Searching an Ordered List

Given an ordered list L of length $n \geq 1$ over an ordered data type $(D, \leq)$ and a value $x \in D$, if x is in L return k ($0 \leq k < n$) such that $L[k] = x$, else return “not found”.

This problem is no harder than the previous one, so LS will solve it.

However the fact that L is ordered means that we can stop once we find a value in the list which exceeds x:

Modified Linear Search (MLS):

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

We regard the three-way conditional as performing a single comparison. We have made an improvement, in that we can return “not found” before inspecting the entire list. However, the worst-case analysis (W(n)) is still n, as we may have to inspect the whole list. The change would show up in the average-case analysis.

However there is a different algorithm which improves MLS drastically, namely binary search (BS).

Let x be a real number. The floor of x, denoted $\lfloor x \rfloor$, is the greatest integer $\leq x$ (if $x \geq 0$ this is the integer part of x). The ceiling of x, denoted $\lceil x \rceil$, is the least integer $\geq x$

Binary Search (BS):

```
procedure BinSearch(left, right):
        # searches for x in L in the range L[left] to L[right]
        if left > right:
                return “not found”
        else:
```

```
                    mid := ⌊(left+right)/2⌋
                    cond
                            x = L[mid]: return mid
                            x < L[mid]: BinSearch(left, mid - 1)
                            x > L[mid]: BinSearch(mid + 1, right)
```

The only comparison we consider for complexity reasons is the `cond` comparison. This only counts as **one** comparison.

A convenient representation of the algorithm is by means of decision trees. A node in a decision tree represents a comparison, and its children represent the next step in the algorithm depending on the result of the comparison.

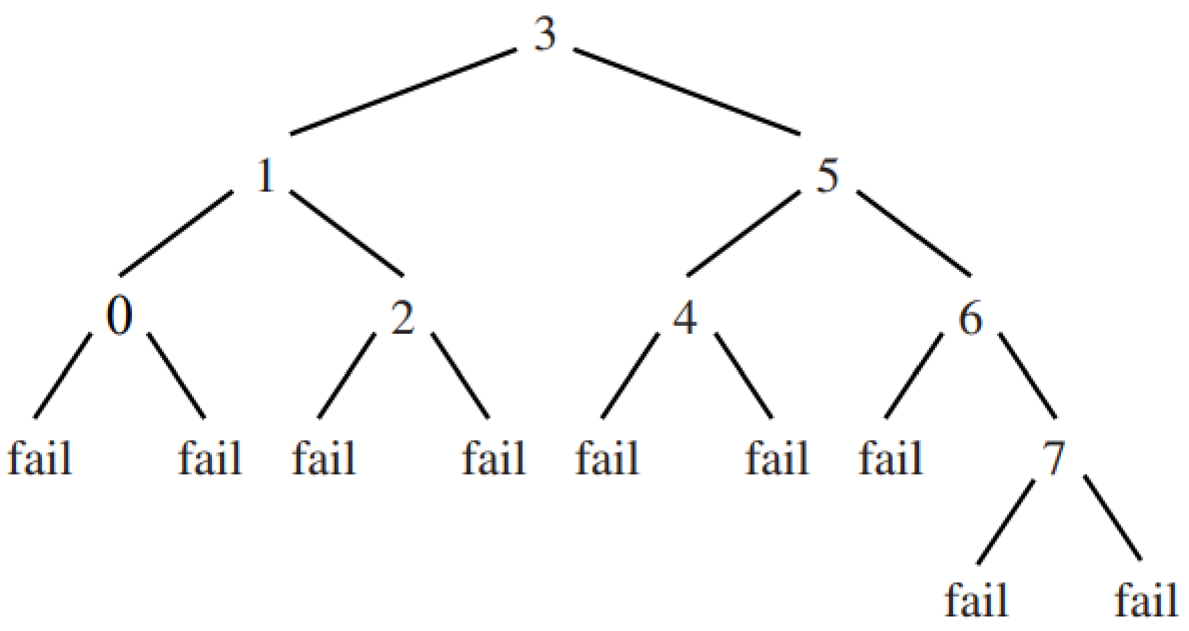

Figure 2.1: Decision tree for Binary Search ($n = 8$)

The left-hand child of a node k corresponds to $x < L[k]$. The right hand child corresponds to $x > L[k]$. If $x = L[k]$ then the algorithm stops straight away.

The tree shows us for instance that if $L[4] = x$ and $L[3] < x, L[5] > x$, then BS takes three comparisons to return the answer 4. It is immediate that the worst case number of comparisons is 4 (when $x > L[6]$, so that $L[3], L[5], L[6]$ and $L[7]$ are inspected). So $W(8) = 4$

We can draw a similar decision tree for every n, giving a complete description of how BS performs on any input of size n, and allowing us to find $W(n)$. Notice that the tree will contain n nodes, one for each of $0,\ldots,n-1$ (as well as nodes for fail).

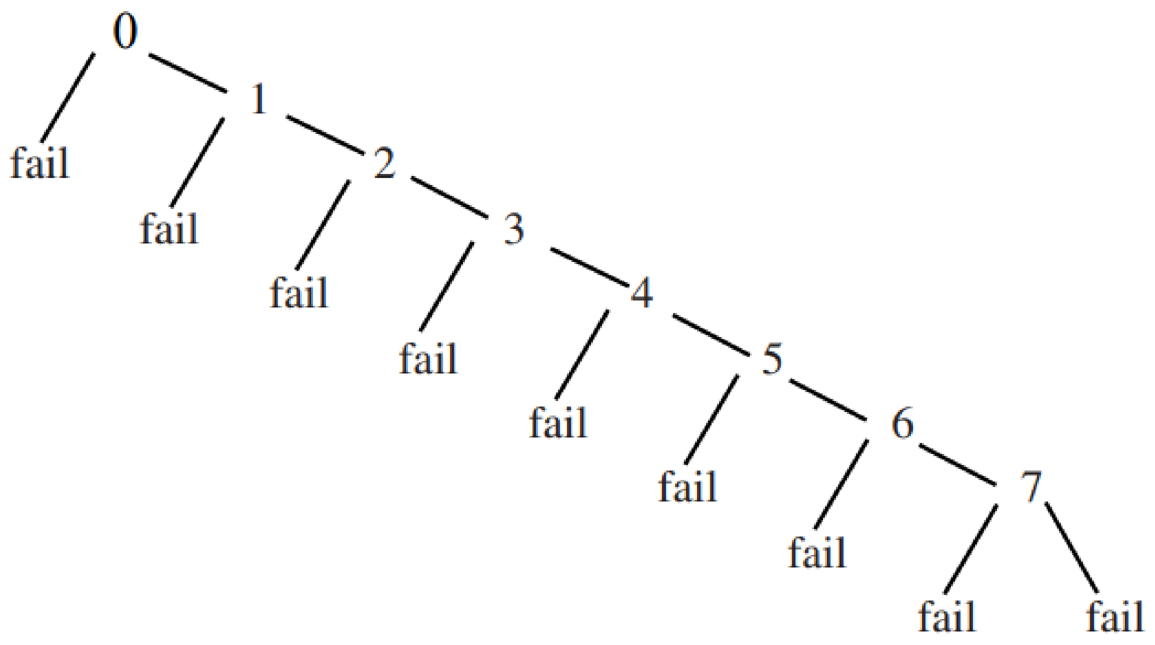

Figure 2.2: Decision tree for Modified Linear Search ($n = 8$)

As an illustration of how BS is better than MLS, this shows a decision tree for MLS on n = 8. It has the same number of nodes, but greater depth, corresponding to higher worst-case complexity.

BS has succeeded in reducing depth by “widening” the tree. BS is in fact optimal, and the reason is that it does the best possible job of keeping the tree-depth low for a given number of nodes.

Before showing that BS is optimal, we calculate $W(n)$. Since BS is recursive, we cannot find $W(n)$ directly. However we can write down the following recurrence relation:

$$\begin{aligned} W(1) &= 1 \\ W(n) &= 1 + W(\lfloor n/2 \rfloor) \end{aligned}$$

Either we find $L[\mathrm{mid}] = x$ immediately, or else we call the procedure on a portion of the array which has half the length. We solve for $W(n)$ by repeated expansion:

$$\begin{aligned} W(n) &= 1+W(\lfloor n/2 \rfloor) \\ &= 1+1+W(\lfloor n/4 \rfloor) \\ &\dots \\ &= 1+1+\cdots+1+W(1) \end{aligned}$$

The number of 1s is the number of times that we can divide n by 2. Suppose that $2^k \le n < 2^{k+1}$. We can always find such a k. Then the number of 1s in the summation is k. If we denote the logarithm to base 2 of n by $\log n$ then k = $\lfloor \log n \rfloor$. Therefore (since $W(1) = 1$):

$W(n) = \lfloor \log n \rfloor + 1$

Consider any algorithm A for searching an ordered list of length n. We can represent it by a binary decision tree. The tree will have at least n nodes since the algorithm must have the capability to inspect every member of the list (by an argument similar to that for unordered lists). Let the depth of the tree be d (not counting fail nodes, since fail is not a comparison). The worst-case performance of A will be $d + 1$. We shall show that d has a minimum value.

A binary tree is a tree where each node has at most two children. Denote the depth of a tree T by $\mathrm{depth}(T)$, the number of nodes in T by $\mathrm{nodes}(T)$, and the root node of T by $\mathrm{root}(T)$.

Suppose that T is a binary tree with depth d. Then T has no more than $2^{d+1} - 1$ nodes

Proof. By induction.

Base case $d = 0$. Then T has at most 1 node (the root), and $1 \le 2^{0+1} - 1$.

Induction step. Assume true for all numbers $\le d$ and show for $d+1$. Let T be a tree of depth $d+1$. Suppose $\mathrm{root}(T)$ has children T_1, T_2. Then $\mathrm{depth}(T_1) \le d$, $\mathrm{depth}(T_2) \le d$. So by induction T_1 and T_2 each have no more than $2^{d+1} - 1$ nodes. But

$$\#\mathrm{nodes}(T) = 1 + \#\mathrm{nodes}(T_1) + \#\mathrm{nodes}(T_2)$$

Hence $\#\mathrm{nodes}(T) \le 1 + 2(2^{d+1} - 1) = 2^{(d+1)+1} - 1$.

If $\mathrm{root}(T)$ has only one child then the argument is similar but easier.

From Lemma 2.3.4 we see that if T has n nodes and depth d then $n \le 2^{d+1} - 1$. So $d + 1 \ge \lceil \log(n+1) \rceil$.

Lemma 2.3.5 $\lceil \log(n+1) \rceil = \lfloor \log n \rfloor + 1$.

Proof. Let $k = \lfloor \log n \rfloor$. Then $2^k \leq n < 2^{k+1}$. So $2^k < n+1 \leq 2^{k+1}$. Hence $k < \log(n+1) \leq k+1$ and the result follows.

Using Lemma 2.3.5, we see that the worst-case behaviour of A (namely d + 1) is at least $\lfloor \log n \rfloor + 1$. This gives us:

Lower Bound For Searching: Any algorithm for searching an ordered list of length n for element x, and which only accesses the list by comparing x with entries in the list, must perform at least $\lfloor \log n \rfloor + 1$ comparisons in worst case. But $W(n)$ for BS is $\lfloor \log n \rfloor + 1$

Therefore, BS is optimal.