Problem 1: Linear and Binary Searches.

Mathematical Analysis of Linear Search:

```
for(int indx = 0; indx < n; indx++) {
    if(val == a[indx]) return indx;
}

return -1;
```

O → represents operations, which equate to clock cycles.

O_B - Operations before loop.

O_i - Operations during loop.

PO_s - operations for our search.
(These operations only happen if a value is found, so there's a probability P.)

O_A - operations after loop.

Then: $O_B + \sum_{i=0}^{n-1} (O_i + PO_s) + O_A$

$* \sum_{i=m}^{n} 1 = (n-m+1)$. Let $O_{is} = O_i + PO_s$.

$\Rightarrow O_B + n(O_{is}) + O_A \qquad \left(\sum_{i=0}^{n-1} 1 = n-1-0+1 = n.\right)$

$= O_{is}n + O_B + O_A$.

$\Rightarrow f(n) =$ 1st order polynomial.

$f(n) = O_{is}n + (O_B + O_A)$

$= c'n + c$,

where $c' = O_{is} = O_i + PO_s$.

$c = O_B + O_A$.

$\therefore$ Linear search $= O(n)$ for all $n > 0$

where $c' > O_{is}$.

Mathematical Analysis of Binary Search:

```
int lowEnd = 0;
int highEnd = n-1;
do {
      int middle = (highEnd + lowEnd)/2;
      if (val == a[middle]) return middle;
      else if (val > a[middle]) lowEnd = middle+1;
       else highEnd = middle -1;
} while (lowEnd <= highEnd);
return -1;
```

In a binary search, we essentially have a halfing problem. With each successive operation, we cut our problem in half. we do so until we've found our value.

visualizing it:

$$
\begin{array}{ccccl}
 & & n & & \\
 & n/2 & & n/2 & (n/2) \\
n/4 & n/4 & & n/4 \quad n/4 & (n/2^2) \\
 & & \vdots & & (n/2^k)
\end{array}
$$

For example: if $n = 8$.

$8 \to 4 \to 2 \to \underline{1}$. we can half it 3 times.

or if $n = 16$.

$16 \to 8 \to 4 \to 2 \to \underline{\underline{1}}$. we can half it 4 times.

⇒ Eventually, we hit 1 when we continue to half our problem. Thus, we can make the observation that for some integer k,

$$\frac{n}{2^k} = 1.$$

∘ where k = the # of times we can half our problem.

(we're essentially halving n, k times.)

Then: $\frac{n}{2^k} = 1$, so $n = 2^k$.

Solving for k:

$$\log n = \log 2^k$$
$$= k \log_2 2^1$$

* $k = \log n$.

we can conclude from this that the most we can half our problem is k times. This k is related to n by logn. Lastly, our loop runs for at most k iterations.

$$\underbrace{O_B}_{\text{Before loop.}} + \underbrace{\Sigma O_{BS}}_{\text{During loop.}} + \underbrace{O_A}_{\text{After loop.}} = O_B + O_A + O(\log n)$$

O - represents operations.

$\therefore$ Binary Search $= O(\log n)$ for all $n > 0$
where c' in $c' \log n > O_{BS}$.

Given that Linear search $= O(n)$
and Binary Search $= O(\log n)$,
Binary search is more efficient.