Сложность представленного алгоритма *Search Algorithm* – $O(\log n)$.

Доказательство (1 вариант).

Рассмотрим алгоритм. За каждый вызов функции выполняется 2 сравнения в лучшем и худшем случаях (поиск закончился или продолжился). Это постоянная величина. Если поиск продолжается, функция вызывает саму себя, в чем и заключается рекурсия. Количество рекурсивных k вызовов заранее неизвестно и зависит от входных данных.

В ходе рекурсии входные данные для каждого последующего вызова уменьшаются приблизительно вдвое (возможно округление в большую/меньшую сторону). Тогда количество входных данных для каждого вызова функции можно описать формулой

$$n_i = \frac{n}{2^i}, i = 0, 1, 2, \dots, k.$$

При последнем вызове функции ей передается единственный элемент, который либо окажется искомым, либо выяснится, что массив не содержит нужного элемента, т.е.

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

Вышесказанное можно представить в виде таблицы.

Вызов функции	0	1	…	$k-1$	k
Количество операций в ходе одного вызова	2	2	2	2	2
Входные данные	n	$n_1 = \frac{n}{2}$	$n_2 = \frac{n}{4}$	$n_{k-1} = \frac{n}{2^{k-1}}$	1

Сложность алгоритма поиска будет произведением количества операций в ходе одного вызова и скорости сходимости ряда входных данных к единице.

Оценим ряд сверху, заменив исходную формулу на следующую

$$n_i = \frac{n+1}{2^i}, i = 0, 1, 2, \dots, k.$$

Скорость сходимости или количество вызовов k можно найти, приравняв эту формулу на k-шаге к единице

$$n_k = \frac{n+1}{2^k} = 1,$$

$$\frac{n+1}{2^k} = 1,$$

$$n + 1 = 2^k,$$

$$\log_2(n+1) = k.$$

Тогда сложность алгоритма будет

$$O(2 \cdot \log_2(n+1)) \approx O(\log n).$$

Доказательство (2 вариант).

Основная теорема о рекуррентных оценках рассматривает функции, удовлетворяющие рекуррентному соотношению

$$F(n) = a \cdot F(n/b) + f(n),$$

где $a \geq 1, b \geq 1$ – некоторые целые константы; f – положительная функция ($f(n) \geq 0$ при целых $n \geq 0$).

Выражение в правой части может пониматься двумя способами – как $F(\lfloor n/b \rfloor) + f(n)$ и как $F(\lceil n/b \rceil) + f(n)$.

Если функция удовлетворяет этому соотношению, что для нее выполняются следующие условия:

1) Если существует $\varepsilon > 0$ такое, что $f = O(n^{\pi-\varepsilon})$, то $F = \Theta(n^{\pi})$.

2) Если $f = \Theta(n^{\pi})$, то $F = \Theta(n^{\pi} \cdot \log n)$.

3) Если существует $\varepsilon > 0$ такое, что $f = \Omega(n^{\pi+\varepsilon})$, и функция f обладает свойством (a, b) - регулярности, то $F = \Theta(f)$.

Примечание: $\pi = \log_b a$.

Для алгоритма бинарного поиска данных в упорядоченном массиве из n элементов (аналогичен нашему алгоритму) можно показать, что для функции F (числа операций сравнения, необходимых для поиска) выполняется рекуррентное соотношение

$$F(n) = F(n/2) + \Theta(1),$$

где $a = 1$, $b = 2$, $\pi = \log_2 1 = 0$, $f = \Theta(1)$.

Из утверждения 2 следует, что

$$F = \Theta(\log n) \rightarrow F = O(\log n).$$