DESIGN AND ANALYSIS OF ALGORITHMS
UE19CS251

Shylaja S S
Department of Computer Science
& Engineering

DESIGN AND ANALYSIS OF ALGORITHMS

Divide and Conquer: Binary Search

Major Slides Content: Anany Levitin

Shylaja S S

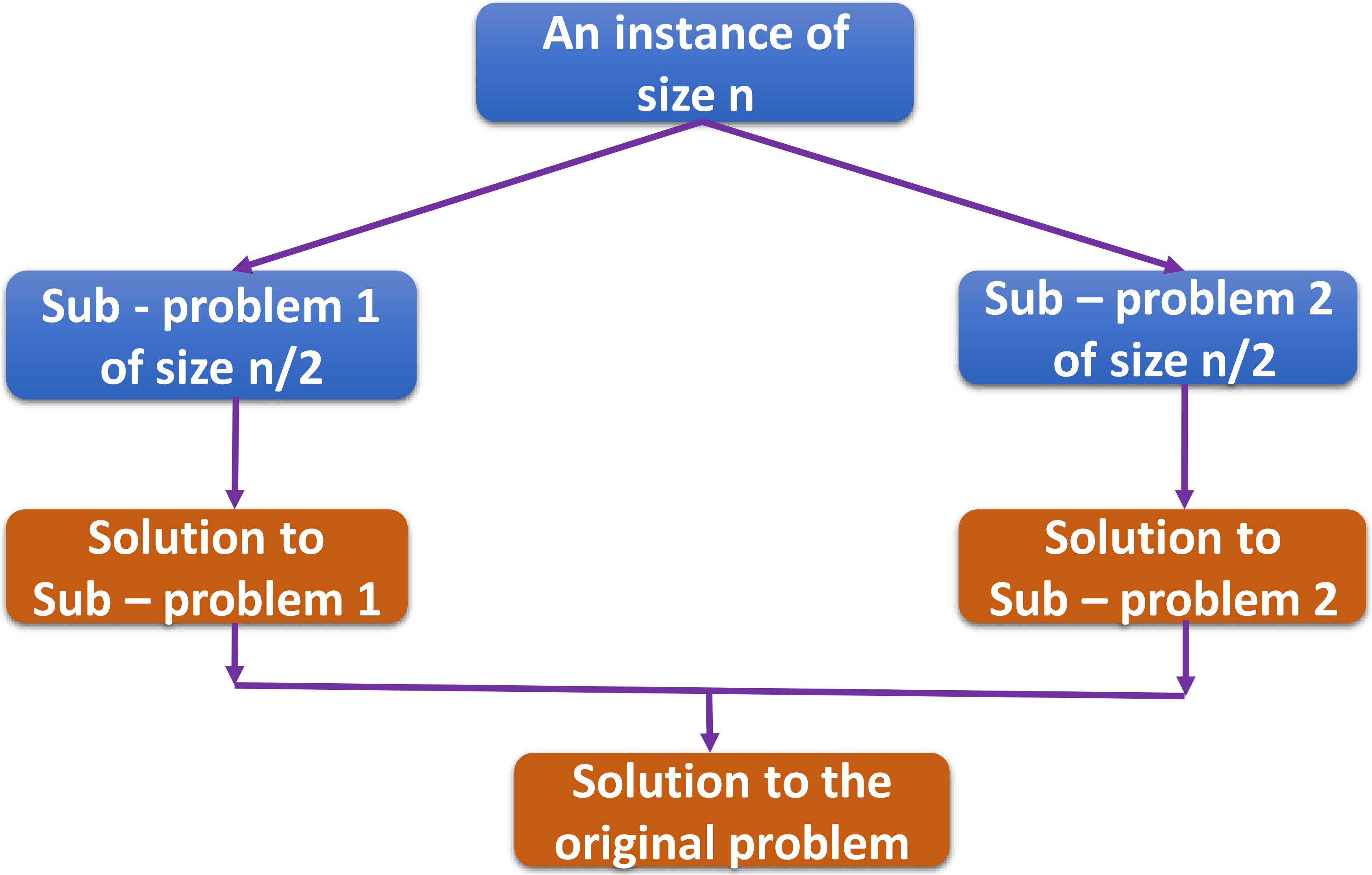
Department of Computer Science & Engineering

Divide and Conquer – Idea

- Divide and Conquer is one of the most well – known algorithm design strategies
- The principle underlying Divide and Conquer strategy can be stated as follows:
  - Divide the given instance of the problem into two or more smaller instances
  - Solve the smaller instances recursively
  - Combine the solutions of the smaller instances and obtain the solution for the original instance

Divide and Conquer – Idea

- Divide and Conquer



General Divide and Conquer

Recurrence

- In the most typical cases of Divide and Conquer, a problem's instance of size n can be divided into b instances of size n/b, with a of them needing to be solved
- Here a and b are constants; a >= 1 and b >= 1
- Assuming that size n is a power of b, we get the following recurrence for the running time:

$$T(n) = a * T(n/b) + f(n)$$

- f(n) is a function that accounts for the time spent on dividing the problem and combining the solutions

Recurrence

- For the recurrence:

$$T(n) = a * T(n/b) + f(n)$$

- If $f(n) \in \Theta(n^d)$, where $d >= 0$ in the recurrence relation, then:
  - If $a < b^d$, $T(n) \in \Theta(n^d)$
  - If $a = b^d$, $T(n) \in \Theta(n^d \log n)$
  - If $a > b^d$, $T(n) \in \Theta(n^{\log_b a})$

- Analogous results hold for O and Ω as well!

Binary Search - Idea

- Binary Search is a remarkably efficient algorithm for searching in a sorted array
- It works by comparing the search key K with the array's middle element A[m]
- If they match, the algorithm stops
- Otherwise, the same operation is repeated recursively for the first half of the array if K < A[m] and for the second half if K > A[m]

$$\begin{array}{ccc} & K & \\ & \updownarrow & \\ \underbrace{A[0] \ldots A[m-2]}_{\text{Search here if } K < A[m]} & A[m] & \underbrace{A[m+1] \ldots A[n-1]}_{\text{Search here if } K < A[m]} \end{array}$$

Binary Search - Algorithm

ALGORITHM BinarySearch(A[0 .. n -1], K)

// Implements non recursive binary search

// Input: An array A[0 .. n - 1] sorted in ascending order and a

// search key K

// Output: An index of the array's element that is equal to K or

// -1 if there is no such element

l ← 0; r ← n-1

while l ≤ r do

  $m \leftarrow \lfloor (l + r)/2 \rfloor$

  if K = A[m] return m

  else if K < A[m] r ← m-1

  else l←m+1

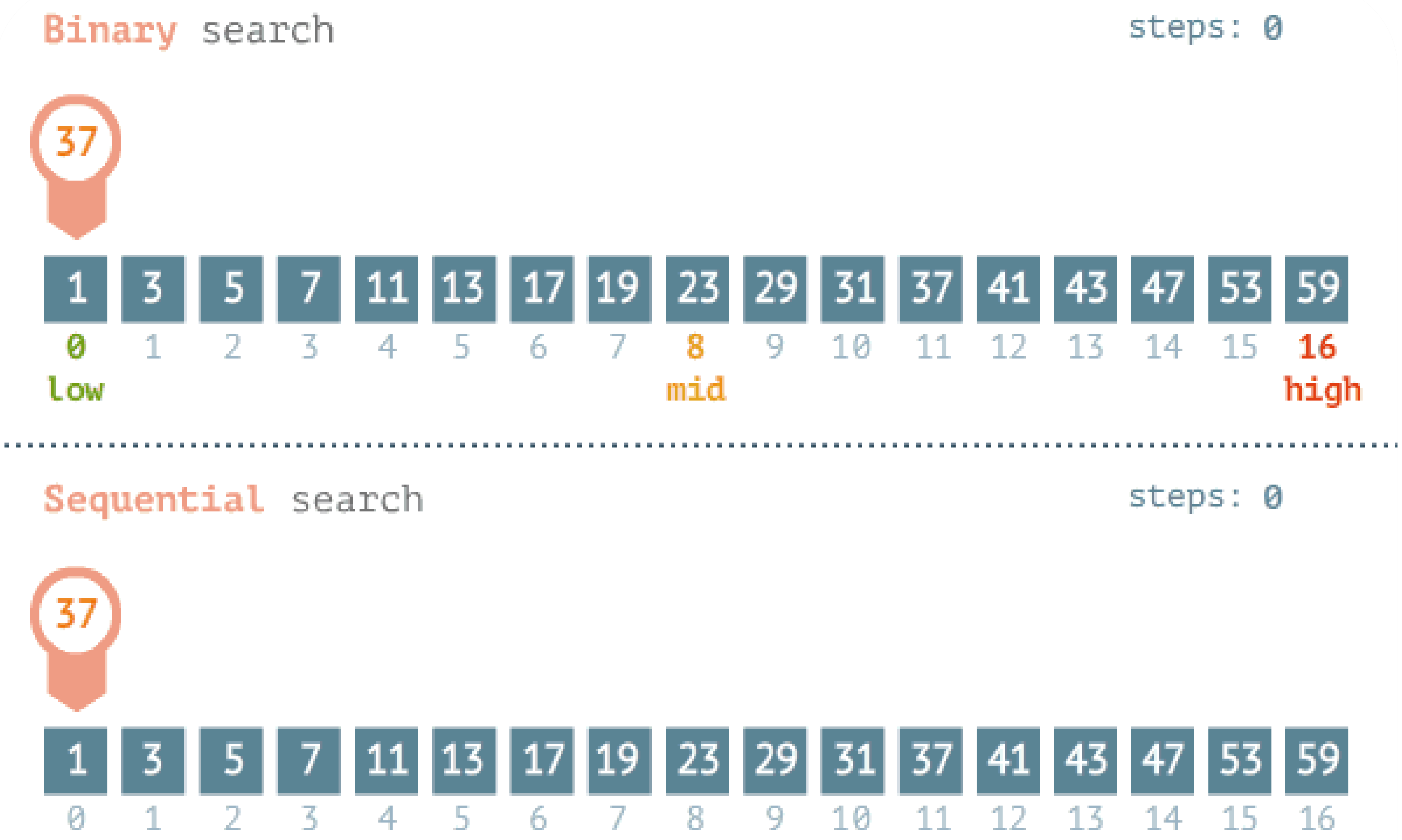
return -1

Binary Search - Example

Search Key K = 70

index	0	1	2	3	4	5	6	7	8	9	10	11	12
value	3	14	27	31	39	42	55	70	74	81	85	93	98
iteration1	*l*						*m*						*r*
iteration2								*l*		*m*			*r*
iteration3								*l,m*	*r*				

Binary Search Vs Linear Search



www.penjee.com

Binary Search - Analysis: Worst Case

The basic operation is the comparison of the search key with an element of the array

The number of comparisons made are given by the following recurrence:

$$C_{worst}(n) = C_{worst}\left(\left\lfloor n/2 \right\rfloor\right) + 1 \; for \; n > 1, C_{worst}(1) = 1$$

For the initial condition $C_{worst}(1) = 1$, we obtain:

$$C_{worst}(2^k) = k + 1 = \log_2 n + 1$$

For any arbitrary positive integer, n:

$$C_{worst}(n) = \lfloor \log_2 n \rfloor + 1$$

Binary Search - Analysis: Average Case

$$C_{avg} \approx \log_2 n$$

THANK YOU

Shylaja S S

Department of Computer Science & Engineering

shylaja.sharath@pes.edu