)$ will be reduced to:
   1 * p($X_i = 1$) + 0 * p($X_i = 0$) + -1 * p($X_i = -1$) = p($X_i = 1$) - p($X_i = -1$) ⟹ (2)

   Equation (2) implies that what we need to check is only the POSSIBILITY for some file in the file pool to be visited or not visited. However, since when X_i = -1, by the definition above, the i^{th} file is not yet accessed or compared. So we only need to compute the possibility for those files that are visited (That is compute the expectation of X_i = 1), that is:

$E(X_i)$ = 1 * p($X_i = 1$) = p($X_i = 1$) ⟹ (3) // this is the one we actually need to compute

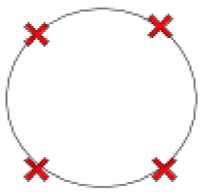
Step3: Final derivation and analysis

1. Assume that we have an index j such that F[j] $\leq$k $\leq$ F[j + 1] , and we need to find such a j. when we make the RANDY() method call and get some (i, a_i)
   1) $i \leq j$, RANDY( ) will return F(i......j), and **p($X_i = 1$) =** $\frac{1}{j-i+1}$
   2) If $i > j$, RANDY( ) will return F(j+1......i), and **p($X_i = 1$) =** $\frac{1}{i-j}$

probability equation: **$E(X_i)$ = ($\frac{1}{j-i+1}$ ($i \leq j$), or $\frac{1}{i-j}$ ($i > j$) ⟹ (4)**

Derivation Steps:

1) $E(Count) = \sum_{i=1}^{n} E(X_i) = \sum_{i=1}^{j} \frac{1}{j-i+1} + \sum_{i=j+1}^{n} \frac{1}{i-j}$ **⇒ (5)**
2) because $\frac{1}{j-i+1} < \frac{1}{i}$ and $\frac{1}{i-j} < \frac{1}{i}$, $\frac{1}{j-i+1} + \frac{1}{i-j} < 2 * \frac{1}{i}$ **⇒ (6)**
3) $E(Count) = \sum_{i=1}^{n} E(X_i) = \sum_{i=1}^{j} \frac{1}{j-i+1} + \sum_{i=j+1}^{n} \frac{1}{i-j} < \sum_{i=i}^{n} \frac{1}{i} + \sum_{i=i}^{n} \frac{1}{i}$ **⇒ (7)**
4) **By the harmonic series,** $\sum_{i=i}^{n} \frac{1}{i}$ **=** $O(logn) \rightarrow E(Count)$ **<** $2O(logn)$ **=** $O(logn)$ **⇒ (8)**
5) **COMPLETE**



Question2:

a) As my drawing above, if we pick k points on the circle, then we will have k arcs (segments) and because the circle is of a unit circumference, so the sum of every single expectation is supposed to be 1. Assume each each arc is X_i, because of the above explanation, we will get an equation:

$\sum_{i=1}^{k} E(X_i) = 1$ **⇒ (1)**

Equation (1) implies that the expectation for some X_i is $\frac{1}{k}$, therefore the expectation for a single arc is $E(X_i)$ **=** $\frac{1}{k}$ **(COMPLETE)**

1) **Pseudocode:**

Algorithm of finding the file with ID x

Declare some variables:
Let fileID = the file ID received through calling **RANDY( )**
Let leftBound = the leftmost fileID
Let rightBound = the rightmost fileID
// we can choose $\sqrt{n}$ from the circularly sorted linkedList
// This is random access
for i from 1 to $\sqrt{n}$:
  Initialize leftBound = 0;
  Initialize rightBound = infinity;
  fileID = **RANDY( )**
  if x == fileID $\rightarrow$ return x
  if fileID > leftBound && fileID < x $\rightarrow$ update leftBound = fileID
  if fileID < rightBound && fileID > x $\rightarrow$ update rightBound = fileID
end for

// This is linear access
for j from leftBound to rightBound:
  Let nextID = **NEXT**(j);
  if nextID == x $\rightarrow$ return x
end for

---

2) Description & explanation of the algorithm:

a) The purpose of declaring the variable leftBound and rightBound is that aftering calling **RANDY()**, we would have a particular file ID that is either larger or smaller (or perhaps equal in the best case scenario) than x, we need to update the leftBound or the rightBound so we can narrow down the range and bring the bound closer and closer to x.

b) The purpose of the first for loop is to get a leftBound and rightBound that is closest to x. After the first for loop finished, we would have a leftBound and rightBound pair that is the closest to the actual x.

c) And the second for loop serves to call **NEXT()** from leftBound to rightBound, and compare each return value to x. Since given the statement that all files are organized and sorted in a way that every file points to the one with the next higher file ID, we will find x between (leftBound, rightBound).

3) Running Time Analysis

a) **Arc length analysis:** Since we know from 2(a) that the expected arc length of any arc in the circle is $\frac{1}{k}$ (with a unit circumference). 2(b) has a little bit different context, this is a "circle" with a circumference = n since there are n files inside the file pool, so the expectation for any single arc is $\frac{n}{k}$. However, in the above algo we have set k to be $\sqrt{n}$, and $\frac{n}{k} \rightarrow \frac{n}{\sqrt{n}} = \sqrt{n}$. In another word, the expected arc length for the above algo is $\sqrt{n}$.

b) **First loop**: We will call the first loop $\sqrt{n}$ times, and in each iteration the only operations are calling **RANDY()** and doing comparisons, both of which will take $O(1)$ time. So the running time for the first loop is:
$O(\sqrt{n})$ * $O(1)$ = $O(\sqrt{n})$

c) **Second loop:** we will call the second loop $\sqrt{n}$ times (expected arc length), and in each iteration the sole operation is calling **NEXT(),** which will take $O(1)$ time, on the range between leftBound and rightBound. So the running time for the second loop is:
$O(\sqrt{n})$ * $O(1)$ = $O(\sqrt{n})$

d) **Total Running Time**: $O(\sqrt{n})$ + $O(\sqrt{n})$ = 2 $O(\sqrt{n})$ = $O(\sqrt{n})$ **(COMPLETE)**