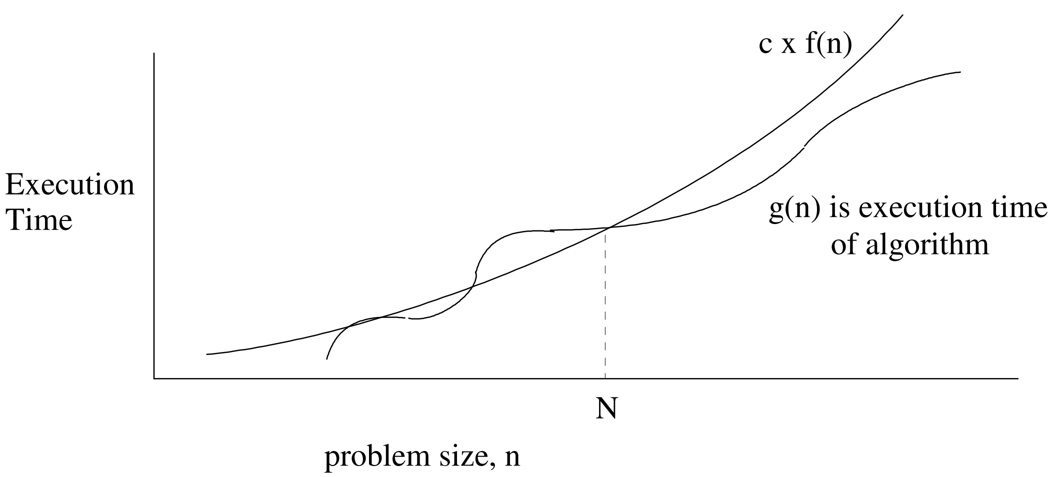
Algorithm 1.1 Sequential Search
Problem: Is the key *x* in the array *S* of *n* keys?
Inputs (parameters): positive integer *n*, array of keys *S* indexed from 1 to *n*, and a key *x*.
Outputs: *location*, the location of *x* in *S* (0 if *x* is not in *S*).

```
void seqsearch ( int n,
                 const keytype S[ ],
                 keytype x,
                 index & location ) {
  location = 1;
  while (location <= n && S[location] != x)
    location++;
  if (location > n)
    location = 0;
}
```

Big-oh Definition - asymptotic upper bound
For a given complexity function $f(n)$, $O(f(n))$ is the set of complexity functions $g(n)$ for which there exists some positive real constant c and some nonnegative integer N such that for all $n \geq N$,

$$g(n) \leq c \times f(n).$$



$T(n) = c_1 + c_2 n = 100 + 10 n$ is $O(n)$.

"Proof": Pick c = 110 and N = 1, then $100 + 10n \leq 110n$ for all $n \geq 1$.

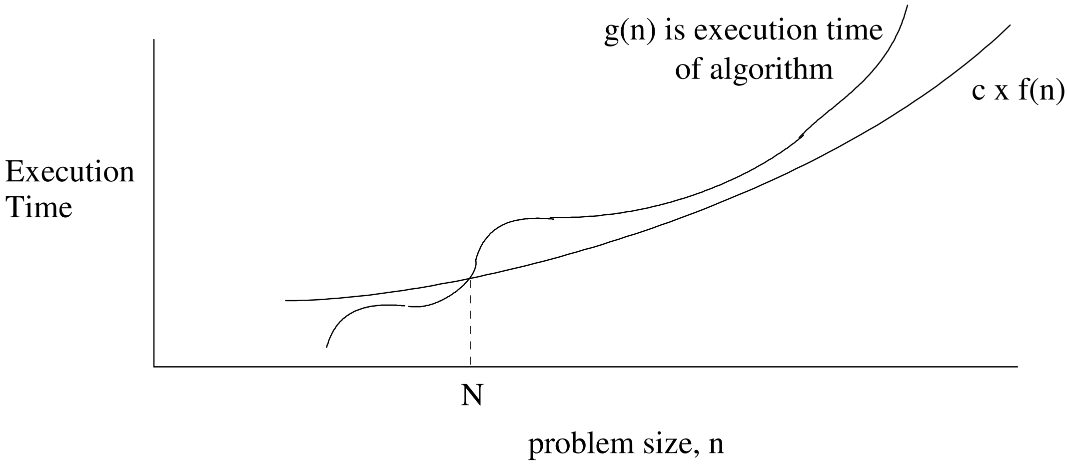
$100 + 10n \leq 110n$
$100 \leq 100n$
$1 \leq n$

Problem with big-oh:

If T(n) is $O(n)$, then it is also $O(n^2)$, $O(n^3)$, $O(n^4)$, $O(2^n)$, ... since these are also upper bounds. What we want is a "tight" upper bound, i.e., the "slowest" growing upper bound.

Omega Definition - asymptotic lower bound
For a given complexity function $f(n)$, $\Omega(f(n))$ is the set of complexity functions $g(n)$ for which there exists some positive real constant c and some nonnegative integer N such that for all $n \geq N$,

$$g(n) \geq c \times f(n).$$



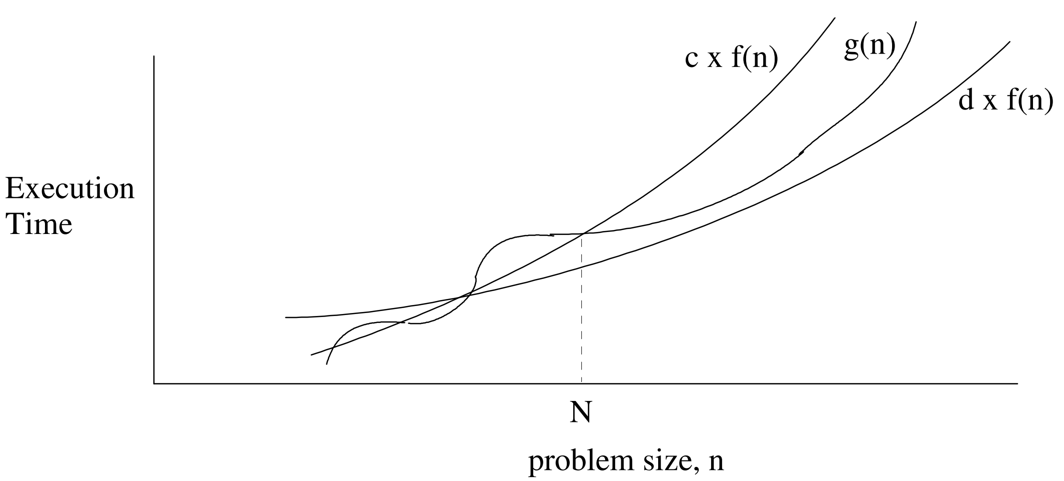
T(n) = $c_1 + c_2$ n = 100 + 10 n is Ω(n).

"Proof": We need to find a c and N so that the definition is satisfied, i.e.,
100 + 10 n $\geq$ c n for all n $\geq$ N.

0) What c and N will work?

Theta Definition - asymptotic upper and lower bound, i.e., a "tight" bound or "best" big-oh
For a given complexity function $f(n)$, $\theta(f(n))$ is the set of complexity functions $g(n)$ for which there exists some positive real constants c and d and some nonnegative integer N such that for all $n \geq N$,

$$c \times f(n) \leq g(n) \leq d \times f(n).$$



Since T(n) = $c_1 + c_2$ n = 100 + 10 n is both O(n) and Ω(n), it is θ(n).

1) Suppose that you have an $\theta(n^5)$ algorithm that required 10 seconds to run on a problem size of 1000. How long would you expect the algorithm to run on a problem size of 10,000?

2) Analyze the below algorithm to determine an obvious big-oh notation **and** its theta notation, $\theta(\ )$.

```
i := n
while (i >= 1) do
    for j := 1 to i do
        for k := 1 to n do
            something that takes O(1)
        end for k
    end for j
    i := i / 2
end while
```

3) Analyze the below algorithm to determine an obvious big-oh notation **and** its theta notation, $\theta(\ )$.

```
i := n
while i > 0 do
    for j = 1 to n do
        k := 1
        while k < = i do
            something that takes O(1)
            k := k * 2
        end while
    end for
    i := i / 2
end while
```

```
void seqsearch ( int n,
                 const keytype S[ ],
                 keytype x,
                 index & location ) {
  location = 1;
  while (location <= n && S[location] != x)
    location++;
  if (location > n)
    location = 0;
}
```

4) For sequential search, what is the best-case time complexity *B* (n)?

5) For sequential search, what is the worst-case time complexity *W* (n)?

6) If the probability of a successful sequential search is *p*, then what is the probability on an unsuccessful search?

7) If the probability of a successful sequential search is *p*, then what is the probability of finding the target value at a specific index in the array?

# compares:	1	2	3	. . .	n
probability:					

Write a summation for the average number of comparisons.

8) What is the average-case time complexity, *A* (n)?