Exponential Search

Date.....

Exponential search is an algorithm used for searching in sorted arrays like Binary Search.

↓

Exponential search is also known as "Doubling search" or "Galloping search"

↓

The idea of "Exponential search" is to determine a range that the target value resides in & perform a Binary search within that range.

↓

In Binary search, we directly set start at 0^{th} index & end at $n-1^{th}$ index, then find the element.

↓

In Exponential search, first we found a range in between which the searched value exists. Then on that particular range, the binary search techniaue is applied.

Date.....

For exponential search, the list should be in sorted order.
First, the subarray size is taken as 1, then the last element of that particular range is compared with the search value, then the subarray size is taken as 2, then 4 and so on.
Parallely we check the element should be less than the search element.
when the last element of the sub-array is greater than the searched element then the process is stopped.
Now, we can say that the searched element is present between i/2 and i (if the last index of the subarray greater than the searched element is i)
The range is i/2 and i because in the previous iteration the last element was not greater than the searched element.

Example :-

0	1	2	3	4
15	20	35	47	59

x = 47

First we compare the arr[0] with 'x'
If arr[0] == x, then we return the element.
Then the range is set to 1, check arr[1], i.e., 20 and compare with 'x'.

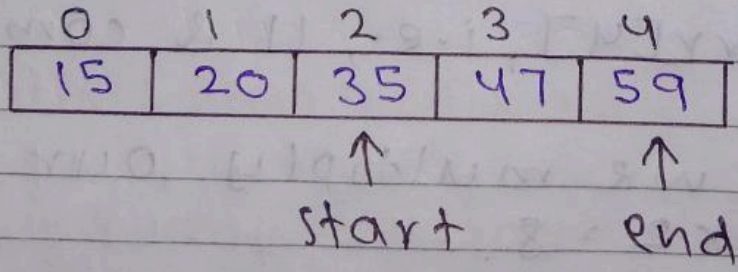
As the element is less than 'x', then the next range will be set to 1 x 2 = 2

index 1 is set to double, means index = index * 2.

Now check arr[2], i.e., 35 and compa-re with 'x'.
As 35 is less than 47, then the bound is set to index = index * 2, i.e., index = 2 * 2, i.e., 4.

Now check arr[4], i.e., 59 and compa-re with 'x'.
Now, 59 is not less than 47. So, the process will be stopped. Means, the binary search will be performed between i/2 and i index.

The "lower index is 2" and the "upper index is 4"




```
mid = (start + end) / 2;
mid = (2 + 4) / 2;
mid = 6 / 2;
mid = 3;
```


Teacher's Sign .....

As, arr[mid] == x
values are matched then the position of the element will be printed.

Example :-

0	1	2	3	4	5	6	7	8	9
3	4	5	6	11	13	14	15	56	70

x = 13

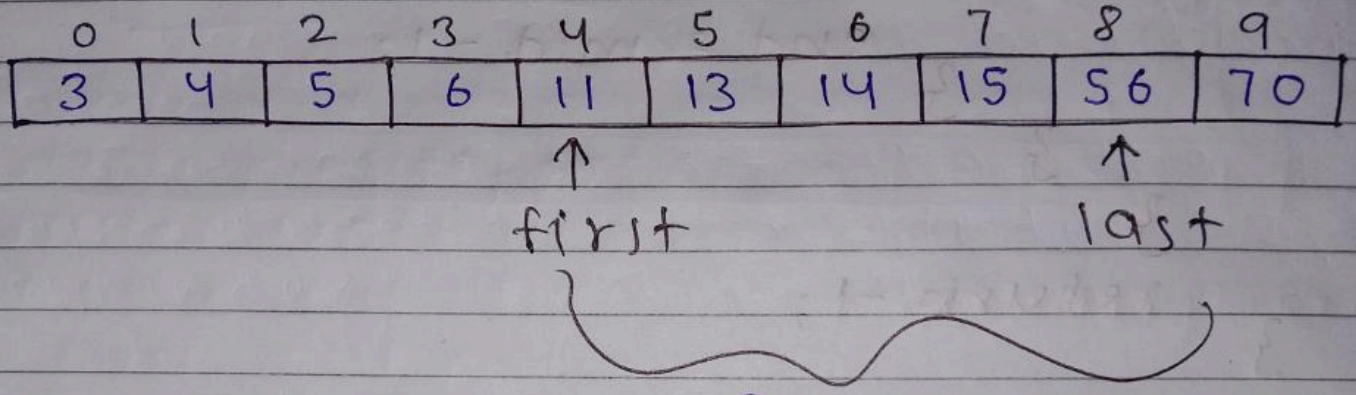
First, we check arr[0] with 'x'.
If arr[0] == x then return 0th index.
Then start the range from 1st index, check arr[1], i.e., 4 & compare with 'x'.
4 < 13, then we multiply our index by 2, i.e., 1 x 2 = 2.

Now check arr[2], i.e., 5. & compare with 'x'.
5 < 13, then we multiply our index by 2, i.e., 2 x 2 = 4.

Now check arr[4], i.e., 11 & compare with 'x'
11 < 13, then we multiply our index by 2, i.e., 4 x 2 = 8.

Now check arr[8], i.e., 56 & compare with 'x'.
56 > 13. So, we stop further comparisons.

Our current i^{th} element is greater than 13, and last $i/2^{th}$ element is less than 13.
It means our range lies from i/2 to i.
Now we apply Binary search on this range only.



code :-

```
#include<bits/stdc++.h>
using namespace std;
int binsearch(int arr[], int start,
                    int end, int x)
{
  while(start <= end)
  {
    int mid = (start + end)/2;
    if(x == arr[mid])
    {
      return mid;
    }
```

```
        else
        {
            if(x > arr[mid])
            {
                start = mid + 1;
            }
            else
            {
                end = mid - 1;
            }
        }
    }
    return -1;
}
int exposearch(int arr[], int n,
                              int x)
{
    if(arr[0] == x)
    {
        return 0;
    }
    int i = 1;
    while(i < n && arr[i] <= x)
    {
        i *= 2;
    }
    return binsearch(arr, i/2,
                min(i, n-1), x);
}
int main()
{
    int arr[] = {3, 4, 5, 6, 11, 13, 14, 15,
                56, 70};
```

Date.....

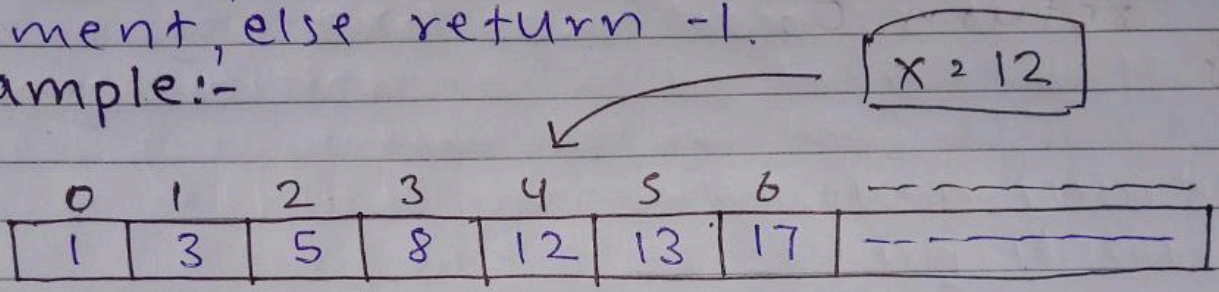
```
int n = sizeof(arr) / sizeof(arr[0]);
int x = 70;

int ans = exposearch(arr, n, x);
cout << "\nElement present at " <<
    ans << " index";
return 0;
}
```

unbounded search

Date.....

In this, we've given an unbounded array & this array is sorted. we have to search the given element in the array.
If found, return the index of the element, else return -1.

Example:-



output:- 4

we can solve this using the previous Exponential search approach.
In this, we increase the interval size by an exponential order of 2.
It helps us to track the upper bound.

code:-

```
int unboundSearch(int arr[], int x)
{
    int i = 0;
    int j = 1;
    while(arr[j] < x)
    {
        i = j;
        j = j * 2;
    }
    return binsearch(arr, i, j, x);
}
```

Teacher's Sign .....