

Complexity of Binary Search

```

start = 1; end = n;
while (start < end) {
    middle = (start + end) / 2;
    if s > amiddle then
        start = middle + 1;
    else end = middle - 1;
}
if (s == astart) location = start;
else location = 0;
    
```

Each pass through the body of the while loop takes constant (unit) time

Constant time

The question is: How many times will this while loop be executed?

Complexity of Binary Search

P Calculate the number of times the while loop will be executed

P Each pass through the while loop decreases the size of the range by at least half

P So the question is:

▶ Given an integer n , how many times does it need to be cut in half before it reaches, or goes below, 1?

P That is, which of the following will first be < 1 ?

▶ $n/2, n/4, n/8, n/16, \dots, n/2^k, \dots$

Complexity of Binary Search

P That is, which of the following will first be < 1 ?

▶ $n/2, n/4, n/8, n/16, \dots, n/2^k, \dots$

P We solve the equation: $n/2^k < 1$, and get $k > \log_2 n$

P So if we set $k = \lceil \log_2 n \rceil$, then we know that after that many iterations of the while loop, we will have found our item, or concluded that it was not in the list

Logarithmic Time Complexity is **Fast**

P Our analysis shows that binary search can be done in time proportional to the \log of the number of items in the list

P This is considered *very fast* when compared to linear or polynomial algorithms

P The table to the right compares the number of operations that need to be performed for algorithms of various time complexities

n	log n	n	n ²	2 ⁿ
1	0	1	1	2
2	1	2	4	4
5	3	5	25	32
10	4	10	100	1024
20	5	20	400	1048576
50	6	50	2500	1.1E+15
100	7	100	10000	1.3E+30
200	8	200	40000	1.6E+60
500	9	500	250000	err
1000	10	1000	1E+06	err
2000	11	2000	4E+06	err
5000	13	5000	3E+07	err

An Algorithm to Test one-to-one-ness

P TESTPAIRS:

P Suppose $f:A \rightarrow B$ is a function, and A and B are finite sets

P For each element a of A , compute $b = f(a)$, and then check, for each element $a' \in A$, $a' \neq a$, whether $f(a') = b$.

- ▶ In finite time we will check all pairs
- ▶ Each checking is a straightforward comparison
- ▶ If the answer is ever “yes,” then the function is not 1-1
- ▶ Otherwise the function is 1-1

P Aren't you wondering what the time complexity of this algorithm is?

Time Complexity of TESTPAIRS

P We have to consider $|A|$ elements

P For each of those, we need to compare its function value with that of $|A| - 1$ other elements

P For a total time of $|A|(|A| - 1)$ steps

P Which is $O(|A|^2)$

- ▶ This is considered a *quadratic algorithm* in the size of A
- ▶ Note that the size of B does not affect the speed of the algorithm

P Can we do better? Can you suggest another algorithm?

Another Algorithm to Test 1-1

P COLORRANGE:

P For each element a of A , color $f(a)$ blue to show that “we've been there already”

P If, during this process, we try to color an element blue which is *already* blue, then the function is not 1-1.

P But if that never happens, then the function is 1-1

P Again, I can see that you are wondering what the time complexity of this algorithm is...

Complexity of COLORRANGE

P We do something once for each element of A , and that something takes constant time

P In the “worst case,” that is, the case that makes our algorithm run the longest, we have to scan through all of A , which will take $|A|$ steps.

P So our algorithm will run in $O(|A|)$ steps.

- ▶ So our algorithm is linear in the size of A
- ▶ Note again that $|B|$ does not affect the running time
- ▶ Is this faster or slower than TESTPAIRS?