Example: Using a Binary Tree for Searching

- What is Searching??
  - Determining whether a particular piece of data is present in a (large) collection of data
  - Example: In a collection of data about IISc students, searching for data on the student with S. R. No. 99999

```
struct StudentRecord{
    int SRNo;
    char Name[40];
    char HomeAddr[200];
    float Weight;
    …
}
```

```
struct StudentRecord STUDENTS[5450];
```

If we assume that the data is given in an array

- Sequential search
  - Look for the value starting from one end of the array

```
int Data[N];
for (int i=0; i<N; i++)
        if (Data[i] == value) break;
```

$$\frac{\sum_{i=1}^{N} i}{N} = \frac{N+1}{2}$$

  - Number of comparisons:
    - Best case? Worst case? Average case?
  - Think: Can we do better if we sort the data in the array?

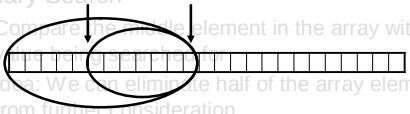
If the array is sorted (say in increasing order)

- Binary Search
  - Compare the middle element in the array with the value being searched for
  - Idea: We can eliminate half of the array elements from further consideration

```
int SortedData[N];
if (SortedData[N/2] >= value)
        /* continue search in SortedData[0 .. N/2]  */
else
        /* continue search in SortedData[N/2+1 .. N-1] */
```

If the array is sorted (say in increasing order)

- Binary Search
  - Compare the middle element in the array with the value being searched for
  - Idea: We can eliminate half of the array elements from further consideration



```
int SortedData[N];
if (SortedData[N/2] >= value)
        /* continue search in SortedData[0 .. N/2]  */
else
        /* continue search in SortedData[N/2+1 .. N-1] */
```

Binary Search ..

```
int BinarySearch (int value, int from, int to) {
        while (1) {
                int mid = (from + to) / 2;
                if (value < SortedData[mid])
                        to = mid - 1;
                else if (value > SortedData[mid])
                        from = mid + 1;
                else
                        return(mid);
                if (from > to)
                        return(/* value not present */);
        }
}
```

- Complexity?
- But we may want to add/delete data to/from the collection

Then there would be 3 operations

1. Searching
2. Inserting a new element
3. Deleting an existing element

Inserting/deleting an element involve $O(N)$ data movements

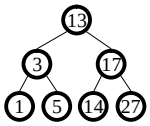
Example: Using a Binary Tree for Searching

- Idea: As new data is entered, insert it into a binary tree that has a special property
- Binary Search Tree Property
  - All elements stored in the left subtree of any node x are less than the element stored at node x, and all elements in the right subtree of node x are greater than the element stored at node x

Building a Binary Search Tree

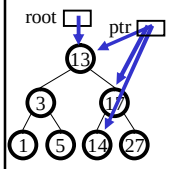
- In the beginning (before any data has been inserted) the tree is empty
- Suppose that the data to be inserted arrives in the order

  13, 17, 3, 5, 1, 14, 27



Searching with a Binary Search Tree

Searching for value = 14




```
struct TreeNode *root, *ptr;
ptr = root                    /* start at the root */
while (1) {
        if (value  >  ptr->data)
                ptr = ptr->right; /* go to right
subtree */
        else if  (value  <  ptr-> data)
                ptr = ptr->left; /* go to left
        else return (ptr);
        if (ptr == NULL)
                return( /* value not present */ );
}
```

Number of Comparisons?

3

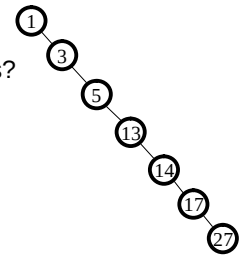
- In terms of number of nodes in tree, $\lceil \log_2 N \rceil$
- But this Binary Search Tree is balanced, in fact, full
- What if the same data had been inserted in a different order?
- e.g., 1, 3, 5, 13, 14, 17, 27

BST with 1, 3, 5, 13, 14, 17, 27

Search for value 14

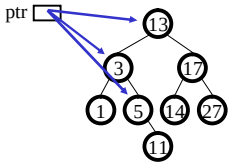
Number of comparisons?
5

Worst case?
N



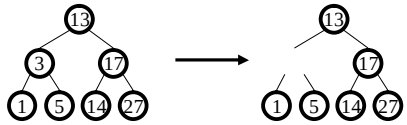
Insertion into a Binary Search Tree

- To insert new value, X
  - Search for X
  - If the search fails (at a NULL link)
    - Insert a new node with value X in place of the NULL
    - Example: Insert 11



Deletion from a Binary Search Tree

- Example: Delete the value 3



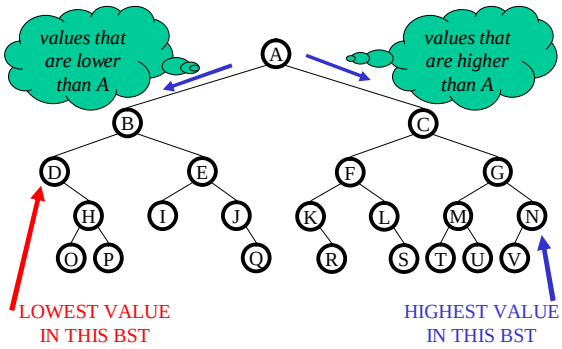
- But this is not a Binary Search Tree!
- We have to fill the vacancy that was created

Deletion from a Binary Search Tree

- Filling the vacancy created
  - Must maintain the Binary Search Tree Property
    - All elements stored in the left subtree of any node x are less than the element stored at node x, and all elements in the right subtree of node x are greater than the element stored at node x
  - i.e., if the value X was deleted, replace it by the immediately lower value in the BST
    - Or the immediately higher value in the BST

Getting to know the BST

- Question: Where is the lowest value located?



Deletion from a Binary Search Tree ...

- Filling the vacancy created ...
  - Must maintain the Binary Search Tree Property
    - All elements stored in the left subtree of any node x are less than the element stored at node x, and all elements in the right subtree of node x are greater than the element stored at node x
  - i.e., if the value X was deleted, replace it by the immediately lower value in the BST
    - Or the immediately higher value in the BST
  - i.e., highest value in left subtree of node with X
    - Or lowest value in right subtree of node with X

Deletion from a Binary Search Tree ...

case (node to be deleted)

1. Leaf node
   - Simply delete the node
2. Node with no left subtree
   - Replace by right subtree
3. Node with no right subtree
   - Replace by left subtree
4. Node with both left and right subtree
   - Replace by either immediately lower value or immediately higher value in the BST