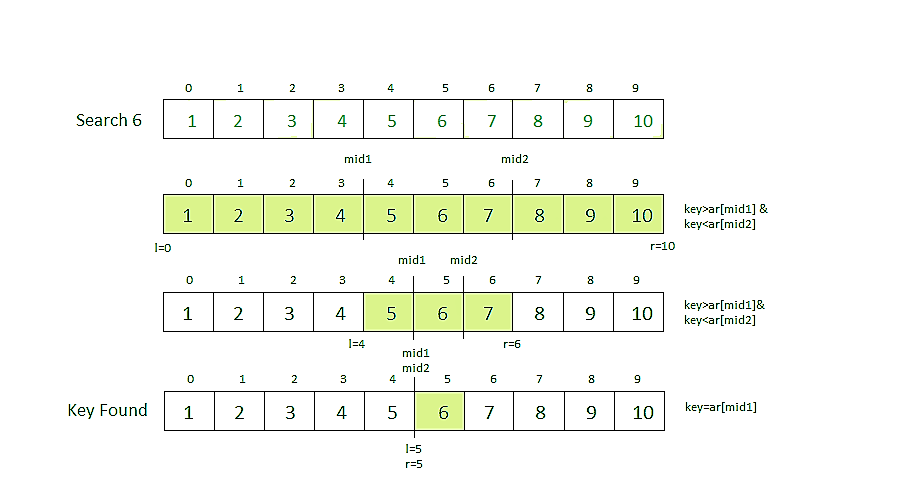
During this lab, use the skeleton of the program provided and fill in the code to test your implementation. The code to measure the execution time of each algorithm to show the efficiency of your code is already provided.

1. **Ternary search** is a search algorithm that is used to find the position of a target value within a sorted array. It operates on the principle of dividing the array into **three** **parts** instead of two, as in **binary search**. The basic idea is to narrow down the search space by comparing the target value with elements at two points that divide the array into **three equal parts**.

mid1 = l + (r-l)/3

mid2 = r – (r-l)/3



1. **Exponential search** involves two steps:

* Find range where element is present
* Do Binary Search in above found range.

How to find the range where element may be present?

The idea is to start with subarray size 1, compare its last element with x, then try size 2, then 4 and so on until last element of a subarray is not greater than x.

Once we find an index i (after repeated doubling of i), we know that the element must be present between i/2 and i (Why i/2? because we could not find a greater value in previous iteration).

1. **Fibonacci Search** is a comparison-based technique that uses Fibonacci numbers to search an element in a sorted array.

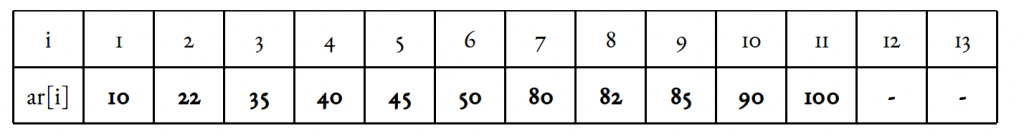
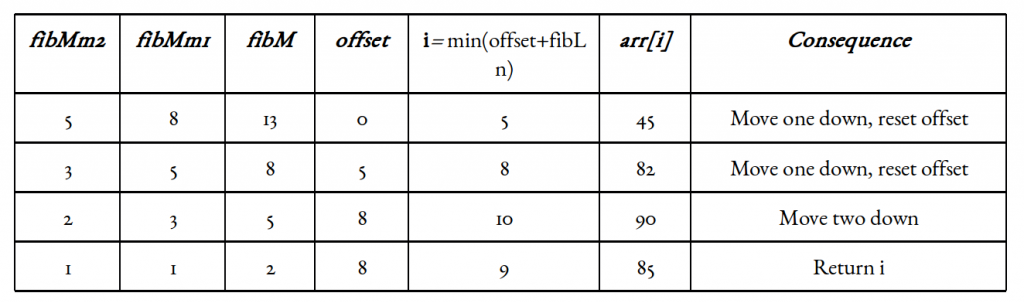
**Recall:** Fibonacci Numbers are recursively defined as F(n) = F(n-1) + F(n-2), F(0) = 0, F(1) = 1.

First few Fibonacci Numbers are 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …

**Idea:**  Let arr[0..n-1] be the input array and the element to be searched be x.

1. Find the smallest Fibonacci Number greater than or equal to n. Let this number be fibM [m’th Fibonacci Number]. Let the two Fibonacci numbers preceding it be fibMm1 [(m-1)’th Fibonacci Number] and fibMm2 [(m-2)’th Fibonacci Number].
2. While the array has elements to be inspected:
   1. Compare x with the last element of the range covered by fibMm2
   2. **If** x matches, return index
   3. **Else If** x is less than the element, move the three Fibonacci variables two Fibonacci down, indicating elimination of approximately rear two-third of the remaining array.
   4. **Else** x is greater than the element, move the three Fibonacci variables one Fibonacci down. Reset offset to index. Together these indicate the elimination of approximately front one-third of the remaining array.
3. Since there might be a single element remaining for comparison, check if fibMm1 is 1. If Yes, compare x with that remaining element. If match, return index.

Here is an example: Search for x = 85, length array = 11



**Step 1:** The size of the input array is 10. The smallest Fibonacci number greater than 10 is 13.

Fm = 13, Fm-1 = 8, Fm-2 = 5, initialize offset = -1

**Step 2:** Compare x with the element at index = minimum (offset + Fm-2, n – 1) = minimum (-1 + 5, 9) = 4.

The fourth element in the array is 20, which is not a match and is less than the key element.

**Step 3:** In the second iteration, update the offset value and the Fibonacci numbers.

Since the key is greater, the offset value will become the index of the element, i.e. 4. Fibonacci numbers are updated as Fm = 8, Fm-1 = 5, Fm-2 = 3.

Now, compare it with the element at index = minimum (offset + Fm-2, n – 1) = 7.

Element at the 7th index of the array is 43, which is not a match and is also lesser than the key.

**Step 4:** We discard the elements after the 7th index, so n = 7 and offset value remains 4.

Fibonacci numbers are pushed two steps backward, i.e. Fm = 3, Fm-1 = 2, Fm-2 = 1.

Now, compare it with the element at index = minimum (offset + Fm-2, n – 1) = minimum (4 + 1, 6) = minimum (5, 7) = 5.

The element at index 5 in the array is 24, which is our key element. 5th index is returned as the output for this example array.