BITS Pilani
Hyderabad Campus

Data Structures and Algorithms Design

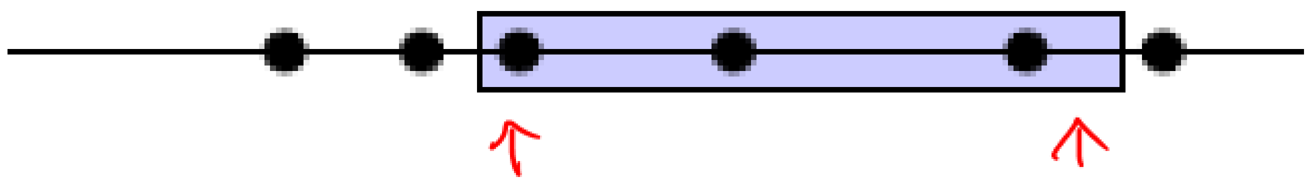
Febin.A.Vahab

Range query

- A **range query** q(A,i,j) on an array A=[a1,a2,a3,..,an] of *n* elements of some set *S*, denoted A[1,n], takes two indices 1<=i<=j<=n ,a function *f* defined over arrays of elements of *S* and outputs f(A[i,j])=f(ai,……,aj)

Range query

- Data:Points P={p_1, p_2, … p_n} in 1-D space (set of real numbers)
- Query: Which points are in 1-D query rectangle (in interval [x, x′])



Range query

n log n

- Range: [x, x’]
- **Data Structure 1:Sorted Array**

A=

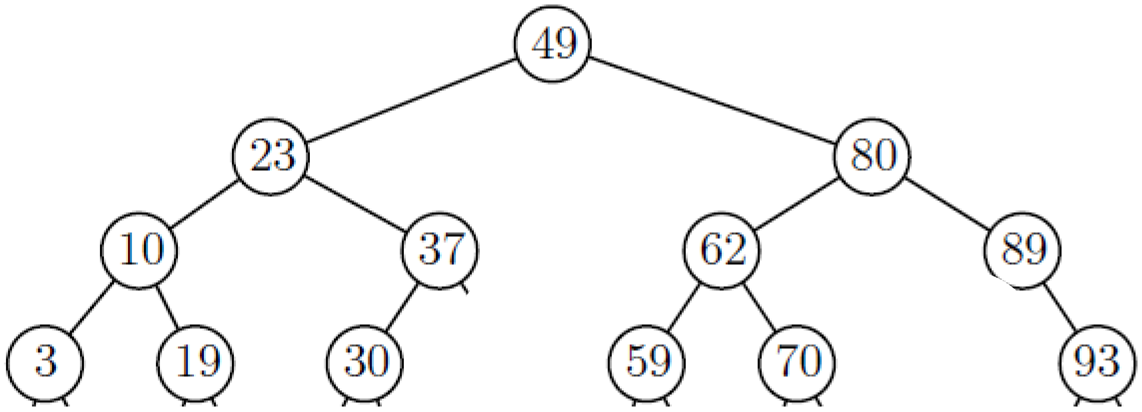
3	9	27	28	29	98	141	187	200	201	202	999

k

- Search for x and x’ in A by Binary search takes
- O(log n) time
- Output all points between them ,takes
- O(k) time
- $\boxed{\text{Total : O(k+logn)}}$

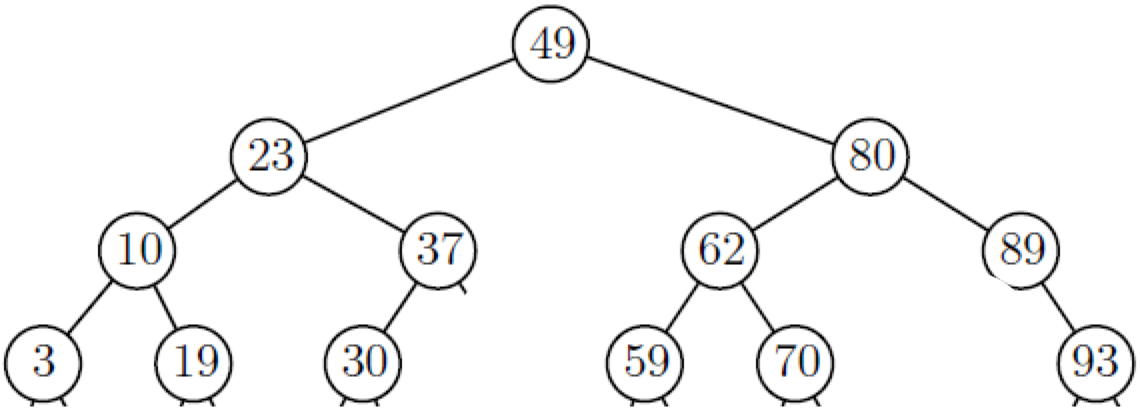
Range query

- Data Structure 2:BST
  - Search using binary search property.
  - Some subtrees are eliminated during search.



Range query

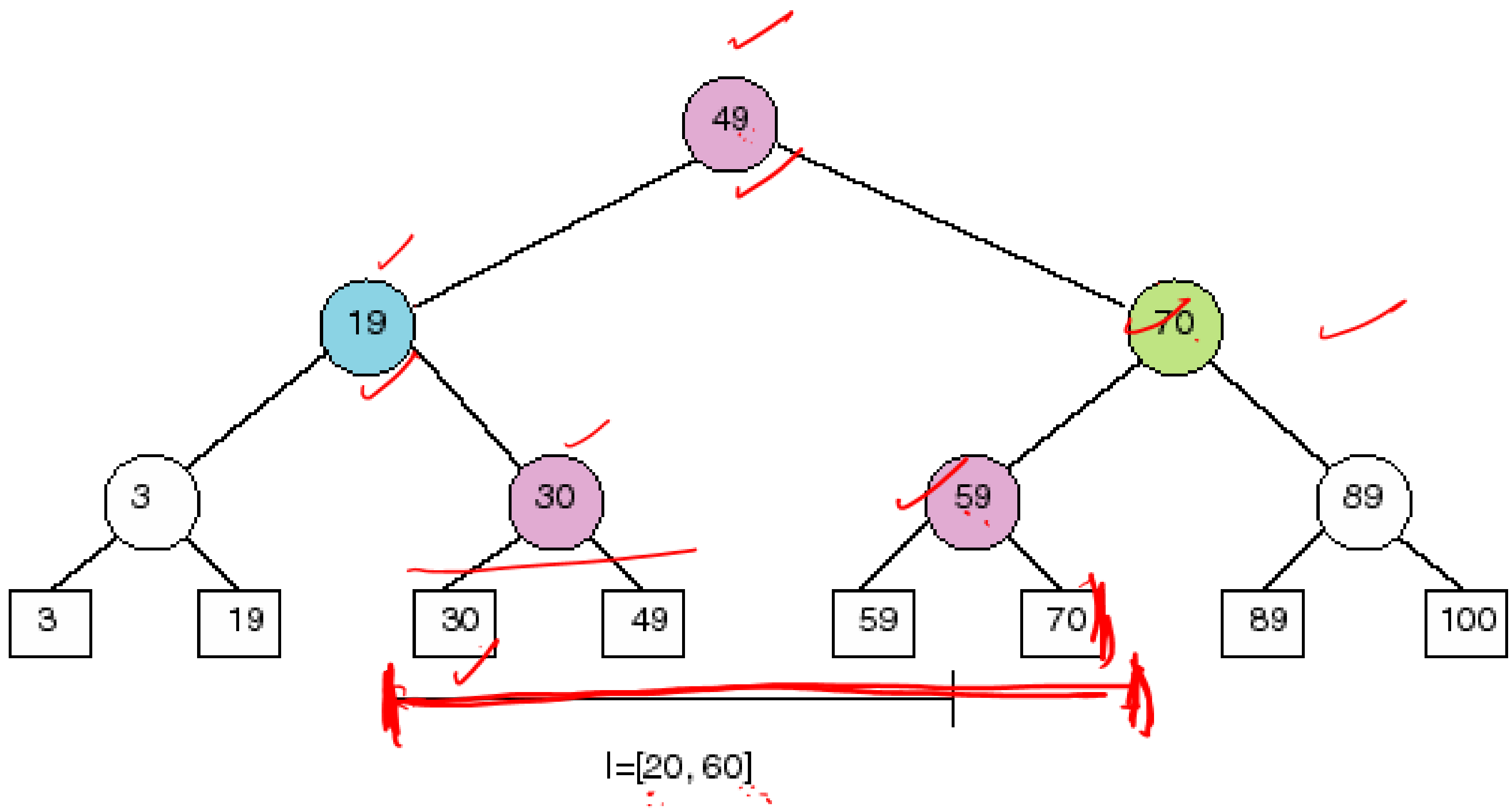
- Data Structure 2:BST
  – Search using binary search property.
  – Some subtrees are eliminated during search.



Range query

```
FindPoints([x, x'], T)
    if T is a leaf node, then
        if x <= val(T) <= x' then
            return { val(T) }
        else
            return {}
        end if
    end if
    <else T is an interior node of tree>
    if x' <= val(T) then
        return FindPoints([x, x'], left(T))
    else if x > val(T) then
        return FindPoints([x, x'], right(T))
    else <interval spans splitting value>
        return FindPoints([x, x'], left(T)) union FindPoints([x, x'], right(T))
    end if
```

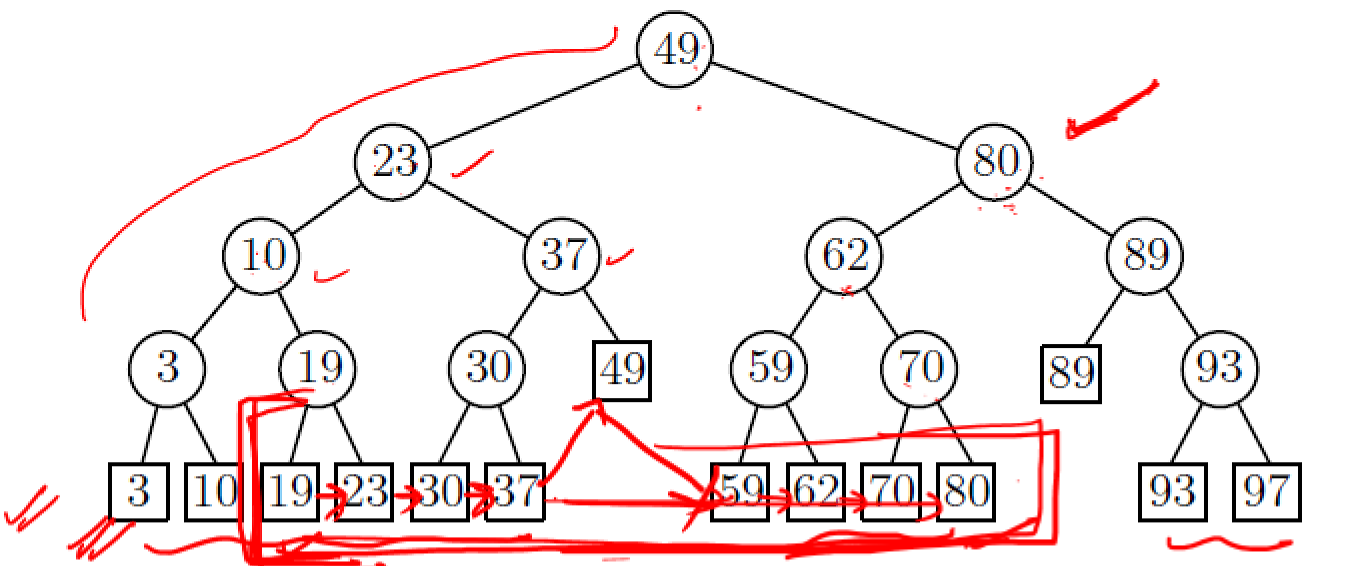
Range query



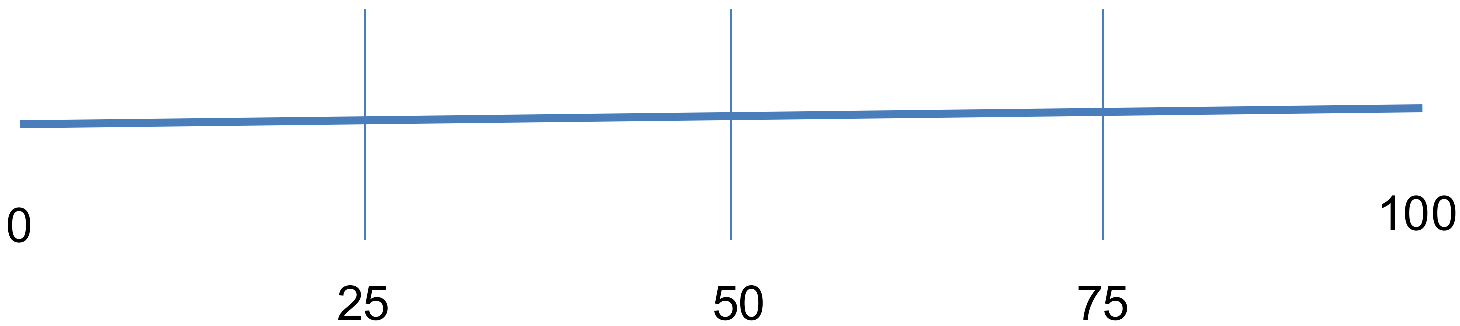
Range query

[19, 80]

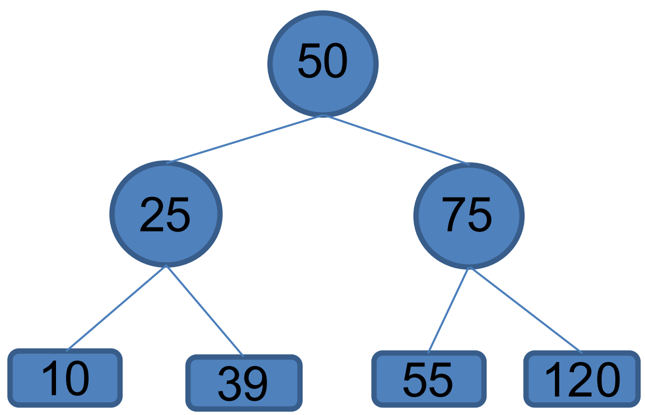
- Data Structure 3: BST with data stored in leaves
  - Internal nodes store splitting values (i.e., not necessarily same as data).
  - Data points are stored in the leaf nodes.



BST with data stored in leaves

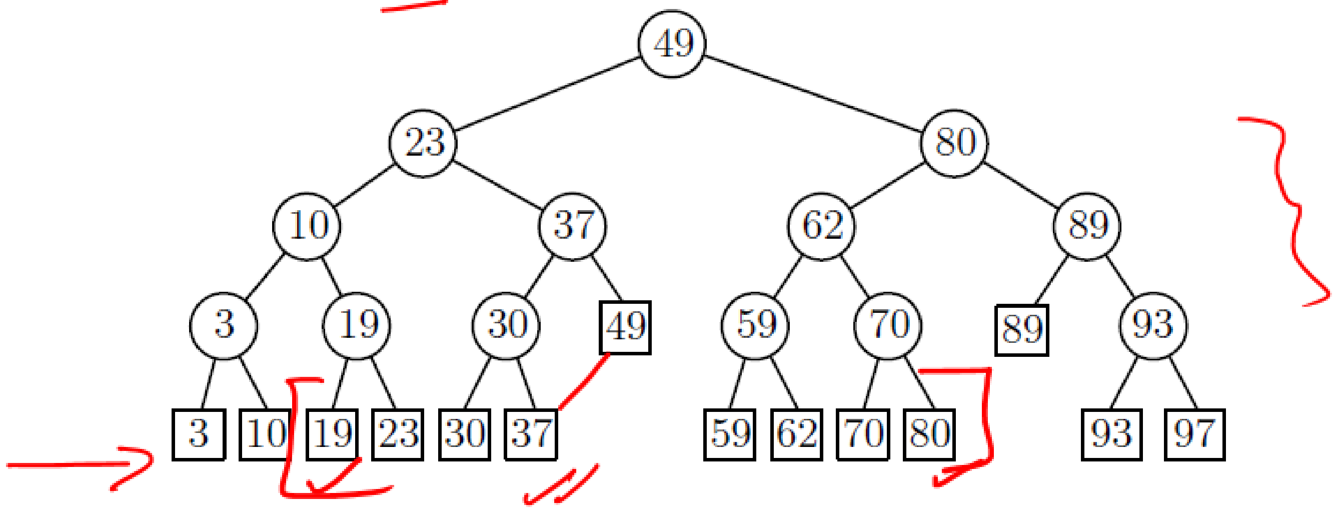


Data: 10, 39, 55, 120



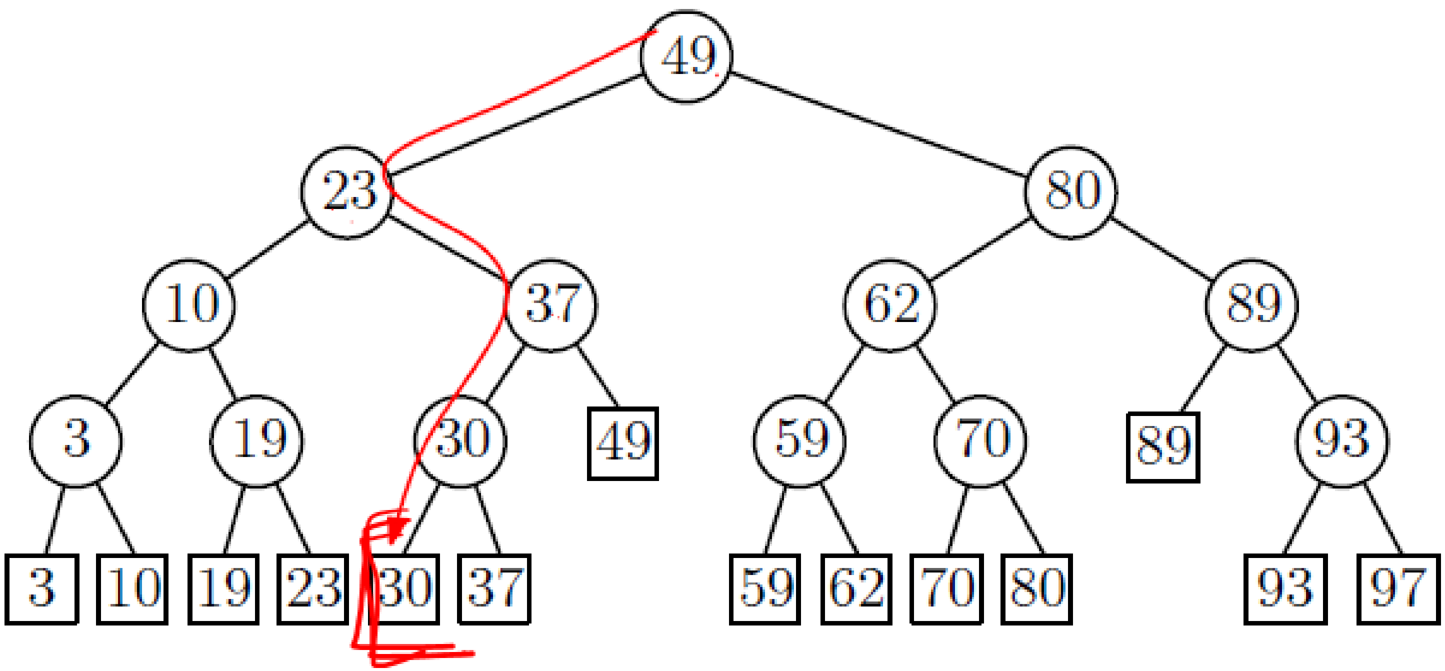
BST with data stored in leaves

- Retrieving data in [x, x’]
  - Perform binary search twice, once using x and the other using x’
  - Suppose binary search ends at leaves l and l’
  - The points in [x, x’] are the ones stored between l and l’ plus, possibly, the points stored in l and l’



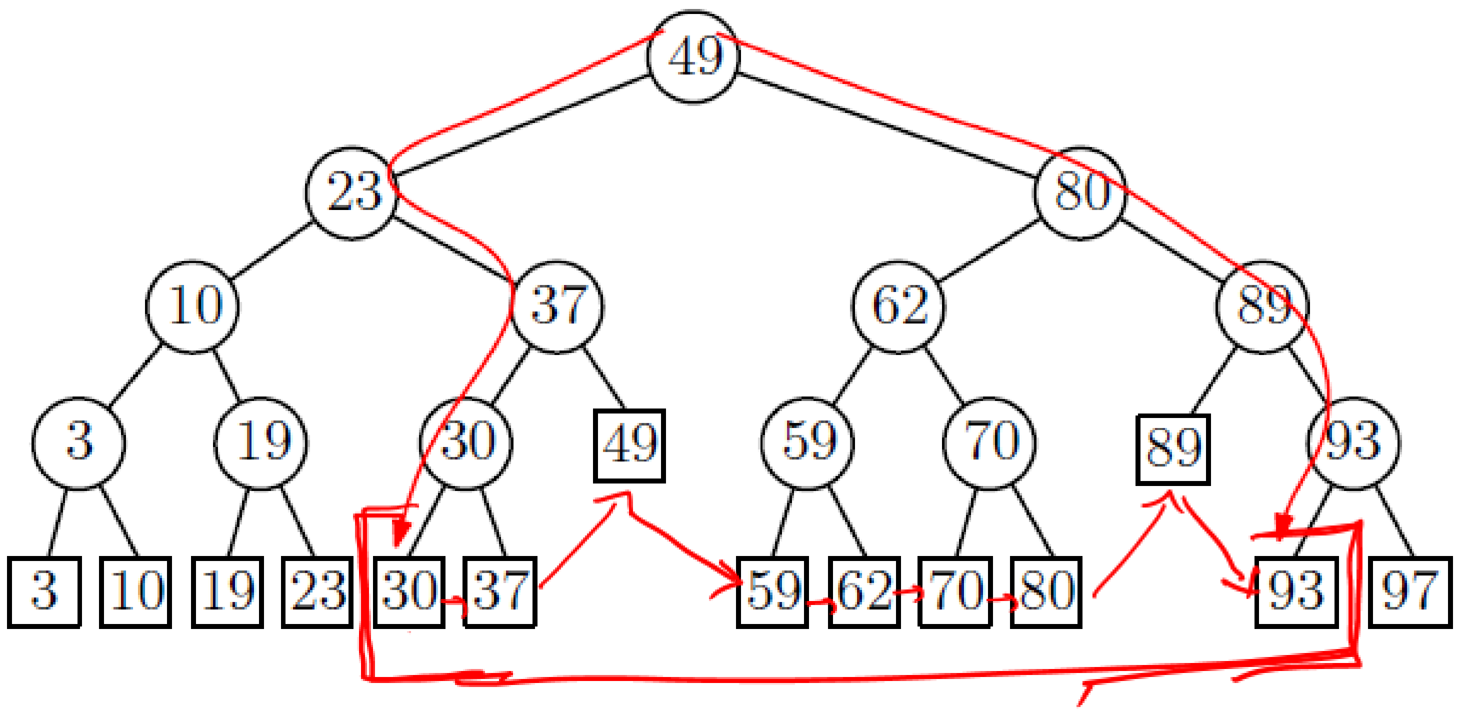
BST with data stored in leaves

- **Example:** retrieve all points in [25, 90]
  - The search path for 25 is:



BST with data stored in leaves

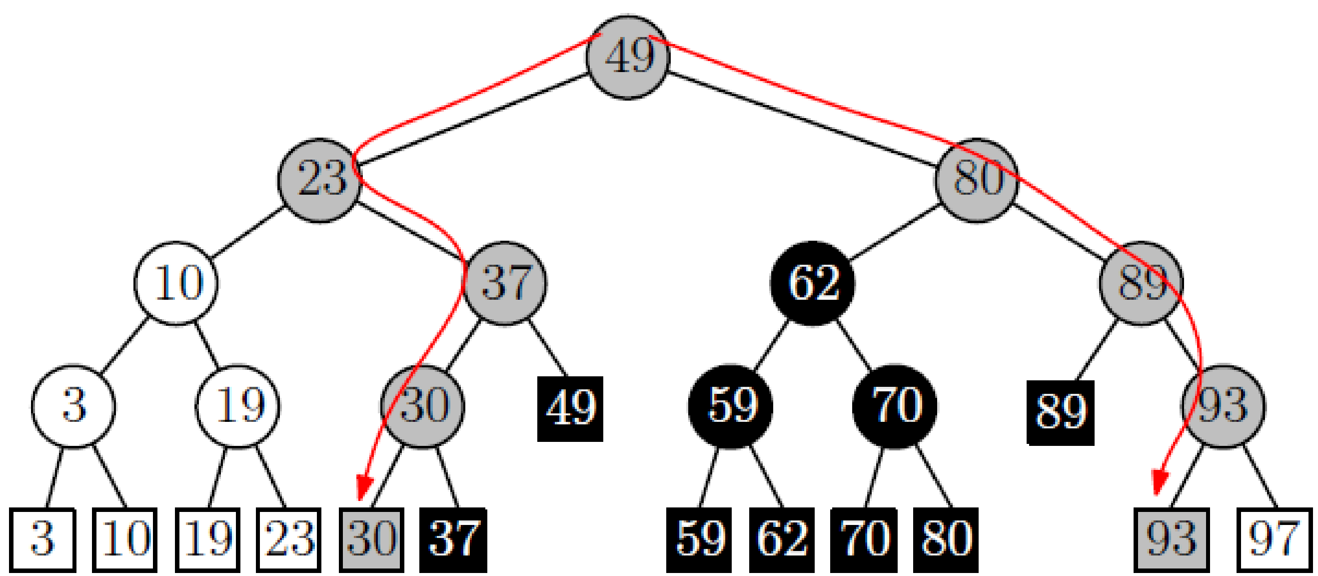
– **Example:** The search for 90 is:



BST with data stored in leaves

- Examine the leaves in the sub-trees between the two traversing paths from the root.

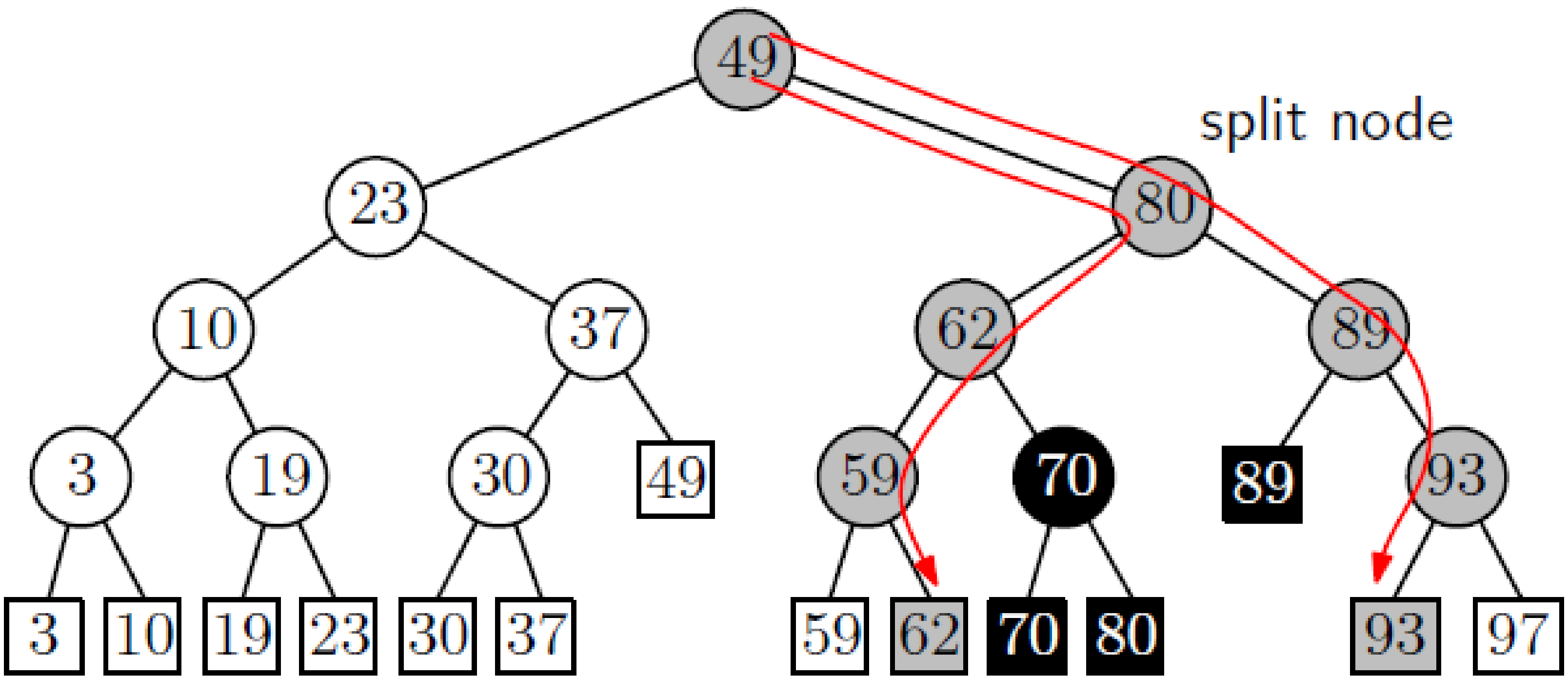
Split node



Retrieve all points in [25, 90]

Range Search – Another Example

A 1-dimensional range query with $[61, 90]$



Range Search

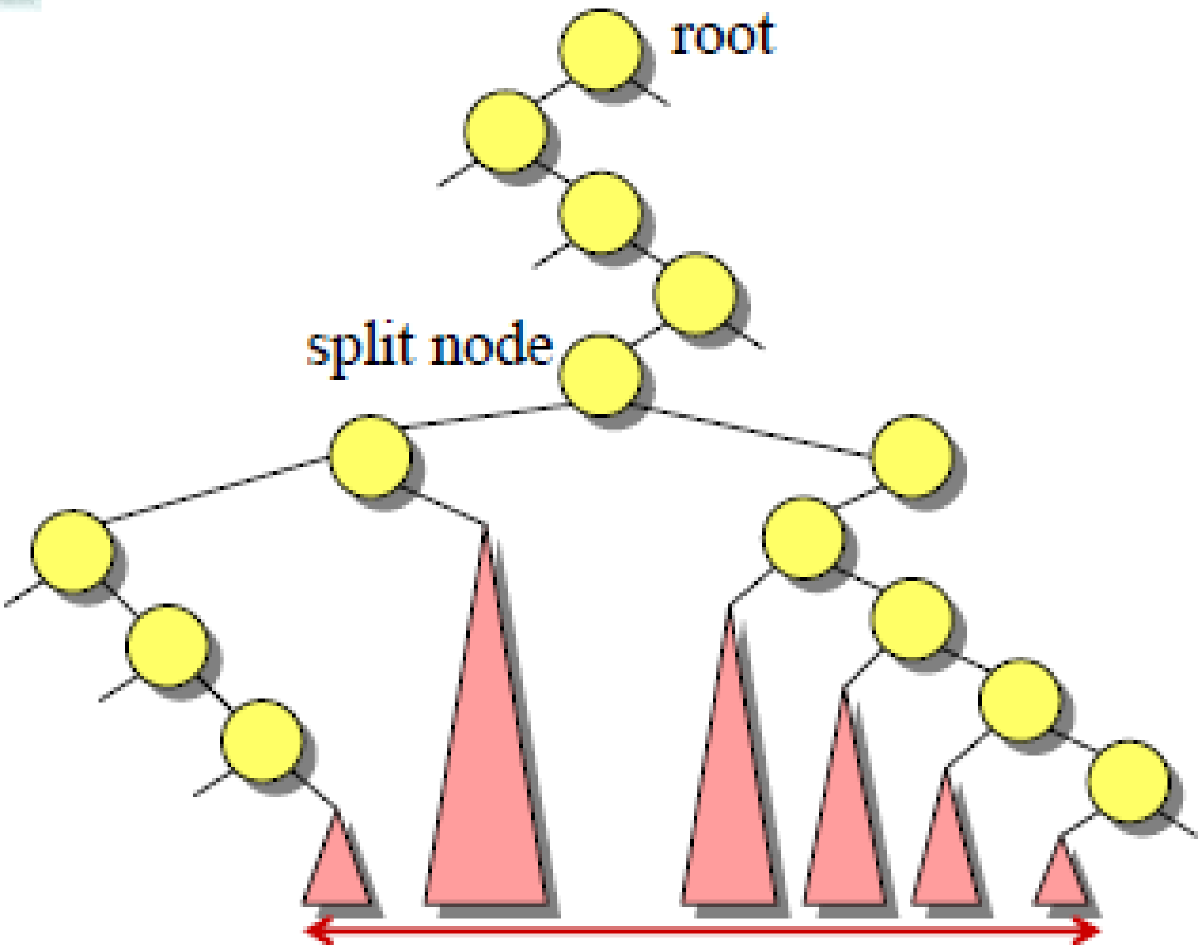
- How do we find the leaves of interest?

- Find **split node** (i.e., node where the paths to x and x’ split).

- **Left turn**: report leaves in right subtrees

- **Right turn**: report leaves in left substrees

Range Search

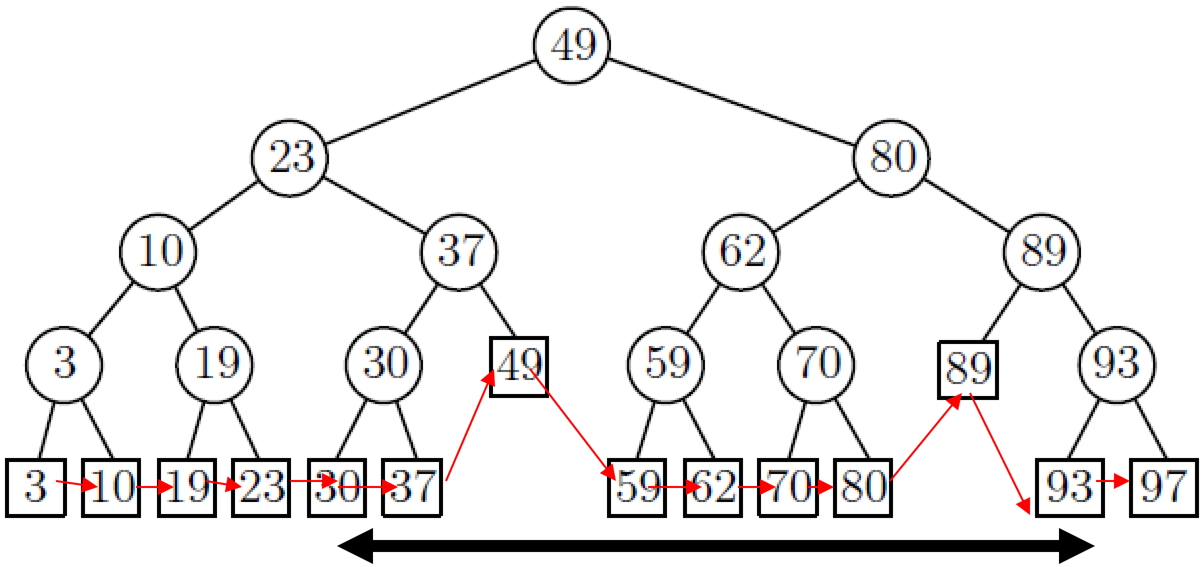


Range Search

- $O(\log n + k)$ time where k is the number of items reported.

Range Search

- Speed-up search by keeping the leaves in sorted order using a linked-list



- Applications

One-dimensional range searching [Goodrich]

- Given an ordered dictionary D, we want to perform the following query operation:
- findAllInRange(k1 , k2) : Return all the elements in dictionary D with key k such that k1 <=k<=k2

One-dimensional range searching

- How we can use a binary search tree T representing dictionary D to perform query
- findAllInRange(k1 , k2)
- Use a recursive method 1DTreeRangeSearch that takes as arguments the range parameters k1 and k2 and a node v in T

One-dimensional range searching

- If node v is external, we are done.
- If node v is internal, we have three cases, depending on the value of key (v) , the key of the item stored at node v:
- **key(v)< k1** : Rrecurse on the right child of v.
- **k1 <= key(v)<=k2** : Report element(v) and recurse on both children of v.
- **key(v) > k2** : Recurse on the left child of v.

Algorithm 1DTreeRangeSearch (k1 , k2 , v) :

Algorithm 1DTreeRangeSearch(k_1, k_2, v):

Input: Search keys k_1 and k_2, and a node v of a binary search tree T

Output: The elements stored in the subtree of T rooted at v, whose keys are greater than or equal to k_1 and less than or equal to k_2

```
if T.isExternal(v) then
    return ∅
if k1 ≤ key(v) ≤ k2 then
    L ← 1DTreeRangeSearch(k1, k2, T.leftChild(v))
    R ← 1DTreeRangeSearch(k1, k2, T.rightChild(v))
    return L ∪ {element(v)} ∪ R
else if key(v) < k1 then
    return 1DTreeRangeSearch(k1, k2, T.rightChild(v))
else if k2 < key(v) then
    return 1DTreeRangeSearch(k1, k2, T.leftChild(v))
```

lDTreeRangeSearch (k1 , k2 , v)

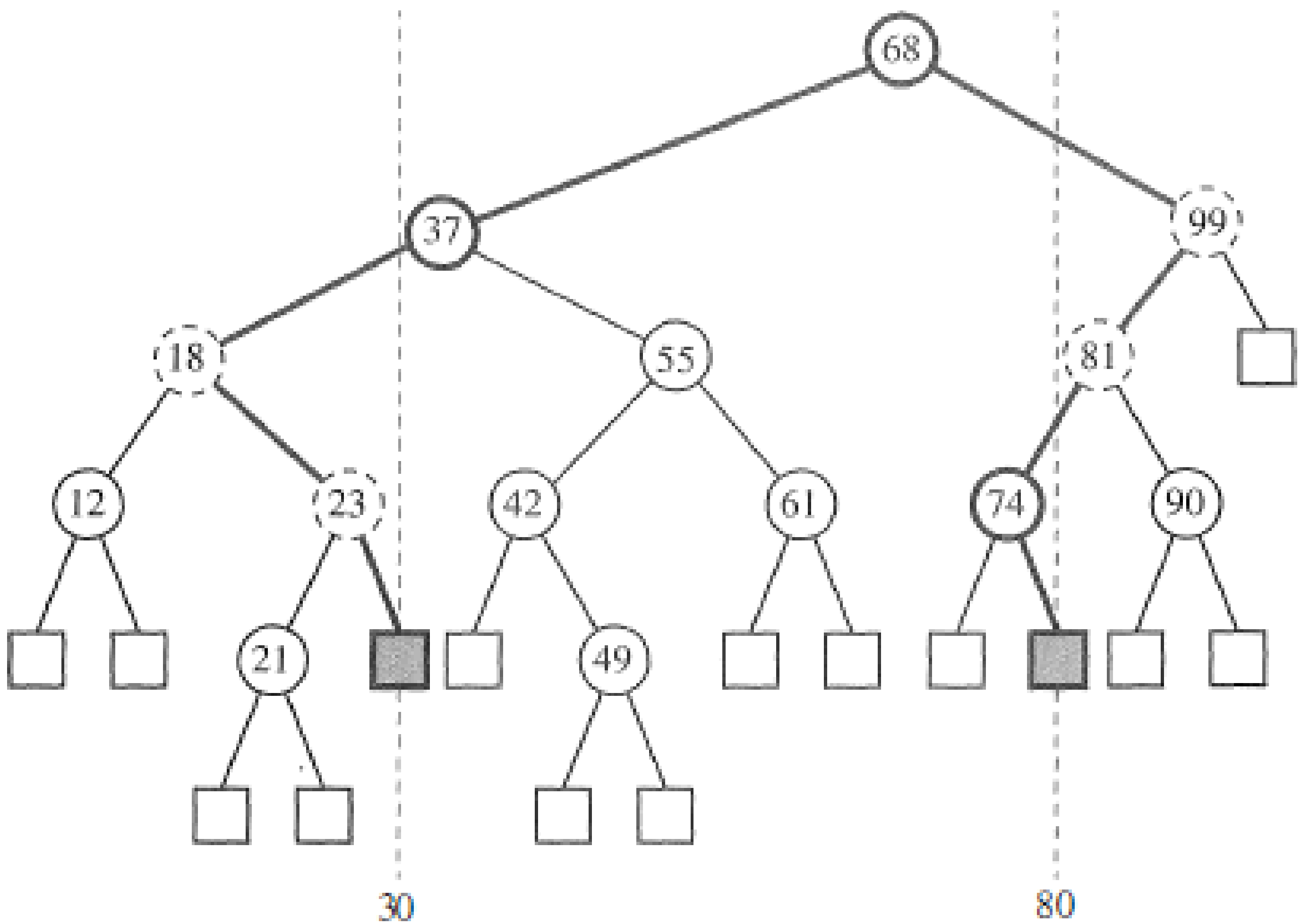
- We perform operation findAIIInRange(k1 , k2) by calling

  lDTreeRangeSearch (k1 , k2 , T. root( ))

  Example(Figure next slide)
- One-dimensional range search using a binary search tree for

  k1 = 30 and k2 = 80.
- Paths PI and P2 of boundary nodes are drawn with thick lines.
- The boundary nodes storing items with key outside the interval [k1 , k2] are drawn with dashed lines.

Algorithm 1DTreeRangeSearch (k1 , k2 , v) :



1DTreeRangeSearch -Performance

- Let P1 be the search path traversed when performing a search in tree T for key k1 .
- Path P1 starts at the root of T and ends at an external node of T .
- Define a path P2 similarly with respect to k2 . We identify each node v of T as belonging to one of following three groups

1DTreeRangeSearch -Performance

- Case 1:Node v is a **boundary node** if v belongs to PI or P2 ; a boundary node stores an item whose key may be inside or outside the interval [k1 , k2] .
- Case 2:Node v is an **inside node** if v is not a boundary node and v belongs to a subtree rooted at a right child of a node of PI or at a left child of a node of P2 ;an internal inside node stores an item whose key is inside the interval [k1 , k2] .

1DTreeRangeSearch -Performance

- Case 3:Node v is an **outside node** if v is not a boundary node and v belongs to a subtree rooted at a left child of a node of PI or at a right child of a node of P2 ; an internal outside node stores an item whose key is outside the interval (k1 , k2] .

1DTreeRangeSearch -Performance

- A balanced binary search tree supports one-dimensional range searching in an ordered dictionary with n items:
- The space used is O(n).
- Operation findAllInRange takes O(log n + s) time, where s is the number of elements reported.
- Operations insertItem and removeElement each take O(log n) time.

BST-Applications

- Used in many search applications where data is constantly entering/leaving, such as the map and set objects in many languages' libraries.
- Binary Space Partition - Used in almost every 3D video game to determine what objects need to be rendered. Binary space partitioning (BSP) is a method for recursively subdividing a space into convex sets by hyper planes. This subdivision gives rise to a representation of objects within the space by means of a tree data structure known as a BSP tree

BST-Applications

- Huffman Coding Tree: The branches of the tree represent the binary values 0 and 1 according to the rules for common prefix-free code trees. The path from the root tree to the corresponding leaf node defines the particular code word.
- It is used to implement multilevel indexing in DATABASE.

Sample Question 1

- *Prof X is standing at the door of his classroom. There are currently **N** students in the class, **i** th student got A_i candies. There are still **M** more students to come. At every instant, a student enters the class and wishes to be seated with a student who has **exactly** the same number of candies. For each student, Professor shouts YES if such a student is found, NO otherwise. Even if the student entering the class can't find a partner with equal no. of candies, he will still enter the class and be seated.*
- *Identify a data structure to implement the above problem statement.*

Sample Question 2

You have decided to run off to Los Angeles for the summer and start a new life as a rockstar. However, things aren't going great, so you're consulting for a hotel on the side. This hotel has N one-bed rooms, and guests check in and out throughout the day. When a guest checks in, they ask for a room whose number is in the range $[l, h]$.[1]

You want to implement a data structure that supports the following data operations as efficiently as possible.

1. INIT(N): Initialize the data structure for N empty rooms numbered $1, 2, \ldots, N$, in polynomial time.
2. COUNT(l, h): Return the number of **available** rooms in $[l, h]$, in $O(\log N)$ time.
3. CHECKIN(l, h): In $O(\log N)$ time, return the first empty room in $[l, h]$ and mark it occupied, or return NIL if all the rooms in $[l, h]$ are occupied.
4. CHECKOUT(x): Mark room x as not occupied, in $O(\log N)$ time.

THANK YOU!!!

BITS Pilani
Hyderabad Campus