CS 577: Introduction to Algorithms	Homework 7
Out: 03/23/21	**Due: 03/30/21**

Name: Aiden Tepper Wisc ID: 9081242969

Ground Rules

- Answer the questions in the boxes provided on the question sheets. If you run out of room for an answer, add a page to the end of the document. Do **not** feel obligated to fill the entire solution box. The size of the box does **not** correspond to the intended solution length.
- The homework is to be done and submitted individually. You may discuss the homework with others in either section but you must write up the solution *on your own*.
- You are not allowed to consult any material outside of assigned textbooks and material the instructors post on the course websites. In particular, consulting the internet will be considered plagiarism and penalized appropriately.
- The homework is due at 11:59 PM CT on the due date. No extensions to the due date will be given under any circumstances.
- Homework must be submitted electronically on Gradescope.

Problem 1

You are a police officer trying to crack a case. You want to check whether an important file is in the evidence room. Files have IDs that are positive integers and the evidence room contains n files in sorted order of their IDs. Unfortunately, you do not have direct access to the evidence room; the only one who has access is Randy, who is corrupt and wants to make things hard for you. In the following we assume that x is the file ID you are looking for.

1. You know that the evidence room is organized as a sorted list of n elements. If Randy was not corrupt you would probably do binary search by asking him to give you the median file of the list. Unfortunately, you can only give Randy two indices l, u and he returns to you a file with index chosen uniformly at random from the range $\{l, \ldots, u\}$. That is you can call

   $$\text{RANDY}(l, u) = (i, a_i), \qquad \begin{array}{l}\text{where } i \text{ is a uniformly random integer chosen from } l, \ldots, u \text{ inclusive} \\ \text{and } a_i \text{ is the ID of the corresponding file.}\end{array}$$

   You solve the problem by doing something similar to binary search. You start by calling $\text{RANDY}(1, n)$. Let's assume that Randy returns (i, a_i). You compare x to a_i.

   - If $x = a_i$ you found the file you were looking for.
   - If $x < a_i$ you call $\text{RANDY}(1, i-1)$
   - If $x > a_i$ you call $\text{RANDY}(i+1, n)$.

   You continue recursively until you have either found the file or conclude that the file is not present in the evidence room.

Show that the above algorithm accesses $O(\log n)$ files in expectation before it terminates.

Define $a_j \le x < a_{j+1}$, $X_i = 1$ if RANDY returns a_i and 0 otherwise.

The expected running time of the algorithm is $E[\#\text{ of values Randy returns}] = E[X_1+X_2+\ldots+X_n] = E[X_1]+E[X_2]+\ldots+E[X_n]$

Note that $E[X_i] = P(X_i=1)$. We compute $P(X_i=1)$ by finding $P(X_i=1 \mid \text{value of } X_i \text{ is determined on round } k)$ across all k. There are two cases:

a) $i \le j$: X_i is set if RANDY returns a value from a_i to a_j. It is set to 1 if RANDY returns a_i. The probability it is set to 1 given that it is set is $1/(j-i+1)$.

b) $i > j$: X_i is set if RANDY returns a value from a_{j+1} to a_i. It is set to 1 if RANDY returns a_i. The probability it is set to 1 given that it is set is $1/(i-j)$.

Note that $P(X_i=1 \mid X_i \text{ is determined on round } k)$ is independent of k. As such, $P(X_i=1) = P(X_i=1 \mid X_i \text{ is determined on round } k)$ for all $k = \max(1/(j-i+1), 1/(i-j))$.

Then $E[\#\text{ of calls to RANDY}] = P(X_1=1)+P(X_2=1)+\ldots+P(X_n=1) = 1/j+1/(j-1)+1/(j-2)$

$+\ldots+1/2+1+1+1/2+1/3+\ldots+1/(n-j) \le 2(1+1/2+1/3+\ldots+1/n) \approx 2\ln(n) = O(\log n)$.

2. With his trick in the previous question Randy was not able to slow you down very much[1]. Now he decides to disallow "range" queries as above and only allows either sequential access to the files or access to a uniformly random file from the entire set. In particular, you now have two ways of accessing the list:

   - By looking at a uniformly random element of the list. That is by calling

     $$\text{RANDY}() = a_i, \text{ where } i \text{ is chosen uniformly at random from } 1, \ldots, n, \text{ inclusive.}$$

     Observe that you only receive the file ID, not the index of the file.
   - By asking Randy to give you the file directly following one he returned to you in some previous call. For example if you first call RANDY() and get back a_i you are allowed to call NEXT(a_i) and get back a_{i+1}. Note that the list wraps around, so that NEXT(a_n) returns a_1. If you haven't already obtained a_i in some previous call you may not call NEXT(a_i).

   To facilitate analyzing this setting, think of the files as being organized in the form of a *circular sorted linked list* where every file points to the one with the next higher ID.

   (a) As a warm up, let us analyze the following setting. You are given a circle of unit circumference. You pick k points on the circle independently and uniformly at random and snip the circle at those points, obtaining k different arcs. Determine the expected length of any single arc.
   (Hint: Note that the length of each arc is identically distributed, so each has the same expectation. What is the sum of their expectations?)

If k points are picked, then the average length of each arc is 1/k. For any cut k, let it be denoted as q_i.

$\sum_{i=1}^{n} q_i = 1$, and because there are k cuts, then the average of the k cuts is $\frac{\sum_{i=1}^{n} q_i}{k} = \frac{1}{k}$. Therefore, the expected length of any single arc is 1/k.

(↑ because of unit circumference)

[1] This randomized binary search is in expectation roughly as fast as the usual deterministic one.

(b) Develop a randomized algorithm for finding the file with ID x that makes at most $O(\sqrt{n})$ calls to the functions NEXT() and RANDY() in expectation and always returns the correct answer. Analyze the running time of the algorithm. A proof of correctness is not necessary.

(Hint: Your algorithm will perform some random accesses and some amount of linear search. Use part (a) to analyze the number of steps in the linear search)

Input: x, FileID, list of files Output: file

```
Algorithm FindFile:
  H=RANDY()
  if H = x:
    return H
  else:
    while H > x:
        H=RANDY()
        if H=x:
            return H
  while x != H:
    H=NEXT(H)
  return H
```

Runtime analysis:

within expectation, x will on average be the midpoint of 1...n, which is n/2. Then, there's a 50% chance that the file is in the half that is less than x. Within that half, the expected value we would get is the 25th %. So, we call NEXT() until we arrive at x after that. Therefore, the runtime is $O(\log n)$.