The term Exponential often signifies fast expansion, and mathematically it means rising in powers.

For example:

Exponential growth of 2: 2^0, 2^1, 2^2, 2^3, 2^4 and so on => (1, 2, 4, 8, 16,.....)

Exponential growth of 3: 3^0, 3^1, 3^2, 3^3, 3^4 and so on => (1, 3, 9, 27, 81,....)

We utilize the same produced integers (powers of 2) to hop indexes in an array and come closer to the index of the key.

In this technique, eventually, we depend on Binary Search for searching, but before that, we determine a range in which the element we want to search could be present.

To finish this range we follow a particular algorithm, let's take a brief look at the overview of the functioning of this algorithm with the assistance of an example.

Understanding Exponential Search

Consider a sorted Array: **7 12 34 57 65 74 81 88 89 93 100**

Element to search: 93

We will start by comparing the first member of the array with the key.

But Array[0]=7, which is not equivalent to 93.

Now we will choose a variable whose value will rise exponentially, thus the term, exponential search. int i= 1 (since 2^0=1)

Now we shall test whether Array[i] is less than or equal to the key.

If it is, then we shall carry on growing the amount exponentially.

i=i*2

This will yield values in powers of 2. (2, 4, 8, 16, 32...) We shall keep on raising the value of I until the condition Array[i]<=key is fulfilled.

So in the preceding example, the value of I will reach 8 (in the real code it will reach 16, then we will split it by 2) and the sub-array after this index will be picked (containing the index of I and then the binary search will be applied to this range.

Exponential Search Algorithm

There are basically two phases involved in conducting an exponential search:-

- Finding the range in which the key could sit
- Applying binary search in this range

Steps

- Start with value i=1
- Check for a condition I < n and Array[i]<=key, where n is the number of items in the array and key is the element being sought
- Increment value of I in powers of 2, that is, i=i*2
- Keep on incrementing the value of I until the condition is met
- Apply binary on the range i/2 to the end of Array - binarySearch(Array, i/2, min(i,n-1))

Example

Consider the array:- 1 3 5 7 9 11 13 15 17 19

Element to search: 19

- We will start by comparing Array[0] element to the key, which in our instance will yield false.
- I will be initialized to 1

Now we will carry on incrementing the value of I until it is less than or equal to the key.

Note that the value of I is now 16, and the index 16 is out of range in this instance, thus to recover the previous value of I we will divide it by 2, then run a binary search using the index as low as i/2.

Now we call the binary search technique.

binarySearch(Array, i/2, min(i, n-1), key)

```
low = 8, high = 10


mid = (low + high)/2


  = 18/2


  = 9
```

Array[9]=19, which is the required element.

Now let's have a look at the Java code for the same.

Code

```
public class Searching {


    boolean exponentialSearch(int arr[], int key) {


        int lengthOfArray = arr.length;


        if (arr[0] == key) { // Checking whether first element
is the key

            return true;

        }
```

```
        // Finding the range in which the element might be
present



        int i = 1;



        while (i < lengthOfArray && arr[i] <= key) {

            i = i * 2; // Exponentially increasing value of i.



        }



        return binarySearch(arr, i / 2, Math.min(i,
lengthOfArray - 1), key); // calling binary search method on
the sub-array



    }



    boolean binarySearch(int arr[], int low, int high, int
key) {



        int mid; // to store middle element



        while (low <= high) {



            mid = (low + high) / 2; // we can also do mid =
low+(high-low)/2 to avoid overflow in some cases



            if (arr[mid] == key) {
```

```
                return true;
            } else if (arr[mid] < key) {
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }
        return false;
    }

    // Driver Code

    public static void main(String args[]) {

        Searching search = new Searching();

        int arr[] = {
            1,
            3,
            4,
            6,
            8,
            13,
            15,
```

```
            24

        };


        if (search.exponentialSearch(arr, 15)) {
            System.out.println("Element found !");
        } else {
            System.out.println("Element not found :( ");
        }


    }


}
```

Copy

Performance

Case	Runtime
Best	O(1)
Average	O(log i)
Worst	O(log i)
Auxiliary Space	O(1)

Here i is the index of the key.

Exponential search is beneficial in circumstances when the arrays are unbounded or of indefinite size and may converge to the answer significantly quicker than binary search in these cases.