CPSC 351 — Tutorial Exercise #20

Application — The Analysis of Algorithms

The goal of this exercise is to give you practice applying probability theory to analyze the performance of algorithms.

Problems To Be Solved

1. Once again, consider the "Integer Search" decision problem: The inputs are an integer array `A` and an integer `key`. The algorithm must return `true` if there is at least one copy of the `key` stored in the array and it must return `false` otherwise.

   Consider the randomized algorithm shown in Figure 1 on page 2. This calls (as a subroutine) the deterministic algorithm, `dSearch`, that is shown in Figure 2 on page 3.

   As in the preparatory lecture material, and the lecture presentation, let's keep things simple by counting executions of numbered steps in these algorithms in order to define their "running times". Then — as explained the lecture presentation for Lecture #22 — the number of steps used by an execution of the `dSearch` algorithm is $3i + 5$ if $0 \leq i \leq n - 1$ and the first copy of the `key` in the input array is in location i. The number of steps used is $3n + 4$ if the n is the length of the input array and the `key` is not stored in this array, at all.

   (a) Explain why the algorithm, shown in Figure 1, is a ***Las Vegas*** algorithm for the "Integer Search" decision problem.

   (b) There is no longer any upper bound on the number of executions of the body of the `while` loop! With that noted, describe, as precisely as you can, a ***sample space*** $\widehat{\Omega}$ that can used in an analysis of the expected number of steps used by this algorithm on a given input.

   (c) Describe a ***probability distribution*** $\widehat{\mathsf{P}} : \widehat{\Omega} \to \mathbb{R}$ for this sample space (that reflects the usual assumption that numbers really *can* be chosen uniformly and independently from finite sets, as the algorithm requires).

```
boolean rSearch4 ( integer[] A, integer key) {
 1. integer n := A.length
 2. while (true) {
 3.   Choose j uniformly from the set                                    .
```

$$\{0, 1, 2, \ldots, \mathtt{n} - 1\}$$

```
      — independently from any previous selections
 4.   if (A[j] == key) {
 5.     return true
      }
 6.   Choose an integer continue uniformly from the set {0, 1} (so that the probability
      that continue = 1 is 1/2), independently from any previous selections.
 7.   if (continue == 0) {
 8.     return dSearch(A, key)
      }
    }
}
```

Figure 1: Another Randomized Algorithm for the "Integer Search" Problem

(d) Describe, as precisely as you can, a ***random variable*** $\widehat{T} : \widehat{\Omega} \to \mathbb{R}$, representing the number of steps used by an execution of this algorithm.

(e) Compute the ***expected value*** of the random variable $\widehat{T}$ with respect to your probability distribution. Note that this is the value that is called the "expected running time" of the algorithm on a given input.

```
integer dSearch (integer[] A, integer key) {
1.  integer n := A.length
2.  integer i := 0
3.  while (i < n) {
4.    if (A[i] == key) {
5.      return true
      }
6.    i := i + 1
    }
7.  return false
}
```

Figure 2: Subroutine Implementing Linear Search

2. Consider a problem whose input is a Boolean array `A`. You know that ***either*** at least $\frac{3}{4}$ of the entries of the array are `true`, ***or*** at least $\frac{3}{4}$ of the entries of the array are `false` — but you do not know which is the case.

   Give a ***very simple*** randomized algorithm, with two-sided error, that receives a Boolean array `A`, as described, and returns `true` if at least $\frac{3}{4}$ of the array's entries are `true`, returning `false`, otherwise. (It does not matter what this algorithm does if its input does not satisfy the above condition.) Then explain, briefly, why your algorithm satisfies the properties described, for "randomized algorithms with two-sided error", as given in the preparatory reading for Lecture #22.