Upper bounds and lower bounds on the time necessary to solve the problem, with example FindMax

(See page 39 – 40 of textbook for more narrative.)

Suppose that a problem Q (I) has only three solutions whose worst – case running times are :

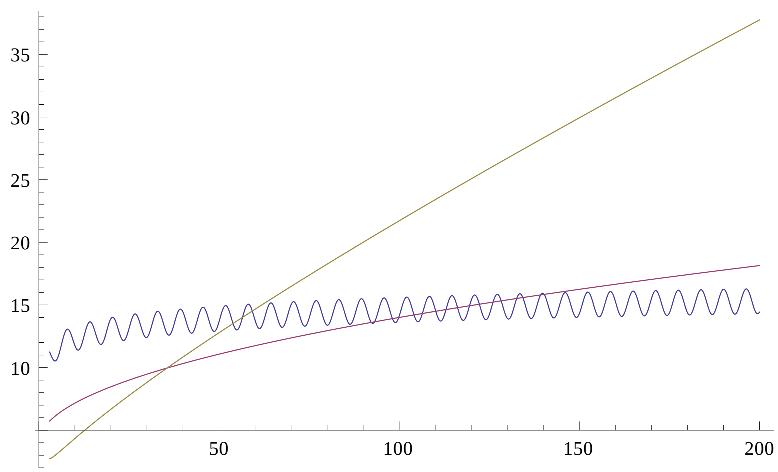
Log[n] + Sin[n] + 10,

$\sqrt{n}$ + 4, and

n / Log[n],

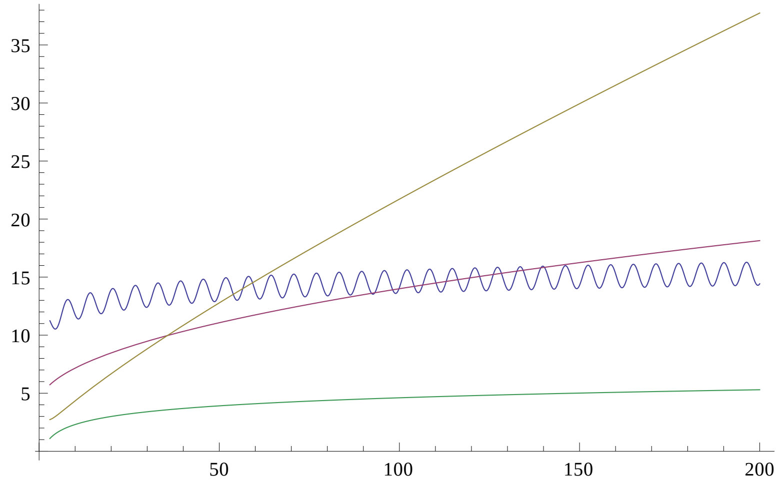
where n is the size of an input I.

Plot[Tooltip[{ Log[n] + Sin[n] + 10, $\sqrt{n}$ + 4, n / Log[n], }], {n, 3, 200}]



Function Log[n] is a lower bound of all three.

```
Plot[Tooltip[{ Log[n] + Sin[n] + 10, √n + 4, n / Log[n], Log[n]}], {n, 3, 200}]
```




```
Limit[(Log[n] + Sin[n] + 10) / Log[n], n → ∞]
```

1

$$\text{Limit}\left[\frac{\sqrt{n}+4}{\text{Log}[n]}, n \to \infty\right]$$

∞

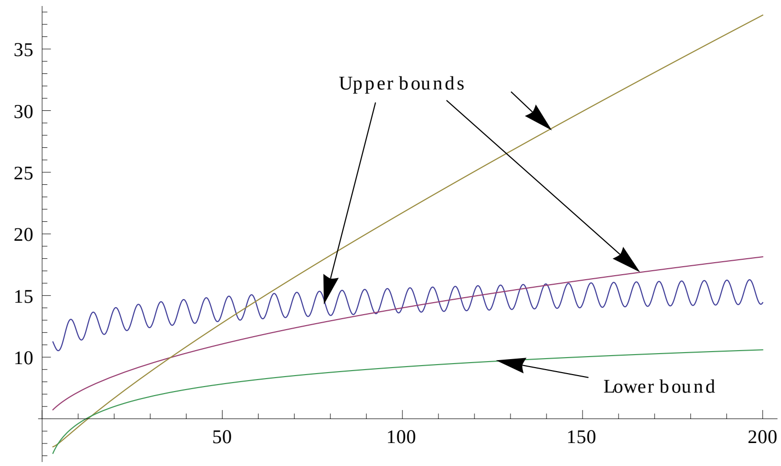
$$\text{Limit}\left[\frac{n / \text{Log}[n]}{\text{Log}[n]}, n \to \infty\right]$$

∞

So, Log[n] is a lower bound on the time necessary to solve the problem Q (I).

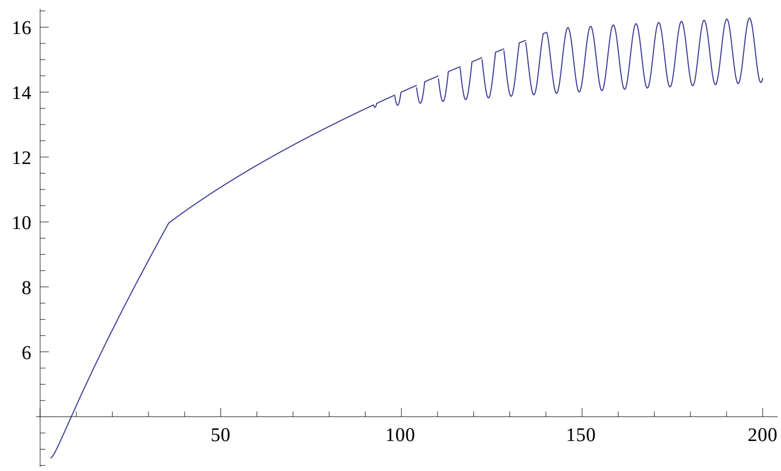
Each of the runnig times if problem' s solutions are (automatically)
upper bounds on the time necessary to solve the problem Q (I).

```
Plot[Tooltip[{ Log[n] + Sin[n] + 10, √n + 4, n / Log[n], 2 Log[n]}], {n, 3, 200}]
```



Here is an improved (smaller, that is) upper bound for Q :

```
Plot[Tooltip[Min[ Log[n] + Sin[n] + 10, √n + 4, n / Log[n]]], {n, 3, 200}]
```



So, if there were a solution that had worst-case running time Log[n] then such a solution would be worst-case optimal.

Average-case analysis is similar.

Example : FindMax.

Problem Q (I) : Given an unsorted integer array I of size n, find an index of the largest element of I, using comparisons of elements of array as the only means of deciding which one it is.

T (n) - the worst-case running time for the problem measured as the number of comparisonsthat are necessary to solve Q (I) for any input I of size n.

Lower bound on T (n).

Assume all elements of I are different.
The worst-case scenario will automaticallybe at least as bad as this.

There are (n - 1) non-maximal elements of I,
and the algorithm must "know" who they are.

The algorithm "knows" that an element x of I is non-maximal iff x lost at least one comparisonto some other element of I.

Each comparisonleaves one loser element.

An algorithm that performed no more than (n - 2) comparisonsidentified no more than (n - 2) losers.

So, at least 2 elements are non-losers, each of which may be the maximal one.

Hence, at least (n - 1) comparisonsare needed.

So,

$$f(n) = n - 1$$

is a lower bound on the worst-case running time of (any solution of) Q (I).

The worst-case running time of linear search is (automatically) an upper bound on the worst-case

Linear search finds max element of I after (n - 1) comparisons.

Hence, the worst-case running time of linear search is both a lower bound and an upper bound on the wost-case running time for Q (I).

In other words,
linear search is a worst-case optimal solution of Q (I) in the class of algorithmsthat make their decisions based only on comparisonsof keys.