Searching an Unordered List

Given a list L of length $n \geq 1$ over a data type D and a value $x \in D$, if x is in L return k ($1 \leq k < n$) such that $L[k] = x$, else return “not found”. If x occurs several times, any k such that $L[k] = x$ will do.

Here is an algorithm which solves the problem:
Linear Search (LS):

```
k = 0
while k < n:
        if L[k] == x:
                return k
        else:
                k = k + 1
return “not found”
```

Clearly LS finds the least k such that $L[k] = x$, if x is in L.

We would like to measure how long LS takes. One measure would be how many times the while loop is entered. However we want to compare LS to other algorithms, which might not use while loops. So instead we should measure the number of comparisons of the form $L[k] = x$, since this will generalise to other algorithms.

It is reasonable to expect the time an algorithm takes to depend on the size of the input. In this case a reasonable measure of input size is the length of L. But now consider the following examples.

Take n = 4 and D to be the natural numbers and x to be 5. Then LS makes 1 comparison on input $[5, 4, 3, 6]$ and 4 comparisons on $[3, 4, 6, 2]$.

In general with input size n, LS makes anything from 1 to n comparisons.

There are two standard solutions to this problem—*worst-case* and *average-case* analysis. We describe them for LS, but clearly the definitions can be made in general for any algorithm.

- Worst-case analysis: Let $W(n)$ be the largest number of comparisons made when performing LS on the whole range of inputs of size n

    - In this case $W(n) = n$
- Average-case analysis: Let $A(n)$ be the average number of comparisons made when performing LS on the whole range of inputs of size n

It might seem obvious that average-case analysis is best. However before we can say that it gives a better measure, we need to know what the probability distribution of inputs is. If we don't know this, we have to make assumptions, such as that x occurs with equal probability in all positions in the list.

Furthermore, worst-case analysis has two great advantages:

- It offers us a guarantee that LS will never take longer than $W(n)$
- $W(n)$ is easier to compute than $A(n)$, and often yields almost as good results. When we discuss orders of complexity we shall see that $W(n)$ and $A(n)$ are often of the same order

Because of this, it is usual to start with worst-case analysis.

What about *best-case analysis B(n)*? Suppose that an algorithm is very poor in general, but happens to do well when x occurs in the first position. Then perhaps $B(n) = 1$, but this is highly misleading. In this way, the most inferior algorithm can look good through hand-picked inputs.

We shall refer to functions from N to N like $W(n)$ and $A(n)$ as *complexity functions*.

LS can be varied (e.g. we could search L backwards, or in some other order). However, **LS is optimal**.

Proof:

- Take any algorithm A which solves the problem.
- We claim that if A returns "not found" it must have inspected every entry of L
- Suppose for a contradiction that A did not inspect $L[k]$ (some $k < n$)
    - Define a new list L' as follows
        - L' is the same as L except that $L'[k] = x$
        - Since A did not inspect $L[k]$ it will not inspect $L'[k]$ either, since the two lists are otherwise identical
        - So A must still return "not found"
        - However A has now given the wrong answer
        - Contradiction

- We conclude that in worst case A must inspect every entry, meaning that n comparisons are needed. So LS is optimal.

The argument was about all possible algorithms to solve the problem, not just those which have been so far devised. Of course proofs of optimality are not always so easy.

There are many problems, such as matrix multiplication, for which it is not known whether the best algorithm so far discovered is in fact optimal.