Average time of sequential search in unordered array

Example

Sequential search on an unordered array.

Find an item x in an unordered array I based only of comparisons of x to elements of I.

Notation

size (I) - number of elements to be searched.

T (n) - number of comparisons performed while searching of an entry in an n - element array I.

Average - case running time for successful search

$$T^{succ}_{avg}(n) = \sum_{i=0}^{n-1} Pr(I_i \mid success) \times t(I_i) =$$

$$\sum_{i=0}^{n-1} \frac{1}{n}(i+1)$$

$$\frac{1+n}{2}$$

For sequential search, $T^{succ}_{avg}(n)$ is the same for ordered and unordered array because sequential search dies not take advantage of the order while successfully searching for a key.

Average - case running time for unsuccessful search

$$T^{fail}_{avg}(n) = n$$

(Because for linear search, it's the worst case each time it's unsuccessful.)

Hence, $T_{avg}(n) = q * T^{succ}_{avg}(n) + p * T^{fail}_{avg}(n) =$

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

$$q \times \frac{1+n}{2} + (1-q) \times n$$

$$n(1-q) + \frac{1}{2}(1+n)q$$

Mathematica can't nicely simplify the above formula to a polynomial $A \times n + B = \left(1 - \frac{q}{2}\right) n + \frac{q}{2}$ for n.

As an exercise, let's pretend that we can't do it, eaither.

$\text{Limit}\left[\frac{1}{n}\left(n\,(1-q)+\frac{1}{2}\,(1+n)\,q\right),\ \{n\to\infty\}\right]$

$\left\{1-\frac{q}{2}\right\}$

$0<1-\frac{q}{2}<\infty$

So, $T_{avg}(n)\in\Theta(n)$

More precisely,

$T_{avg}(n)\sim\left(1-\frac{q}{2}\right)n$

meaning

$T_{avg}(n)=\left(1-\frac{q}{2}\right)n+o(n)$

$\left(n\,(1-q)+\frac{1}{2}\,(1+n)\,q\right)-\left(\left(1-\frac{q}{2}\right)n\right)$

$n\,(1-q)-n\left(1-\frac{q}{2}\right)+\frac{1}{2}\,(1+n)\,q$

Simplify[%]

$\frac{q}{2}$

Thus

$T_{avg}(n)=\left(1-\frac{q}{2}\right)n+\frac{q}{2}$

Optimality

Theorem

For an unordered array, sequential search is average - case optimal in the class C of algorithms that search by comparison of keys.

Proof.

1. Unsuccessful sequential search on an unordered array is average - case optimal in class C.

The lowewr bound on the number of comparisons is n, because n comparisons are needed in order to establish that an item x is not an element of an unordered array I.

This can be established as follows. First, give any search algorithm P an x to search in an array I that does not contain x. This will force P to perform n comparisons. Should P neglect to compare x to some element I[j] (here we use assumption that P searches by comparisons of keys), that element will be assigned value x after P halted, thus proving that P is incorrect.

(The above is called adversary strategy.)

But n is also the number that unsuccessful sequential search performs.

Hence the optimality of unsuccessful sequential search in class C.

2. Successful sequential search on an unordered array is average - case optimal in class C.

Regardless of the order in which x is compared to the elements of array I, any search that serches an unordered array by comparisons of keys can be forced (by an adversary strategy) for each
$0 \le i < n$
and some array I, to perform i + 1 comparisons, each with probability of $\frac{1}{n}$.

(Exercise : Design such an adversary strategy.)

This demonstrates that the formula

$$\sum_{i=0}^{n-1} \frac{1}{n} (i+1)$$

provides a lower bound for average number of comparisons while searching an unordered array of n elements.

But the above formula also provides the average number of comparisons sequential search will perform while searching an unordered array of n elements.

Hence the average - case optimality of sequential search on an unordered array.

Since both unsuccessful and successful sequential search of an unordered array are average - case optimal in class C, the sequential search of an unordered array is optimal in class C, too. □