Design and Analysis of Algorithms

Lecture - 9

Success is always inevitable with Hard Work and Perseverance

N. Ravitha Rajalakshmi

Learning Objective

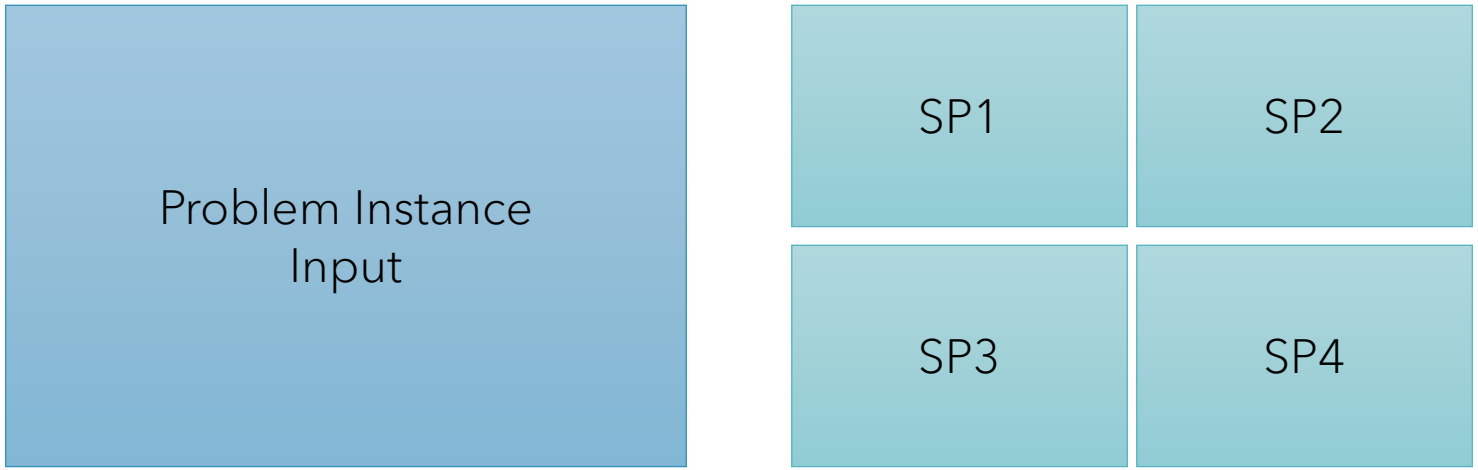
- Discuss the technique of Divide and Conquer
- General Framework for deriving solution through Divide and Conquer

Divide & Conquer

1. Divide Problem into subproblems

   Doing all at once is difficult

   Involves Repeated Computation



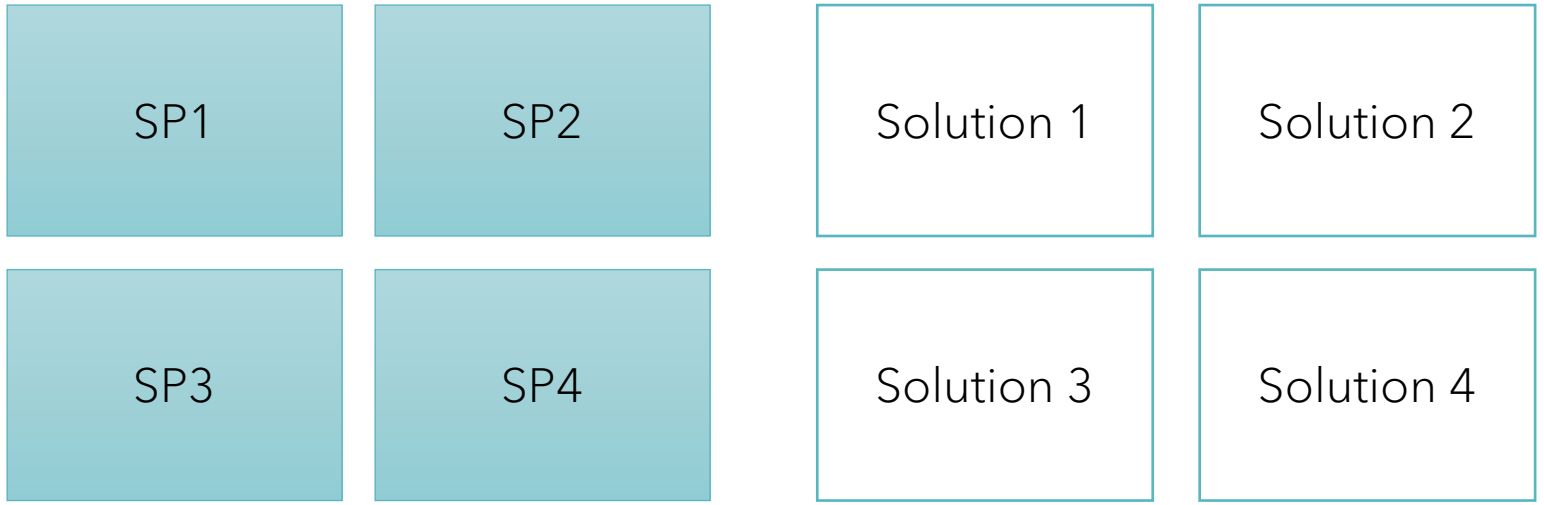
Subproblem should be of same type as that of original Problem

Non overlapping subproblems

Divide & Conquer

2. Solve the subproblem

Perform the required computation over the subproblem

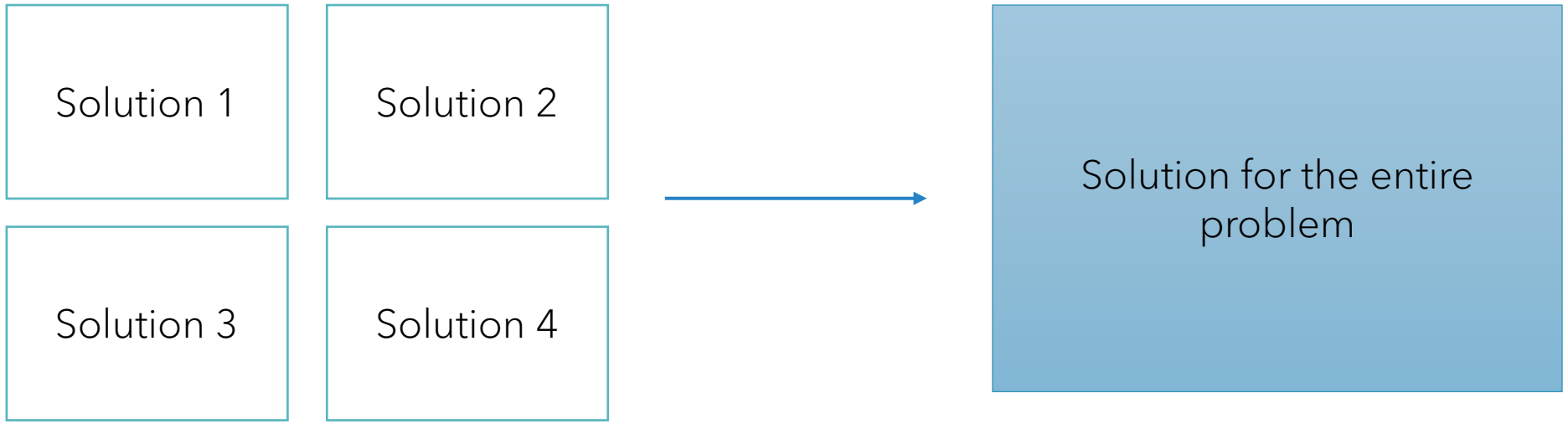


Divide & Conquer

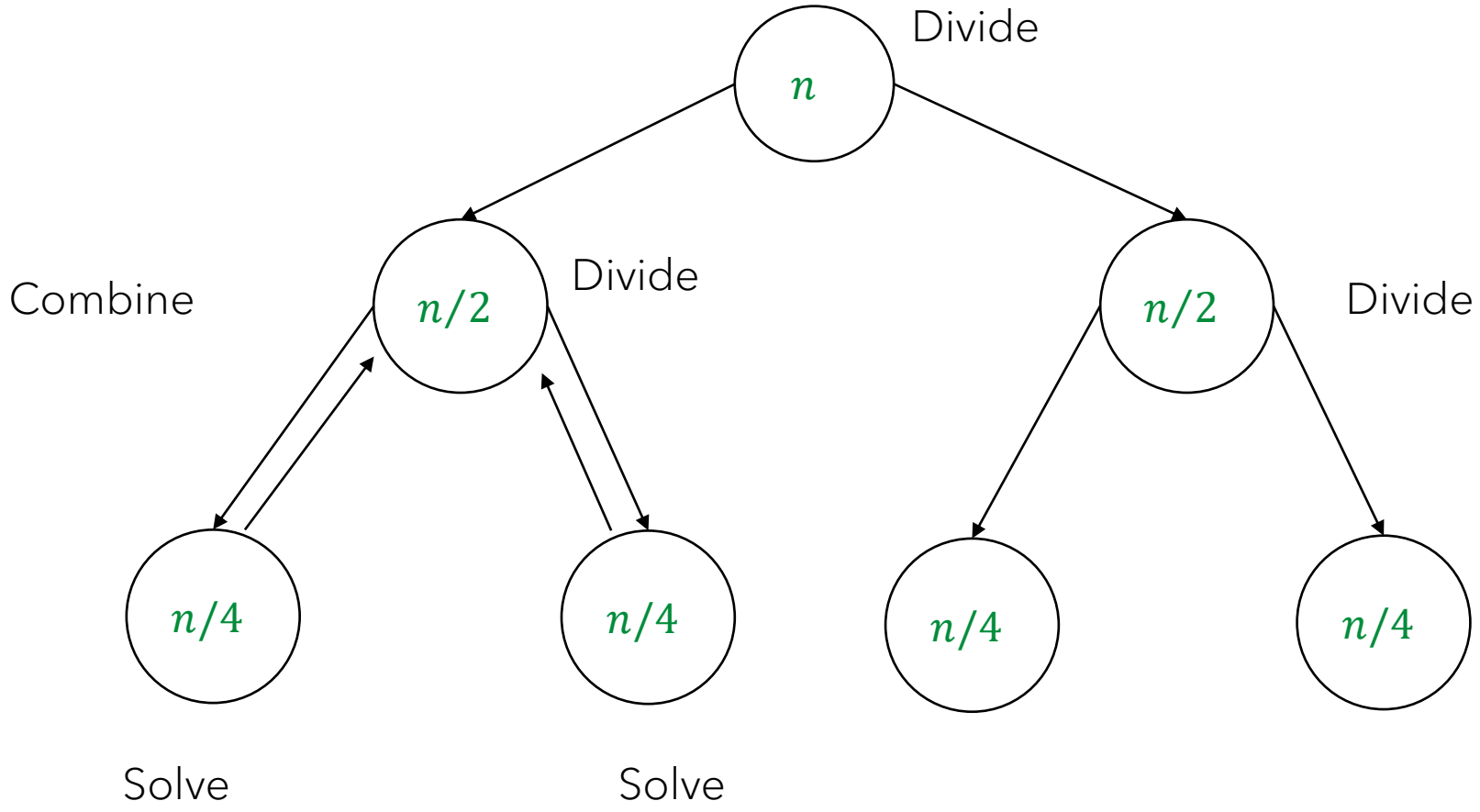
3. Combine the solution of subproblems

Solutions of the subproblem can be used directly.

Process the subproblem solutions to derive the final solution.



D&C ~ Recursion



Code Structure

Function Recursive (n)

```
# n – indicative of input size

    if (test on input size(n)){
            Solve
    }
    else {
            Compute
            Divide
            Conquer(Compute)
    }
```

Base Case

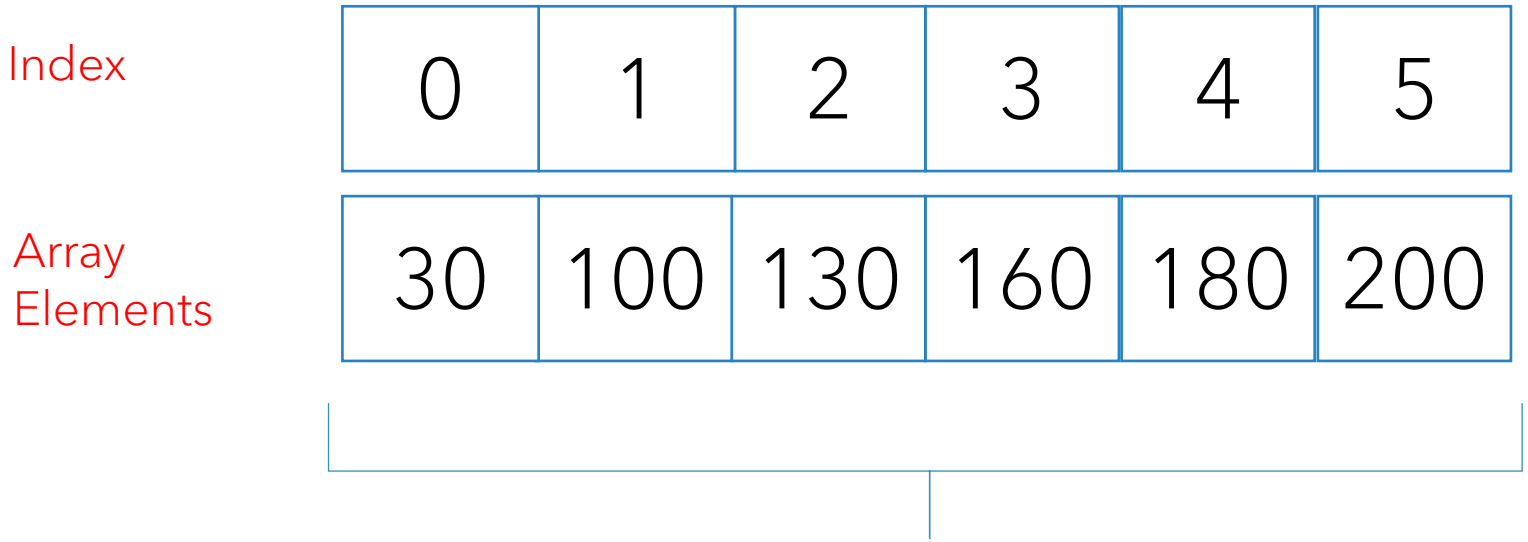
Recursive Case

Recursive call where problem instance is reduced

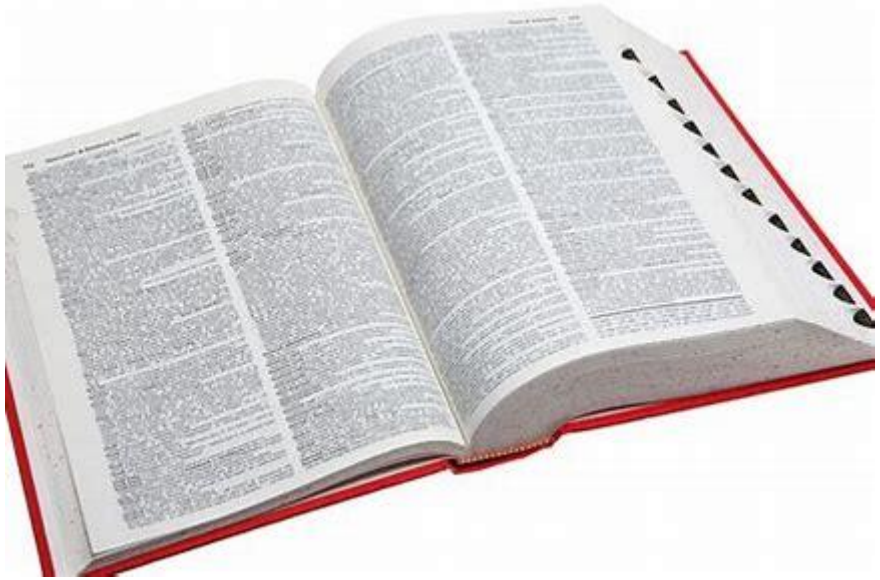
Binary Search

Input: An Sorted array A with n elements. A key k.

Output: An index, i, where A[i] = k. If there is no such i, then NOT_FOUND.



Dictionary



- Lookup for a word ‘Hard Work ‘
- Pick up a random Page x it shows words starting with letter M

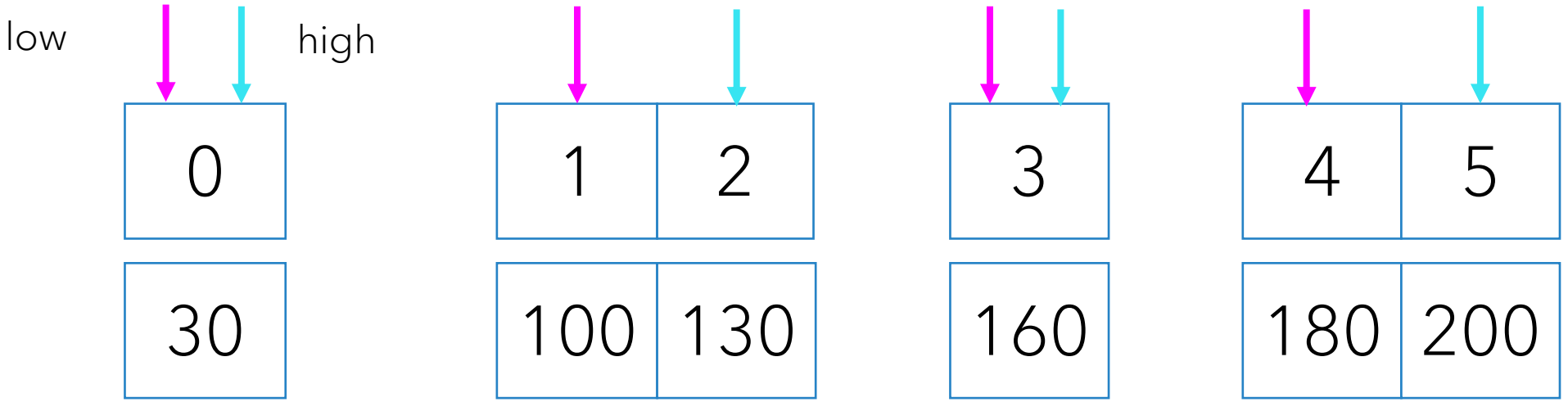
Now how will you identify the page further?

The word will be present in some page i < x

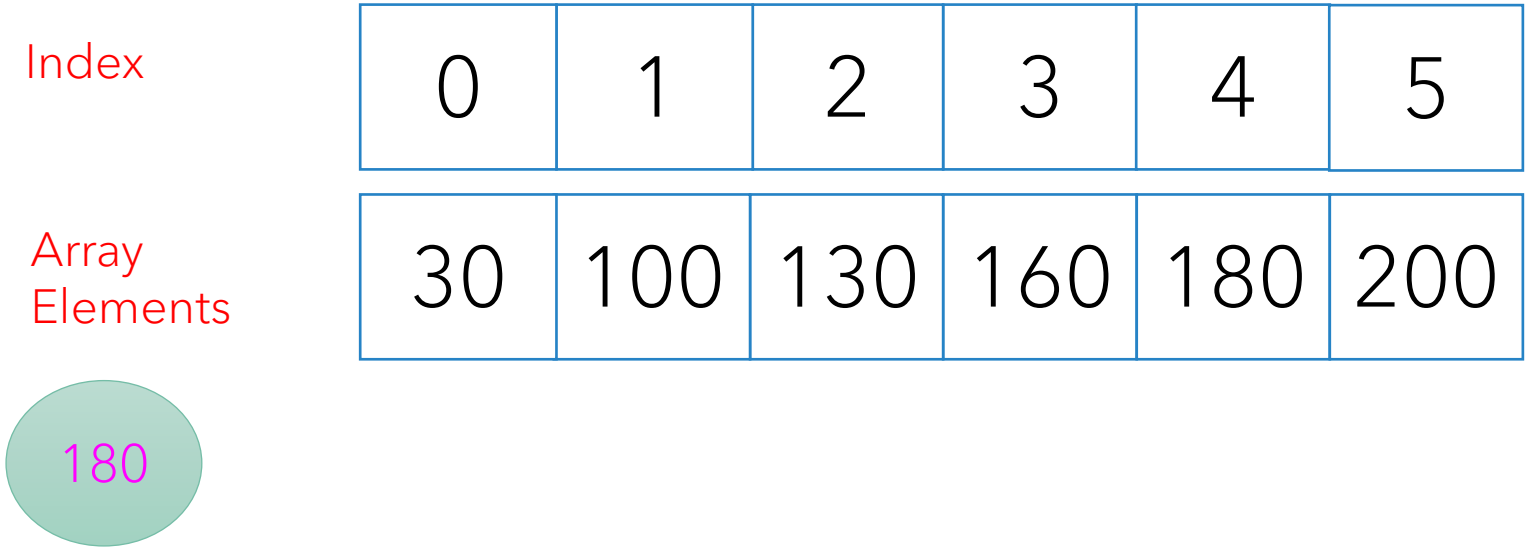
low & high

Subproblem indicators

0	1	2	3	4	5
30	100	130	160	180	200



Binary Search



- Look for the element in mid position of the array
- Element is present , return index
- Now confine the search to either left or right part of the array

Pause & Think

- Low and high , does it refer to index (or) store values

  Index

- Low == high

  Array contains only one element

- Low < high

  Array contains more than one element

Low > high

No elements in the array

Function Recursive (n)

```
# n – indicative of input size

    if (test on input size(n)){
        Solve
    }
    else {
        Compute
        Divide
        Conquer(Compute)
    }
```

Function BinSearch(A, low, high, key)

```
    if (low>high){
        return -1;
    }
    else {
        mid = (low + high)/2
        if(A[mid] == key)
            return mid;
        else if(A[mid]>key)
            return BinSearch(A,low,mid-1,key);
        else
            return BinSearch(A,mid+1,high,key);
    }
```

Index	0	1	2	3	4	5
Array Elements	30	100	130	160	180	200

BinSearch(A, 0, 5, 120)

mid =(0+5)/2 = 3

Trace the code for searching an element 120 in the given array

BinSearch(A, 0, 2, 120)

mid =(0+2)/2 = 1

BinSearch(A, 2, 2, 120)

mid =(2+2)/2 = 2

BinSearch(A, 2, 1, 120)

Element is not found in the array

Analysis of Binary Search

- Let n denote the number of elements in array (high – low)
- After every recursive call the array is divided into one subarray

  (elements from index low to mid) (or) (elements from index mid to high)

Recurrence relation

Input size : n Basic Operation: Comparison

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

T(0) = 1

By Master's Theorem , T(n) = O(log n)

Summary

- Discussed the basic framework of Divide and Conquer
- How D&C is applied for efficient search

Thank You

Happy Learning

Success is always inevitable with Hard Work and Perseverance