Data Structure and Algorithms

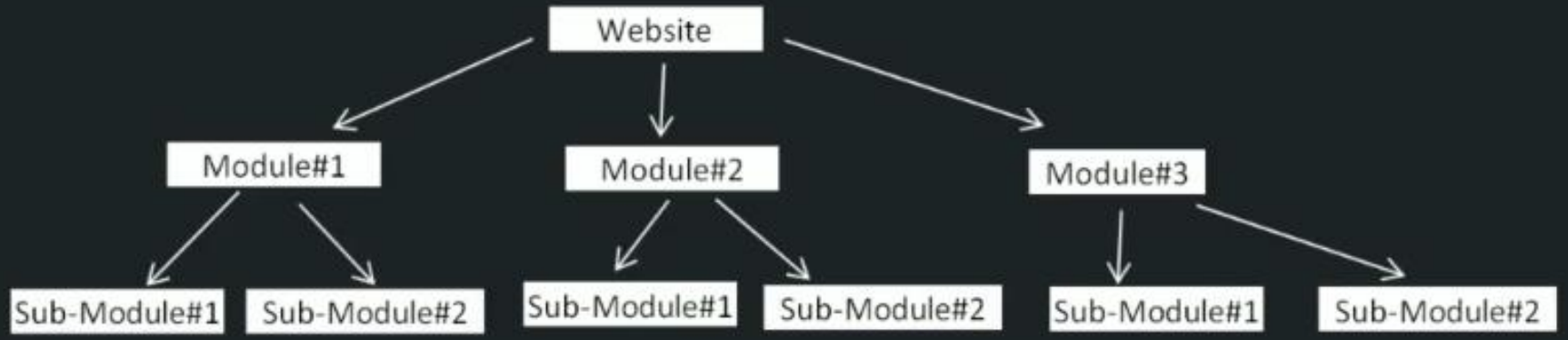
Session-18

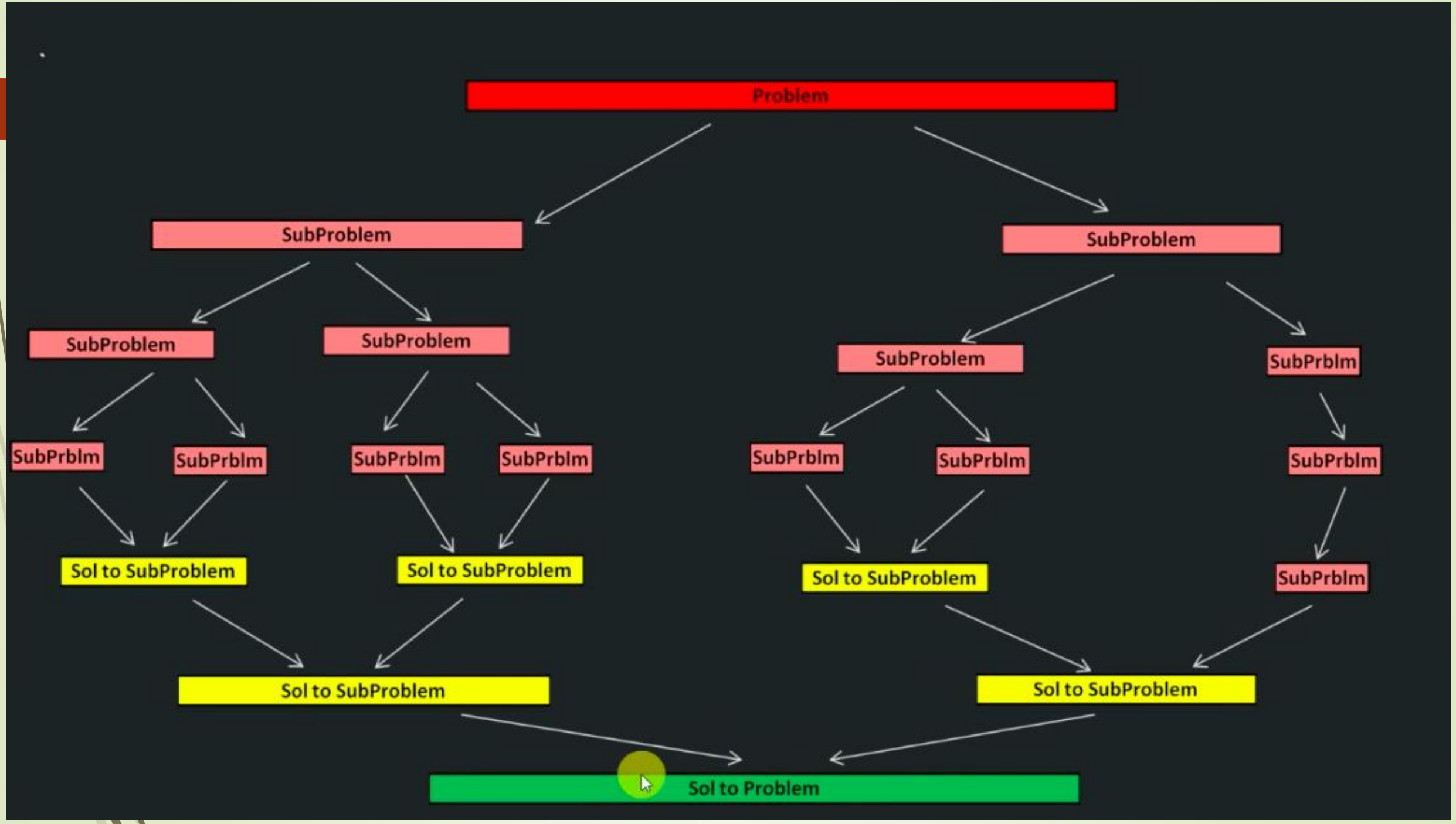
Dr. Subhra Rani Patra
SCOPE, VIT Chennai

What is 'Divide & Conquer' ?

- ✓ *Divide & conquer is an algorithm design paradigm which works by recursively breaking down a problem into sub-problems of similar type, until these become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a solution to the original problem.*

- ✓ *Example:*
  - ✓ *Software development by modules*




Problem
SubProblem
SubProblem
SubProblem
SubProblem
SubProblem
SubProblem
SubPrblm
SubPrblm
SubPrblm
SubPrblm
SubPrblm
SubPrblm
SubPrblm
Sol to SubProblem
Sol to SubProblem
Sol to SubProblem
SubPrblm
Sol to SubProblem
Sol to SubProblem
Sol to Problem

Property of D&C Algo:

- ✓ **Optimal Substructure :**
  - ✓ Any problem has optimal substructure property if its overall optimal solution can be constructed from the optimal solutions of its subproblem.
  - ✓ Example: Fib(n) = Fib(n-1) + Fib(n-2)

Why learn 'Divide and Conquer' ?

- ✓ Divide and conquer is most effective when problem has 'optimal substructure' property.
- ✓ Example:
  - ✓ Merge Sort
  - ✓ Quick Sort
  - ✓ Binary Search.

A **Binary search** algorithm finds the position of a specified input value (the search "key") within a sorted array . For binary search, the array should be arranged in ascending or descending order.

Why Binary Search Is Better Than Linear Search ?

A linear search works by looking at each element in a list of data until it either finds the target or reaches the end. This results in O(n) performance on a given list. A binary search comes with the prerequisite that the data must be sorted. We can use this information to decrease the number of items we need to look at to find our target. That is ,

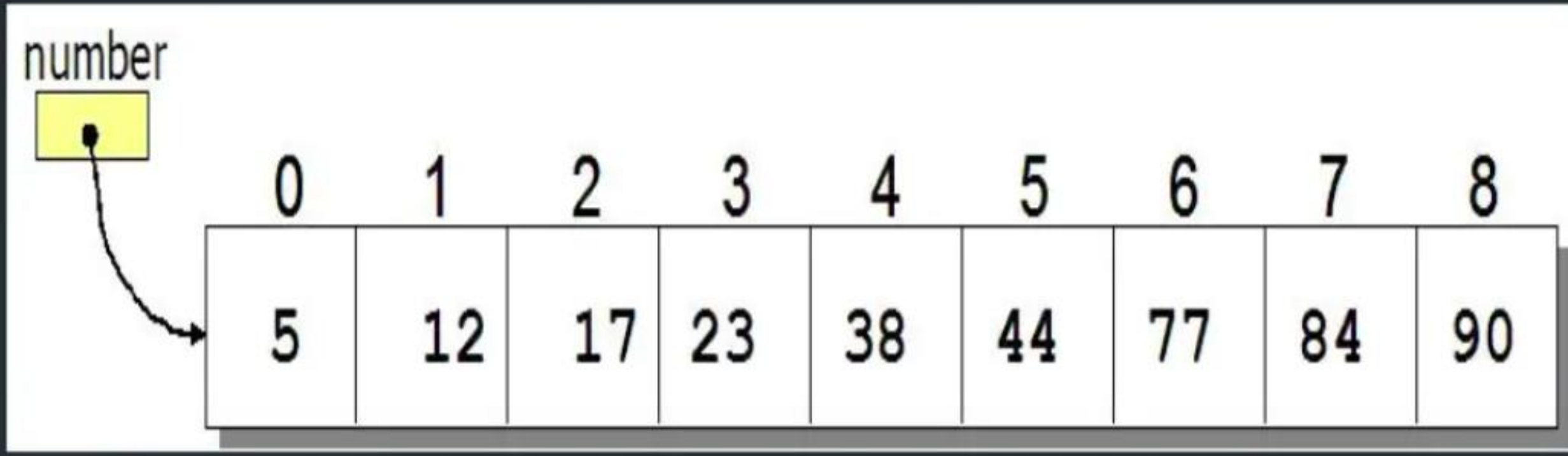
Binary Search is fast as compared to linear search because of less number of computational operations .

BINARY SEARCH ALGORITHM

Algorithm is quite simple. It can be done either recursively or iteratively:

- Get the middle element;
- If the middle element equals to the searched value, the algorithm stops;
- Otherwise, two cases are possible:
- Searched value is less, than the middle element. In this case, go to the step 1 for the part of the array, before middle element.
- Searched value is greater, than the middle element. In this case, go to the step 1 for the part of the array, after middle element.
- Now, The iterations should stop when searched element is found & when sub array has no elements. In this case, we can conclude, that searched value is not present in the array.

Example :

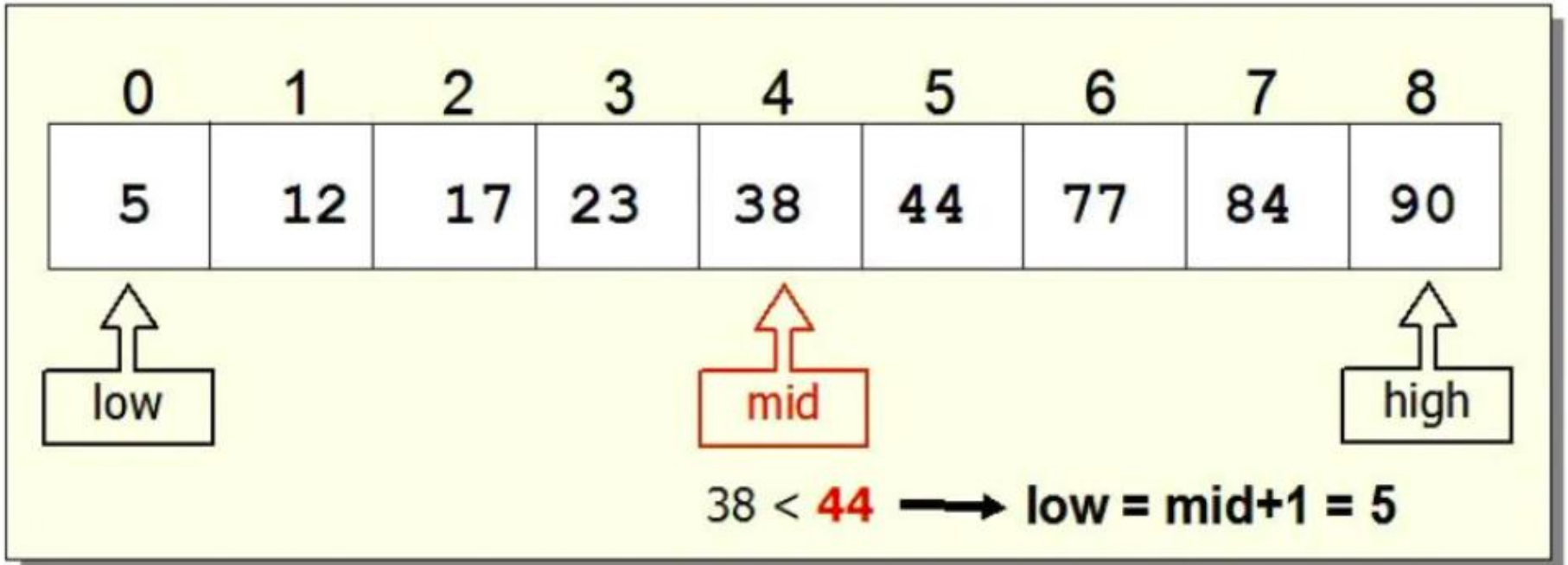


Step 1 :

search(44)

	low	high	mid
#1	0	8	4

$$mid = \left\lfloor \frac{low + high}{2} \right\rfloor$$

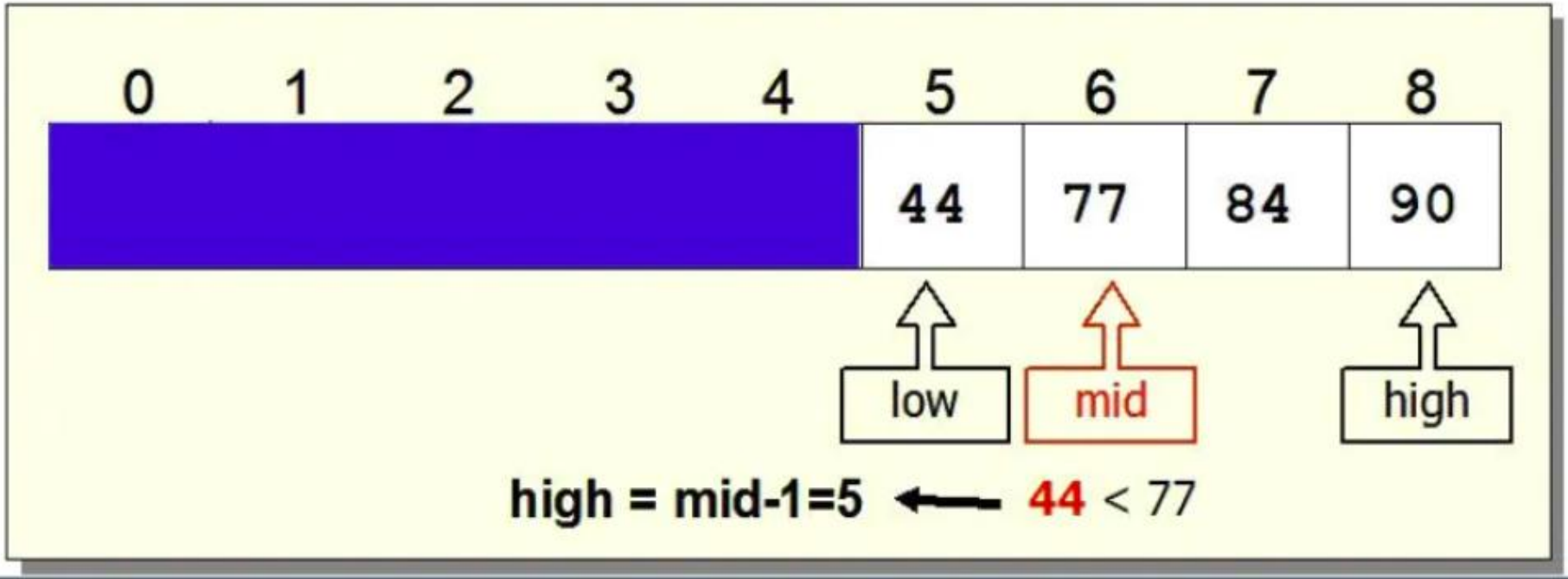


Step 2:

search(44)

	low	high	mid
#1	0	8	4
#2	5	8	6

$$mid = \left\lfloor \frac{low + high}{2} \right\rfloor$$

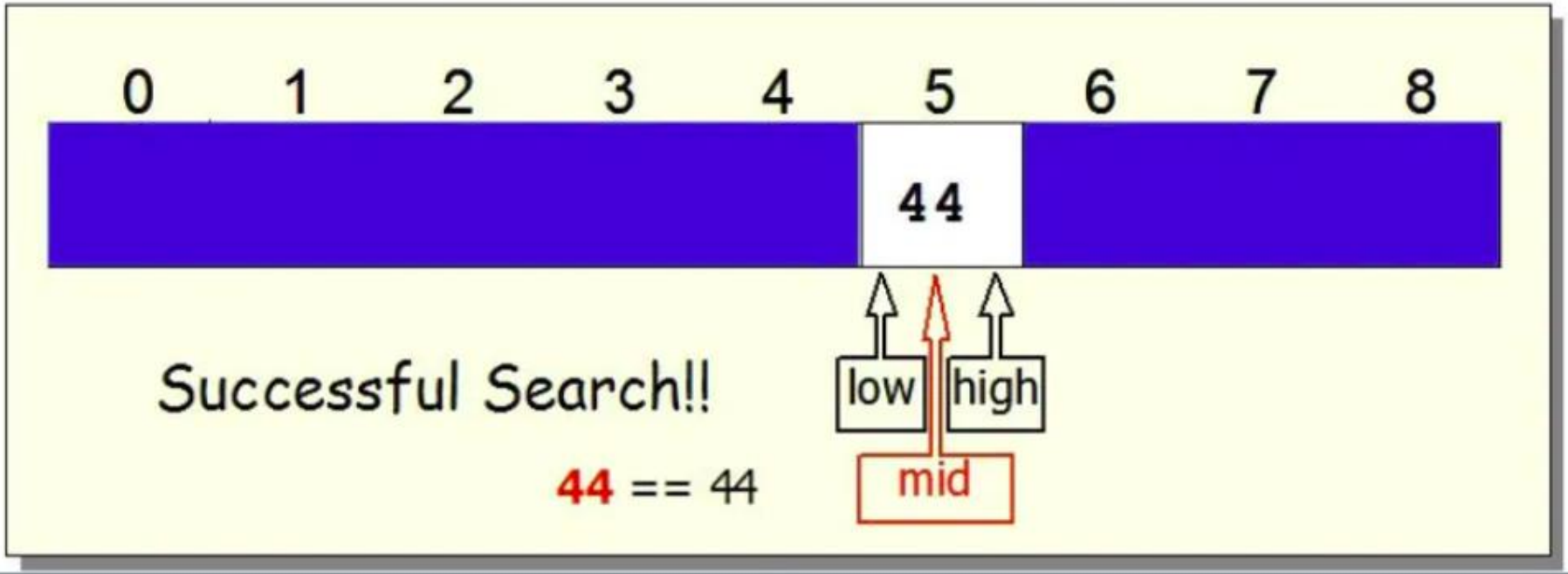


Step 3:

search(44)

	low	high	mid
#1	0	8	4
#2	5	8	6
#3	5	5	5

$$mid = \left\lfloor \frac{low + high}{2} \right\rfloor$$



Iterative method

```
Binary Search ( A[0 1 2 3.....n-1] , key)
    low← 0
    high← n-1
    while(low <= high)
    do
    {
          m←(low + high)/2
          if (key=A[m])
          then
```

```
        return m
        else if (key<A[m])
        then
                high← m-1
        else
                low← m+1
}
```

ALGORITHM (Recursive)

```
recursivebinarysearch(int A[], int first, int last, int key)
 if last<first
    index=-1
else
int mid = (first + last) / 2
if key=A[mid]
     index=mid
else if key<A[mid]
     index= recursivebinarysearch(A, first, mid – 1, key)
 else
     index= recursivebinarysearch(A, mid + 1, last, key)
return index
```

✓***Problem Statement:*** *Given a sorted array of 11 numbers, find number 110.*

10	20	30	40	50	60	70	80	90	100	110

```
BinarySearch(int findNumber, int arr[ ], start, end):  ---------- T(n)
    if (start equals end) ---------- O(1)
        if (arr[start] equals findNumber) ---------- O(1)
            return start ---------- O(1)
        else return error message that number does not exists in the array ---------- O(1)
    mid = FindMid(arr[ ], start, end) ---------- O(1)
    if mid > findNumber ---------- O(1)
        BinarySearch(int findNumber, int arr[ ], start, mid) ---------- T(n/2)
    else if  mid < findNumber ---------- O(1)
        BinarySearch(int findNumber, int arr[ ], mid, end ---------- T(n/2)
    else if mid = findNumber ---------- O(1)
        return mid ---------- O(1)
```

Time Complexity:

$T(n) = O(1) + T(n/2)$

Back Substitution:

$T(n)=T(n/2)+1$ ---------- Equation#1

$T(1) = 1$ ---------- Base Condition

$T(n/2)= T(n/4) + 1$ ---------- Equation#2

$T(n/4) = T(n/8) + 1$ ---------- Equation#3

$T(n) = T(n/2) + 1$

$= (T(n/4) + 1) + 1$

$= T(n/4) + 2$

$= (T(n/8) + 1) + 2$

$= T(n/8) + 3$

$= T(n/2^k) + k$

$= T(1) + \log n$

$= 1 + \log n$

$= \log n$

$n/2^k = 1$

$n = 2^k$

$K = \log n$

1) If $a > b^k$, then $T(n) = \Theta\left(n^{\log_b^a}\right)$
2) If $a = b^k$
   a. If $p > -1$, then $T(n) = \Theta\left(n^{\log_b^a} \log^{p+1} n\right)$
   b. If $p = -1$, then $T(n) = \Theta\left(n^{\log_b^a} \log\log n\right)$
   c. If $p < -1$, then $T(n) = \Theta\left(n^{\log_b^a}\right)$
3) If $a < b^k$
   a. If $p \geq 0$, then $T(n) = \Theta(n^k \log^p n)$
   b. If $p < 0$, then $T(n) = O(n^k)$

Time Analysis

- Best case - O (1) comparisons : In the best case, the item X is the middle in the array A. A constant number of comparisons (actually just 1) are required.

- Worst case - O (log n) comparisons : In the worst case, the item X does not exist in the array A at all. Through each recursion or iteration of Binary Search, the size of the admissible range is halved. This halving can be done ceiling(log n) times. Thus, ceiling(log n) comparisons are required.

Time Analysis

- Average case - O (log n) comparisons : To find the average case, take the sum over all elements of the product of number of comparisons required to find each element and the probability of searching for that element. To simplify the analysis, assume that no item which is not in A will be searched for, and that the probabilities of searching for each element are uniform.

Advantage and Disadvantage

- Advantage:

1. Binary search is an optimal searching algorithm using which we can search desired element very efficiently.

- Disadvantage:

1. This algorithm requires the list to be sorted . Then only this method is applicable.


Thank
you