Worst – case lower bound on searching for x in a sorted array E with n elements

Carries on the details of computations in the textbook, Section 1.6. 4 Optimality, p. 59 – 60.

The result is proven for any algorithm that searches via decision tree;
in particular, for any algorithm that searches by comparisons of keys.

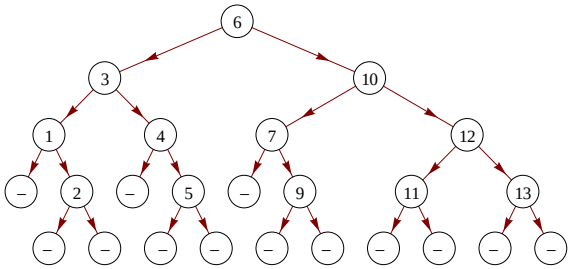
Eg. : 1 2 3 4 5 6 7 9 10 11 12 13

n = 12

x = 8 (unsucessful)

Decision tree T for searching :

```
TreePlot[{6 → 3, 6 → 10, 3 → 1, 3 → 4, 10 → 7, 10 → 12, 1 → "–",
  1 → 2, 4 → "– ", 4 → 5, 7 → " – ", 7 → 9, 12 → 11, 12 → 13, 2 → " – ",
  5 → "  – ", 9 → "  –  ", 11 → "  –  ", 13 → "  –  ", 2 → "  – ",
  5 → "  – ", 9 → "  –  ", 11 → "  –  ", 13 → "  –  "},
 VertexLabeling → True, DirectedEdges → True, VertexRenderingFunction →
  ({White, EdgeForm[Black], Disk[#, .2], Black, Text[#2, #1]} &), AspectRatio → 0.47]
```



p = the length of the longest path in T from
the root to any decision node (p = 3 on the above picture)

p + 1 = the number of comparisons done along the longest path in T (p = 4 on the above picture)

N – the number of decision nodes in the decision tree

n ≤ N

What is the length p of a longest path (from the root down, measured by the number of edges passed) in a shortest decision tree T with N decision nodes?

We will use an example of the shortest decision tree with N decision nodes. Because all shortest decision trees with N nodes have the same depth, using that example will not constrain the generality of proof.

We define T so that all levels of T, except perhaps for the last level, are full (have maximum possible number of decision nodes). Clearly, one cannot have a shorter decision tree with N decision nodes than that.

At every level i of T, except, perhaps, for the last level p, the number of nodes is 2^i. So,

$N > 1 + 2 + 4 + \ldots + 2^{p-1} = \sum_{i=0}^{p-1} 2^i =$

$\sum_{i=0}^{p-1} 2^i$

$-1 + 2^p$

The last level p contains no more than 2^p nodes. So,

$N \leq 1 + 2 + 4 + \ldots + 2^p = \sum_{i=0}^{p} 2^i =$

$\sum_{i=0}^{p} 2^i$

$-1 + 2^{1+p}$

So, $-1 + 2^p < N \leq -1 + 2^{1+p}$

or $2^p < N + 1 \leq 2^{1+p}$

or $2^p \leq N < 2^{1+p}$

$p \leq \text{Log2}[N] < 1 + p$

or

$p = \lfloor \text{Log2}[N] \rfloor$

or

$p + 1 = \lfloor \text{Log2}[N] \rfloor + 1$

The above is a lower bound on the number of comparisons that any search that searches for an entry by comparing it to elements of the array must perform in the worst case.

Exercise: Prove it!

It is the depth of a shortest binary tree with N nodes.

Hence, binary search is worst case optimal in the mentioned above class of searching algorithms.