

Problem 1: Linear and Binary Searches.

Mathematical Analysis of Linear Search:

```
for (int index = 0; index < n; index++) {  
    if (val == a[index]) return index;  
}  
return -1;
```

$O \rightarrow$ represents operations, which equate to clock cycles.

O_B - Operations before loop.

O_i - Operations during loop.

PD_i - operations for our search.

(These operations only happen if a value is found, so there's a probability P .)

O_A - operations after loop.

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

$$* \sum_{i=0}^{n-1} 1 = (n - 0 + 1). \text{ Let } O_{is} = O_i + PD_i.$$

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

$\Rightarrow P(n) = 1^{st} \text{ order polynomial.}$

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

$$= C'n + C,$$

$$\text{where } C' = O_{is} = O_i + PD_i.$$

$$C = O_B + O_A.$$

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

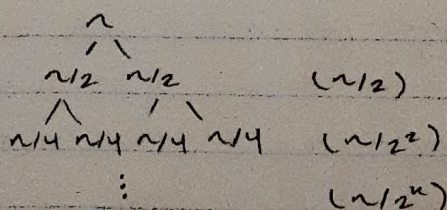
where $C' > 0$.

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 halving problem. With each successive operation, we cut our problem in half. We do so until we've found our value.

visualizing it:



For example: if $n = 8$.

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

or if $n = 16$.

$16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow \underline{1}$. 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 ,

$$\frac{n}{2^k} = 1 \quad \text{where } k = \text{the \# of times we can half our problem.}$$

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

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

Solving for k :

$$\begin{aligned}\log n &= \log 2^k \\ &= k \log_2 2^1\end{aligned}$$

$$* k = \log n.$$

we can conclude from this that the most we can half our problem is k times.

this k is related to n by $\log n$. Lastly, our loop runs for at most k iterations.

$$\begin{array}{ccccc} O_B + \sum O_{BS} + O_A & = & O_B + O_A + O(\log n) \\ \uparrow & \uparrow & \uparrow & & \\ \text{Before} & \text{During} & \text{After} & & \\ \text{loop.} & \text{loop.} & \text{loop.} & & \end{array}$$

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.