

# Find-The-Number

## 1 Find-The-Number With Comps

Consider the following two-person game, which we call *Find-The-Number with Comps*. Player  $A$  (for answerer) has a number  $x$  between 1 and 1000. Player  $Q$  (for questioner) asks questions of the form “ $x \leq y$ ?” and  $A$  responds with “YES” or “NO”. They keep doing this until  $Q$  determines  $x$ .

Lets see what might happen. Lets say  $Q$  first ask  $x \leq 600$ ? If  $A$  say YES then  $Q$  knows  $x \in [1, 600]$ , so its down to 600 numbers. If  $A$  say NO then  $x \in [601, 1000]$ , so its down to 400 numbers. So  $Q$  might luck out and get down to 400 numbers BUT  $Q$  could get stuck with 600 numbers.

Lets say  $Q$ 's first question is “ $x \leq 500$ ?” If  $A$  says YES then  $Q$  knows  $x \in [1, 500]$ , so its down to 500 numbers. If  $Q$  says NO then  $A$  knows  $x \in [501, 1000]$ , so its down to 500 numbers. So  $Q$  never does as well as 400 numbers, but never does as badly as 600 numbers.

KEY:  $Q$  can always cut the interval in half by asking a question in the middle of the interval.

Lets see how the worst case of this may go: Lets use a smaller number for exposition. Say  $x \in [1, 100]$ .

Initially  $x \in [1, 100]$ .

$Q$ : is  $x \leq 50$ ?

$A$ : YES so  $x \in [1, 50]$  (if  $A$  said NO then  $x \in [51, 100]$ , same size)

$Q$ : is  $x \leq 25$ ?

$A$ : YES so  $x \in [1, 25]$  (if  $A$  said NO then  $x \in [26, 50]$ , same size)

$Q$ : is  $x \leq 12$ ?

$A$ : NO so  $x \in [13, 25]$  (if  $A$  said YES then  $x \in [1, 12]$ , one less in size)

$Q$ : is  $x \leq 19$ ?

$A$ : YES so  $x \in [13, 19]$  (if  $A$  said NO then  $x \in [20, 25]$ , one less in size)

$Q$ : is  $x \leq 16$ ?

$A$ : YES so  $x \in [13, 16]$  (if  $A$  said NO then  $x \in [17, 19]$ , one less in size)

$Q$ : is  $x \leq 14$ ?

$A$ : YES so  $x \in [13, 14]$  (if  $A$  said NO then  $x \in [15, 16]$ , the same size)

$Q$ : is  $x \leq 13$ ?

$A$ : YES so  $x = 13$  (if  $A$  said NO then  $x = 14$ )

This took 7 comparisons.

The KEY here is that  $Q$  does not give  $A$  a chance to make the interval too big. In fact  $Q$  makes it so that  $A$ 's answer is almost irrelevant. Both intervals  $A$  can leave us with are roughly the same size. By contrast, if the first comparison was  $x \leq 10$ , then  $A$  would say NO leaving  $Q$  with  $x \in [11, 100]$ .

**Theorem 1.1** *Let  $A$  and  $Q$  play the game on  $[1, n]$ . If  $Q$  always guesses a number as close to the midpoint as possible then the game will terminate in at most  $\lceil \log_2 n \rceil$  steps.*

**Proof:**

We SKETCH the proof.

Every question will cut the interval in half. Hence at the end of the  $i$ th question the interval that the number is in is roughly  $n/2^i$ . After  $i = \lceil \log_2 n \rceil$  steps, the interval is size 1 hence the number is known. ■

Can we do better than  $\log_2 n$ ? We will show the answer is NO. The key to this and later proofs is to represent what we know at any given stage.

**Theorem 1.2** *Let  $A$  and  $Q$  play the game on  $[1, n]$ . No matter how clever  $Q$  is,  $A$  can answer the questions so that  $A$  is forced to take  $\lceil \log_2 n \rceil$  steps.*

**Proof:** Initially the set of possible answers is  $S_0 = \{1, \dots, n\}$ . (The subscript 0 means that 0 comparisons have been asked.) Note that  $|S_0| = n$ . This will be an *adversary argument* which means that we simulate  $A$  supplying answers to force the number of questions to be large.

Let " $x \leq x_0$ ?" be the first question. If the answer is YES then  $Q$  knows  $x \in \{1, \dots, x_0\}$ . If the answer is NO then  $A$  knows  $x \in \{x_0 + 1, \dots, n\}$ . One of these sets will have  $\geq n/2$  elements. Have  $A$  answer the question so that the set with  $n/2$  elements in it is the one that has  $x$  in it. Call this set  $S_1$ . We know that  $|S_1| \geq n/2$  and  $x \in S_1$ .

Keep doing this so that after  $i$  questions there is a set  $S_i$  such that  $x \in S_i$  and  $|S_i| \geq n/2^i$ .

If you ask LESS THAN  $\lceil \log_2 n \rceil$  questions then  $|S_i| > 1$  so you cannot know the answer. Hence you must ask at least  $\lceil \log_2 n \rceil$  questions. ■

## 2 Find-The-number With Parallel Comps

What if  $Q$  can ask TWO Comparisons at a time and wants to minimize the number-of-rounds of comparisons?

Look at the interval  $[1, 1000]$ .

In the last section  $Q$  HALVED the interval at every turn. Can  $Q$  do better?

$Q$  ask “ $x \leq 333?$ ” and “ $x \leq 666?$ ”

1. If the answers are “ $x \leq 333$ ” and “ $x \leq 666$ ” then we get  $x \in [1, 333]$ . Length 333.
2. If the answers are “ $x \leq 333$ ” and “ $x > 666$ ” then we say LIAR LIAR, PANTS ON FIRE since there is no number  $x$  such that both  $x \leq 333$  and  $x > 666$ .
3. If the answers are “ $x > 333$ ” and “ $x \leq 666$ ” then we get  $x \in [334, 666]$ . Length 332.
4. If the answers are “ $x > 333$ ” and “ $x > 666$ ” then we get  $x \in [667, 1000]$ . Length 333.

The interval is cut in thirds. We can keep doing this.

**Theorem 2.1** *Let  $A$  and  $Q$  play the game on  $[1, n]$ . If  $Q$  can ask 2 comparisons at a time then by making them as close to the  $n/3$  and  $2n/3$  points as possible as possible then the game will terminate in at most  $\lceil \log_3 n \rceil$  rounds.*

**Exercise 1** Show that if you ask 2 comparison questions at a time then  $\log_3 n$  questions are REQUIRED.

**Exercise 2** What happens when you ask 3 questions at a time?  $p$  questions at a time? Upper and lower bounds.

### 3 Find-the-Number With Boolean Queries

In the last section we noted that one of the possibilities,  $x < 333$  and  $x > 666$ , could not happen. This is unavoidable if you ask COMPARISONS. What if you ask any YES-NO questions at all? Can you do better than  $\lceil \log_3 n \rceil$  ?

First express the number in base 2. Let  $(n)_b$  denote  $n$  in base  $b$ .

$$(1000)_{10} = (111110100)_2.$$

Note that it has 9 places.

Q: Is the first bit 1? Is the second bit 1?

Whatever  $A$  answers you KNOW the first two bits.

Keep doing this and you can find the answer out in 5 questions.

**Theorem 3.1** *Let  $A$  and  $Q$  play the game on  $[1, n]$ . If  $Q$  can ask 2 questions at a time then by asking questions about the bits then the game will terminate in at most  $\lceil \log_2 n/2 \rceil$  steps.*

**Proof:**

Every number between 1 and  $n$  can be expressed in  $\lceil \log_2 n \rceil$  bits. Let  $k = \lceil \log_2 n \rceil$ . Let  $x = b_{k-1}b_{k-2} \cdots b_0$ . We do not know the bits  $b_0, \dots, b_{k-1}$ .

After asking the following questions and getting the answers,  $Q$  knows the number.

Q: Does  $b_0 = 0$ ? Does  $b_1 = 0$ ?

Q: Does  $b_2 = 0$ ? Does  $b_3 = 0$ ?

Q: Does  $b_4 = 0$ ? Does  $b_5 = 0$ ?

⋮

Q: Does  $b_{k-1} = 0$ ? Does  $b_k = 0$ ?

(This assume that  $k$  is even. If  $k$  is odd then the last question is just one question.)

The answers to these questions tell  $Q$  all of the bits, and hence the number  $x$ . ■

Is this better than using two comparisons at a time?

We need to know if

$$\left\lceil \frac{\log_2 n}{2} \right\rceil < \lceil \log_3 n \rceil.$$

It is easy to show that  $\log_x y = \frac{\log_b x}{\log_b y}$  where  $b$  is any base. Hence

$$\lceil \log_3 n \rceil = \left\lceil \frac{\log_2 n}{\log_2 3} \right\rceil.$$

So we want to know if

$$\left\lceil \frac{\log_2 n}{2} \right\rceil < \left\lceil \frac{\log_2 n}{\log_2 3} \right\rceil.$$

$\frac{1}{\log_2 3}$  is roughly 0.63. For  $n$  large enough (bigger than 10 certainly works) this inequality holds.

**Exercise 3** What if  $Q$  asks  $p$  Boolean questions at a time. How many rounds does  $Q$  need to find the number?

Can you do better than  $\frac{\log_2 n}{2}$ . We explore this in the next section.

## 4 Lower Bounds with Boolean Queries

Earlier we showed that if you use COMPARISONS then the problem requires  $\log_2 n$  comparisons. What if you asked ANY Boolean queries. Can you do better than  $\log_2 n$ ?

But to even state this requires more care. We need to use Decision trees. Note the following:

1. There are at least  $n$  leaves since there are  $n$  answers  $1, 2, \dots, n$ .
2. Since the questions are Boolean every internal node has two children.

If a Decision tree has depth 1 then it has  $2^1 = 2$  leaves.

If a Decision tree has depth 2 then it has  $2^2 = 4$  leaves.

etc.

**Lemma 4.1** *If a Decision tree has depth  $d$  then it has  $2^d$  leaves.*

**Exercise 4** Proof Lemma 4.1.

**Theorem 4.2** *Let  $A$  and  $Q$  play the game on  $[1, n]$  where  $Q$  can ask any Boolean queries whatsoever. No matter how clever  $Q$  is,  $A$  can answer the questions so that  $A$  is forced to take  $\lceil \log_2 n \rceil$  steps.*

**Proof:** Look at an algorithm by looking at the decision tree that represents it. Assume it has depth  $d$ . We want to find a lower bound on  $d$ .

The tree has at least  $n$  leaves, one for each answer. The tree has depth  $\leq 2^d$  by Lemma 4.1. Hence we need

$$2^d \geq n$$

$$d \geq \lceil \log_2 n \rceil.$$

Hence there is some path in the tree that takes at least  $\lceil \log_2 n \rceil$  questions.

■

Hence, even with general Boolean Queries, you need  $\log_2 n$  queries.

What about if we can ask 2 Boolean queries at a time?

**Theorem 4.3** *Let  $A$  and  $Q$  play the game on  $[1, n]$  where  $Q$  can ask rounds of 2 Boolean queries. No matter how clever  $Q$  is,  $A$  can answer the questions so that  $A$  is forced to take  $\lceil \frac{\log_2 n}{2} \rceil$  steps.*

**Proof:** Assume, by way of contradiction, that there was an algorithm that solved find-the-number in  $\leq \lceil \frac{\log_2 n}{2} \rceil - 1$  rounds of 2 Boolean Queries. By asking the questions in sequentially we would have an algorithm that took  $\leq 2(\lceil \frac{\log_2 n}{2} \rceil - 1) < \lceil \log_2 n \rceil$  Boolean Queries. This contradicts Theorem 4.2.

■

**Exercise 5** Look into upper and lower bounds for algorithms that use rounds of  $p$  Boolean Queries.

## 5 Find-The-Number with Boolean Queries but One Lie

Consider the following two-person game, which we call *Find-The-Number with Boolean Questions but one lie*. Player  $A$  (for answerer) has a number  $x$  between 1 and 1000. Player  $Q$  (for questioner) asks questions of the form “ $x \leq y?$ ” and  $A$  responds with “YES” or “NO”. But  $Q$  should beware! One of the answers that  $A$  gives is a LIE! They keep doing this until  $Q$  determines  $x$  (and is sure of the answer).

HOW MANY BOOLEAN QUERIES DOES IT TAKE YOU TO FIND THE NUMBER IN THE WORST CASE, KNOWING THAT  $A$  WILL LIE ONCE?

**Theorem 5.1** *Let  $A$  and  $Q$  play the game on  $[1, n]$ . If  $Q$  always guesses a number as close to the midpoint as possible, and asks it twice, then the game will terminate in at most  $2 \lceil \log_2 n \rceil + 1$  steps.*

**Exercise 6** Prove Theorem 5.1.

Can we do better than this?

Since  $A$  will be able to lie once, we'll allow  $Q$  to make Boolean Queries.

**Theorem 5.2** *Let  $A$  and  $Q$  play the game on  $[1, n]$ . Assume that  $Q$  can ask Boolean queries and that  $A$  can lie once. Then there is a strategy for  $Q$  that finds the number in  $\leq \log n + \log \log n + O(1)$  steps.*

**Proof:** The questions will be of the form  $x \in C$  where  $C$  is some set. For example, " $x \in \{1, 2, 3, \dots, 17\} \cup \{n - 10, n - 7, n\}$ ?" could be a question. Assume that the answer to this question was NO. It could still be the case that (say)  $x = 17$  since this could have been the question that was lied about. If the next question was  $x \in \{1, 3, 5, 17, 18, 19, 20, n - 7\}$  and the answer was NO then we know that  $x \notin \{1, 3, 17, n - 7\}$ . That is because these elements were TWICE declared to not be  $x$ .

We will keep track of two sets: (1) the set of all  $y$  that have NEVER been declared to NOT be  $x$ , and (2) the set of all  $y$  that have ONCE been declared to NOT be  $x$  (these could still be  $x$  since  $A$  can lie once). If an element has been declared NOT  $x$  twice or more, then it is no longer considered since it is not  $x$ .

Let  $S_0 = \{1, \dots, n\}$  and  $T_0 = \emptyset$ . Throughout the algorithm (1)  $S_i$  will be elements that, after  $i$  questions, may still be  $x$  and have NEVER been declared to NOT be  $x$ , (2)  $T_i$  will be elements that, after  $i$  questions, may still be  $x$  but have ONCE been declared to NOT be  $x$ . Since  $A$  can lie once, its possible that  $x \in T_i$ .

We pursue the first few steps of the algorithm, and one set of possible answers, as a thought experiment. We will return to doing it formally later.

Lets say  $S_0 = \{1, \dots, 100\}$ . I first ask "Is  $x \in \{1, \dots, 50\}$ ?" The answer is YES. Now

$S_1 = \{1, \dots, 50\}$  (these have never been declared NOT  $x$ )

$T_1 = \{51, \dots, 100\}$  (these have been declared NOT  $x$  only once)

I now ask “Is  $x \in \{1, \dots, 25\} \cup \{51, \dots, 75\}$ .” Note that I took half from each pile. The answer is NO. This means (1) all the  $x \in \{1, \dots, 25\}$ , which prior to this question were never declared to not be in  $x$ , have been so declared but just once, and (2) all the  $x \in \{51, \dots, 75\}$ , which prior to this question were declared once to not be  $x$ , have now been so declared a second time. Hence they are not  $x$  and are removed from the board.

$S_2 = \{26, \dots, 50\}$  (these have never been declared NOT  $x$ )

$T_2 = \{1, \dots, 25\} \cup \{76, \dots, 100\}$  (these have been declared NOT  $x$  only once)

I now ask “Is  $x \in \{26, \dots, 39\} \cup \{1, \dots, 25\}$ ”

The answer is YES. This means that  $x$  has been declared (for the first time) to not be in  $\{40, \dots, 50\}$  and for the second time to not be in  $\{76, \dots, 100\}$ . Hence

$S_3 = \{26, \dots, 39\}$  (these have never been declared NOT  $x$ )

$T_3 = \{40, \dots, 50\} \cup \{1, \dots, 25\}$  (these have been declared NOT  $x$  only once)

I now ask “ $x \in \{26, \dots, 31\} \cup \{1, \dots, 17\}$ ?” (I pick this since it cuts both piles in half.) The answer is NO.

$S_4 = \{32, \dots, 39\}$  (these have never been declared NOT  $x$ )

$T_4 = \{40, \dots, 50\} \cup \{18, \dots, 25\}$  (these have been declared NOT  $x$  only once)

I now ask “ $x \in \{32, \dots, 35\} \cup \{40, \dots, 49\}$ ?” The answer is YES

$S_4 = \{32, \dots, 39\}$  (these have never been declared NOT  $x$ )

$T_4 = \{40, \dots, 50\} \cup \{18, \dots, 25\}$  (these have been declared NOT  $x$  only once)

FINISH AFTER PROOFREADING.

Let  $S_0 = S_{01} \cup S_{02}$ , two equal sized sets (nearly equal if  $n$  is odd).

$Q: x \in S_{01}$ ?

If the answer is YES then everything in  $S_{02}$  has been declared to not be  $x$  (once). Hence we would set

$S_1 = S_{01}$

$T_1 = S_{02}$ .

Now what do we ask? First set

$S_1 = S_{11} \cup S_{12}$  (two equal or near-equal sets)

$T_1 = T_{11} \cup T_{12}$  (two equal or near-equal sets)

$Q: x \in S_{11} \cup T_{11}$ ?

If the answer is NO then everything in  $S_1$  has been declared NOT  $x$  once, and everything in  $T_1$  has been declared NOT  $x$  twice. Hence

$$S_2 = S_1$$

$$T_2 = (T_1 \cup S_1) - T_1.$$

More generally, we will ask a question about a set formed from part of the  $S$  and part of the  $T$  and then shift down as appropriate.

How big is  $S_0, S_1, S_2, \dots$ . Note that

$|S_0| = n$ ,  $|S_1| = n/2$ ,  $|S_2| = n/4$ , etc. Hence in  $\lceil \log_2 n \rceil$  the  $S$  pile will have only one element in it.

We present the first phase of the algorithm.

ALGORITHM: Phase 1.

$$S_0 = \{1, \dots, n\}$$

$$T_0 = \emptyset.$$

$$k = \lceil \log_2 n \rceil$$

For  $i = 1$  to  $k$

$$S_{i-1} = S_{i-1}^1 \cup S_{i-1}^2 \text{ (near even divide)}$$

$$T_{i-1} = T_{i-1}^1 \cup T_{i-1}^2 \text{ (near even divide)}$$

$$Q: x \in S_{i-1}^1 \cup T_{i-1}^1$$

if answer is YES then

$$S_i = S_{i-1}^1$$

$$T_i = (T_{i-1}^1 \cup S_{i-1}^2) - T_{i-1}^2.$$

if answer is NO then

$$S_i = S_{i-1}^2$$

$$T_i = (T_{i-1}^2 \cup S_{i-1}^1) - T_{i-1}^1.$$

END OF PHASE 1.

When this part of the algorithm is completed  $S_i$  has at most 1 element  $y$  in it. Ask if  $x = y$  three times. If the majority say YES then  $x = y$  and we only used  $\lceil \log_2 n \rceil + 3$  queries. If the first answer is NO then put the element into  $T_i$ .

Now we have to look at the  $T$  pile. How big is it?

Let  $s_i = |S_i|$  and  $t_i = |T_i|$ . Note that for any  $i \leq k$ .

$$\begin{aligned} s_i &= \frac{1}{2}s_{i-1} \\ t_i &= \frac{1}{2}t_{i-1} + \frac{1}{2}s_{i-1} \end{aligned}$$

We first solve for  $s_i$ :

$$s_i = \frac{1}{2}s_{i-1} = \frac{1}{2^2}s_{i-2} = \frac{1}{2^3}s_{i-3} = \dots = \frac{1}{2^i}s_{i-i} = \frac{1}{2^i}s_0 = \frac{1}{2^i}n.$$

We can use this in our equation for  $t_i$ .

$$\begin{aligned}
t_i &= \frac{1}{2}t_{i-1} + \frac{1}{2}s_{i-1} \\
&= \frac{1}{2}t_{i-1} + n/2^i \\
&= \frac{1}{2}\left(\frac{1}{2}t_{i-2} + n/2^{i-1}\right) \\
&= \frac{1}{2^2}t_{i-2} + n/2^i + n/2^i \\
&= \frac{1}{2^3}t_{i-3} + n/2^i + n/2^i + n/2^i \\
&= \frac{1}{2^4}t_{i-4} + 4(n/2^i) \\
&= \vdots \\
&= \frac{1}{2^i}t_{i-i} + i(n/2^i) \\
&= \frac{1}{2^i}t_0 + i(n/2^i) \\
&= \frac{1}{2^i} \cdot 0 + i(n/2^i) \\
&= \frac{in}{2^i}
\end{aligned}$$

Our major concern is what happens when  $i = k = \lceil \log_2 n \rceil$ .

$$t_k = \frac{kn}{2^k} = \frac{n \lceil \log_2 n \rceil}{2^{\lceil \log_2 n \rceil}} = \lceil \log_2 n \rceil + O(1).$$

So after the first  $k = \lceil \log_2 n \rceil$  questions there are  $\lceil \log_2 n \rceil$  elements in the T-pile. Since there are no elements in the S-pile that game is now that of finding a number in a set of  $\lceil \log_2 n \rceil$  numbers by using Boolean Queries.

PHASE TWO: By Theorem 3.1 this can be done with  $\lceil \log \log n \rceil$  queries.  
 END OF PHASE TWO

Hence the entire algorithm takes  $\lceil \log n + \log \log n \rceil + O(1)$  queries. ■

**Exercise 7** Explore what happens when  $Q$  asks 1 Boolean queries at a time and  $A$  is allowed to lie about two question once in the entire game.

**Exercise 8** Explore what happens when  $Q$  asks 2 Boolean queries at a time and  $A$  is allowed to lie about one question once in the entire game.