2 SEARCHING ON A SORTED LIST

**Problem:** Given a list $L[1, \ldots, n]$ containing

keys such that $L[i] \leq L[i + 1]$, for $i = 1, 2, \ldots, n - 1$.

Problem is to find out if a given $x$ is in $L$.

I. Use sorted property on sequential search

Quit search as soon as $x$ is less than $L[i]$ and declare that $x \not \in L$.

Could be shown that

$$A(n) = O(n) \approx n/2$$

$W(n)$ is still $O(n)$.
2.1 Jump Search

2.1.1 Jump Search (Divide and Conquer)

Scan every $i$th entry of $L$ for a fixed $i$. Suppose you have established that

$$L[2i] < x < L[3i]$$

then sequentially search $L[2i + 1], L[2i + 2], \ldots, L[3i - 1]$.

$$W(n) = \frac{n}{i} + (i - 1)$$

because there are $\frac{n}{i}$ partitions, and there are $(i - 1)$ items to search sequentially within a partition.

eg. $i = 4, w(n) = n/4 + 3$

$i = 10, w(n) = n/10 + 9$

$i = 100, w(n) = n/100 + 99$

but still $O(n)$ for fixed $i$.  

How about $i$ depending on $n$?

eg. $i = \log n$

\[ w(n) = \frac{n}{\log n} + \log n - 1 = O\left(\frac{n}{\log n}\right) \]

better than $O(n)$ of seq search.

\[ (n \not\in O\left(\frac{n}{\log n}\right)) \]

eg. $i = \sqrt{n}$

$W(n) = n/\sqrt{n} + \sqrt{n} - 1$

$= 2\sqrt{n} - 1 = O(\sqrt{n})$

better than $O\left(\frac{n}{\log n}\right)$. 

3
eg. \( i = \frac{n}{\log n} \)

\[
W(n) = \frac{n}{\log n} + \frac{n}{\log n} - 1
\]

\[
= \log n + \frac{n}{\log n} - 1 = O\left(\frac{n}{\log n}\right)
\]

Let us minimize for \( i \),

\[
W(n) = n/i + i - 1
\]

\[
\frac{d}{di} (W(n)) = -\frac{n}{i^2} + 1
\]

set to 0 and solve, gives \( i = \sqrt{n} \).

Hence \( W(n) = O(\sqrt{n}) \) is the best possible with the approach.
2.1.2 Recursively apply partitioning in a smaller list

Let us partition into $k$ intervals of size $n/k$ each

$$W(n) = n/i + i - 1$$  \hspace{1cm} (1)

$$W(n) = \frac{n}{n/k} + n/k - 1$$  \hspace{1cm} (2)

$$= k + \frac{n}{k} - 1$$  \hspace{1cm} (3)

$$= O(n/k)$$  \hspace{1cm} (4)

If we apply partitioning recursively in the sublist, we get

$$W(n) = k + W(n/k)$$

for $k$ partitions and each partition of size $n/k$.

Since, in order to identify a partition, we need not compare $x$ with $L[1]$,
\[ W(n) = k - 1 + W(n/k) \]

Say \( k = 4 \), then

\[
W(n) = 3 + W(n/4)
\]

\[
= 3 + \left( 3 + W\left( \frac{n}{4} \right) \right)
\]

\[
= 3 + 3 + W\left( \frac{n}{4^2} \right)
\]

\[
= 3 + 3 + \left( 3 + W\left( \frac{n}{4^3} \right) \right)
\]

\[
= 3 + 3 + \left( 3 + W\left( \frac{n}{4^3} \right) \right)
\]

\[
= 3 \times 3 + W\left( \frac{n}{4^3} \right)
\]

\[
= 4 \times 3 + W\left( \frac{n}{4^4} \right)
\]

\[
\ldots
\]

\[
= i \times 3 + W\left( \frac{n}{4^i} \right)
\]

How large can \( i \) be?

Eventually, partition size will become equal to 1, when

\[
y = 4^i \text{ or } \log_4 n = i
\]
Then \( W(1) = 1 \)

Thus,

\[
W(n) = 3 \log_4 n + W(1)
\]

\[
W(n) = 3 \log_4 n + 1
\]

\[
= O(\log_4 n)
\]

- much better than \( \sqrt{n} \).

In general, for any fixed \( k \),

\[
w(n) = (k - 1) \log_k n + 1
\]
2.2 RECURSIVE JUMP SEARCH: With Best Value for $k$

Minimize $W(n)$ w.r.t $k$.

Find $\frac{dw}{dk}$ & solve by setting to 0.

$$w(n) = (k - 1) \log_k n + 1$$

$$= (k - 1) \frac{\log_e n}{\log_e k} + 1$$

$$\frac{dw(n)}{dk} = (k - 1) \log_e n(-1)\left(\frac{1}{\log_e k}\right)^{-2} \frac{1}{k} + \frac{\log_e n}{\log_e k}$$

$$= -\frac{k - 1}{k} \frac{\log_e n}{(\log_e k)^2} + \frac{\log_e n}{\log_e k}$$

Set $\frac{dw}{dk} = 0$ and solve to get

$$\frac{k - 1}{k} = \log_e k$$

$$\Rightarrow \log_e k = 1 - \frac{1}{k} < 1$$

$$\Rightarrow k < e^1 = e = 2.7$$

$$\Rightarrow k = 2 \text{ (can not be 1)}$$
Further check that $\frac{d^2w}{dk^2}$ at $k = 2$ is $\geq 0$ for a minimum value.

Thus, $k = 2$ is the best. So, dividing in 3 partition is not better than that in 2.

$$w(n) = k - 1 + w(n/k)$$

$$= 2 - 1 + w(n/2)$$

$$= \log_2 n + 1$$

$$w(n) = \lfloor \log_2 n \rfloor + 1$$

Binary Search: For binary search, we assumed that $n$ is a power of 2.

If not, $w(n) = \lfloor \log_2 n \rfloor + 1$

$$A(n) = \log_2 n + 1/2$$

for binary search.
3 Optimality of Binary Search

**Computation Model:** Only operation allowed is comparison: Comparison Model

**To Show:** Binary Search is optimal in the class of search algorithms on an ordered list that can perform no other operation on the entries except comparison.
3.1 Decision Trees

- Sequential Search

\[ n = 16 \]
\[ w(n) = n \]

- Jump Search

\[ n = 16 \text{ sublist size } = \sqrt{16} = 4 \]
\[ w(16) = 7 = 2n - 1 = 2 \cdot 4 - 1 = 7 \]

- Binary Search

Middle = \( \lfloor \frac{first + last}{2} \rfloor \)
\[ w(16) = 5 = 4 + 1 = \lfloor \log_2 16 \rfloor \]
Proof: (Binary search is optimal)

- Numbers of nodes in any decision tree is $\geq n$

- Minimum numbers of levels in any binary tree with $n$ nodes is $\geq \lceil \log_2 n \rceil + 1$

  (H.W.)

- $\Rightarrow 1 + \lfloor \log_2 n \rfloor$ is a lower bound on problem complexity

- $\Rightarrow$ Binary Search is optimal