Tutorial 08: Selection Problem

CSCI2520 - DATA STRUCTURES AND APPLICATIONS

TUTOR: ZHENG CHENGUANG

Outlines

- Selection Problem
- Two Naive Approaches
- Quickselect
- Dynamic Selection

Selection Problem

- A selection problem is the problem for finding the ***kth smallest*** element in a set of values.
- For example
  - For array (11, 6, 43, 7, 14, 28, 9, 2, 4, 37, 18, 34, 8)
  - What's the answer if k = 3? k = 7? k = 1? k = 13?
- Question: what value of k makes the problem simplest?

Find the minimum/maximum of a list.

Two Naive Approaches

- How to find kth smallest element of a list?
  - Sort the array and go to the kth position.
  - Find the smallest element, remove it from the list, and find the (k-1)th smallest element in the remaining list.
- Question: what are the complexities of them (suppose array length is n) ? Which one is better?

First: O(nlogn)   Second: O(kn)

Exercise 1

- Implement the second approach with the following prototype.
  - find the smallest element
  - remove it from the list
  - find the (k-1)th smallest element in the remaining list.

```
int FindKthElement(int array[], int n, int k);
```

Exercise 1

```
int FindKthElement(int array[], int n, int k){
    int i, j, iMin, tmp;
    for (i = 0; i < k; i++) {
        iMin = i;
        for (j = i + 1; j < n; j++)
            if (array[j] < array[iMin])
                iMin = j;
        tmp = array[i];
        array[i] = array[iMin];
        array[iMin] = tmp;
    }
    return array[k - 1];
}
```

Round `i`

Find the index `iMin` such that `array[iMin]` is the smallest in `array[i … n-1]`

Exchange `array[i]` and `array[iMin]`

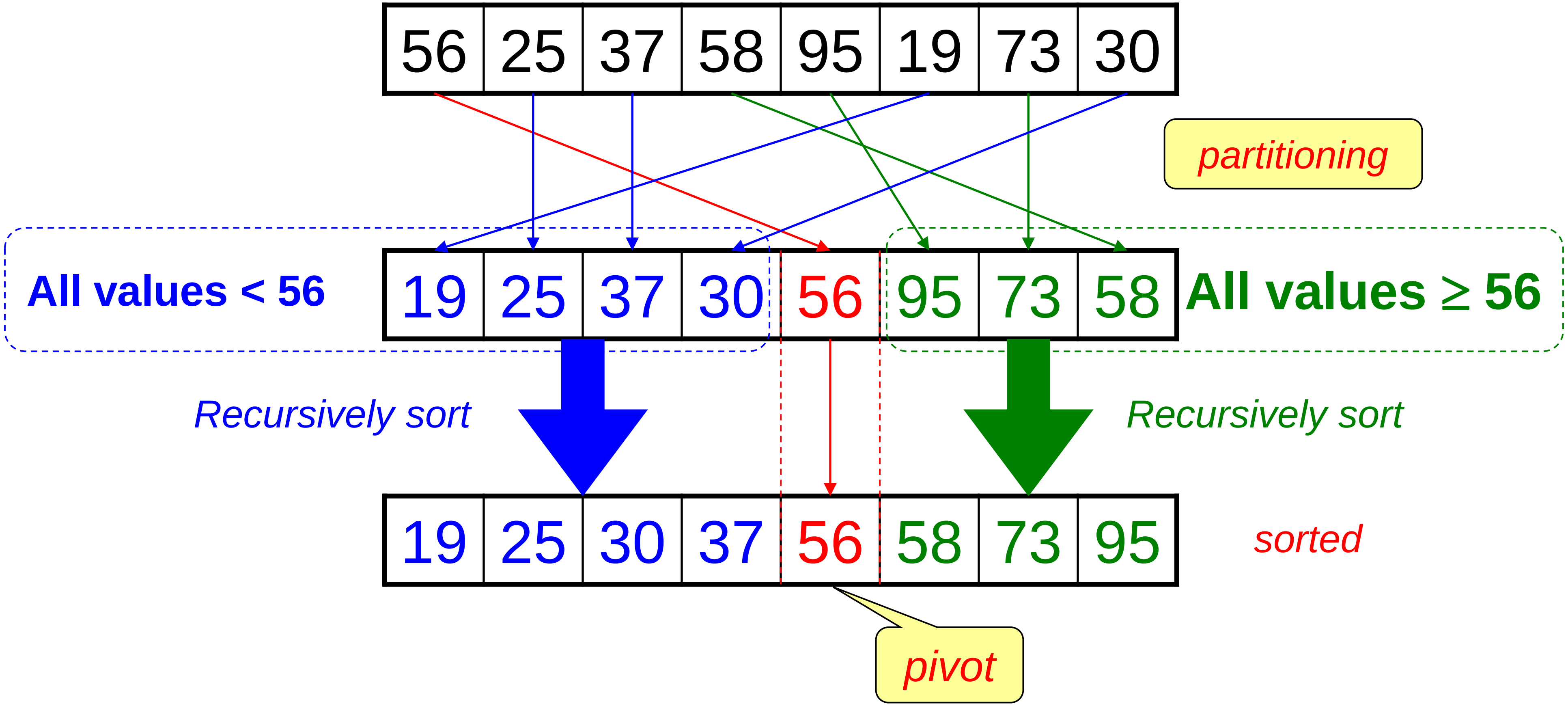
Round `i`: find the smallest element in `array[i … n-1]` and exchange it with `array[i]`.

Quickselect

- A complete sorting can always solve the problem. However, for selection sort, it is enough to stop in the middle (after k rounds).
- Selection sort can be extended to partial selection sort for selection problem. How about other sorting algorithms?
- Let's look at quicksort!

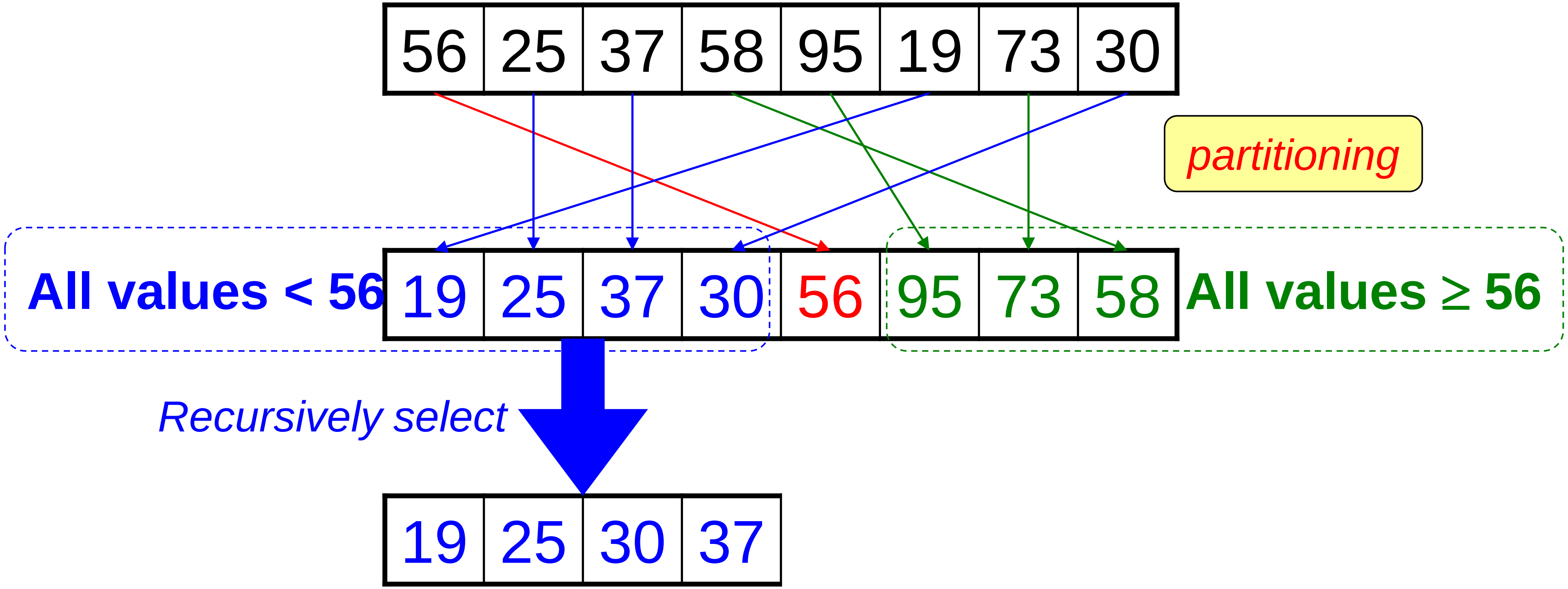
The Quicksort Algorithm

For “selection”, are both recursive calls necessary?



The Quickselect Algorithm

If k is smaller than the position of pivot (5 in this example), left part is enough!



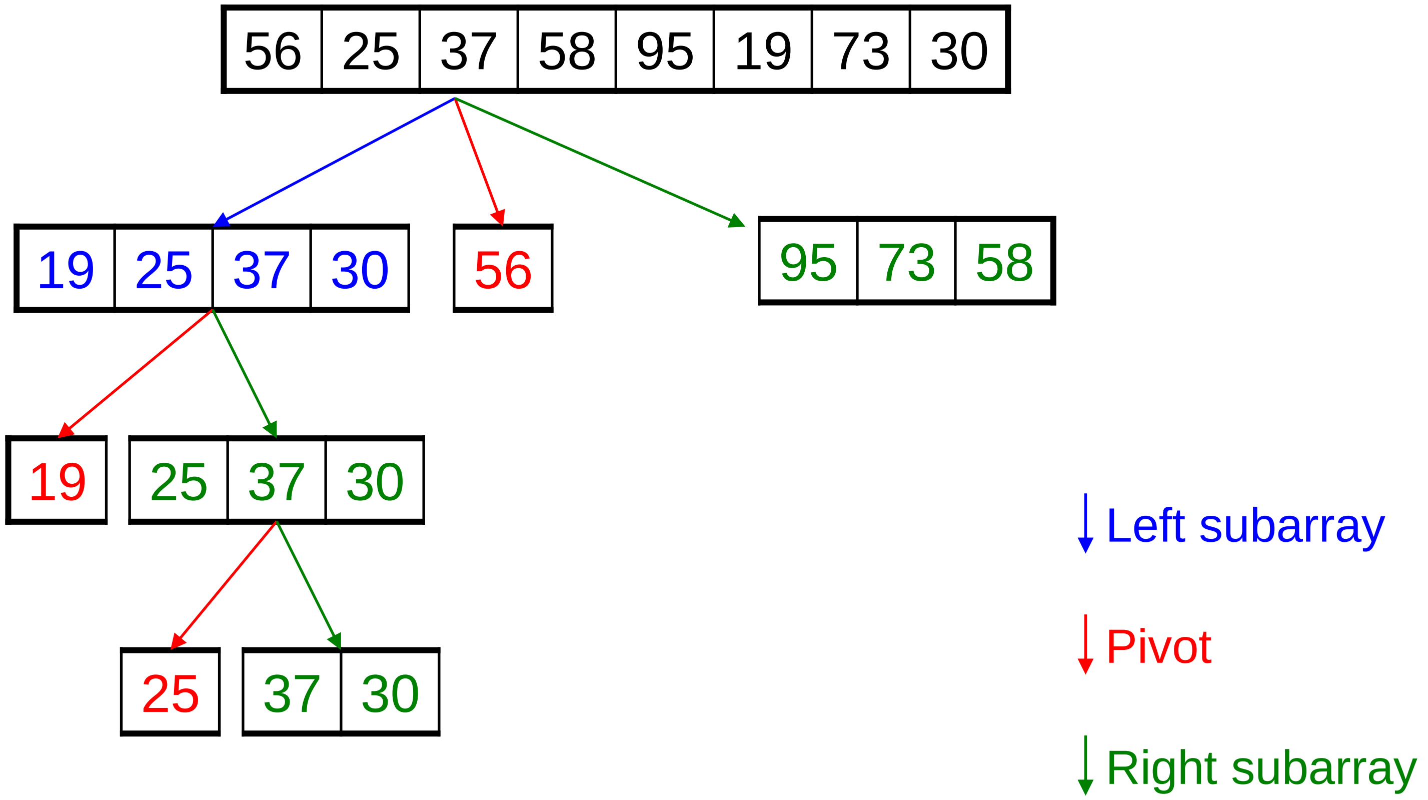
Quickselect: Example Run

find the 2-nd smallest value in this array

find the 2-nd smallest value in the left subarray

find the 1-st smallest value in the right subarray

pivot is the 1-st smallest!



Quickselect Complexity

- Computational complexity of quickselect is similar to that of quicksort.
- The time for partition is $O(n)$.
- In the *worst* case, target subarray is much larger than the other. There are $O(n)$ levels. Each level executes in $O(n)$ time. Thus, *worst case* time complexity is $O(n^2)$.
- In *average* case, the left and right subarrays have equal size.
- Strictly, *average case* time complexity $T(n) = O(n) + T(n/2)$

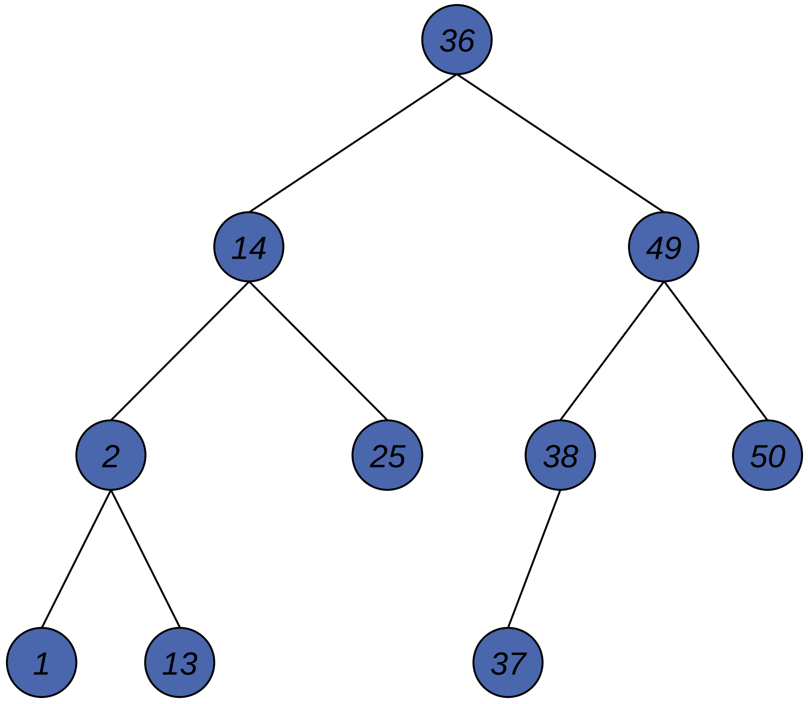
$$T(n) = O(n)$$

Dynamic Selection

- For selection,
  - What if we need to do it with several k for several times? Sort it!
  - What if we need to insert and delete elements in the set frequently?
- Augment balanced BST!
  - Balanced BST is faster than sorted/unsorted array in terms of insertion and deletion.

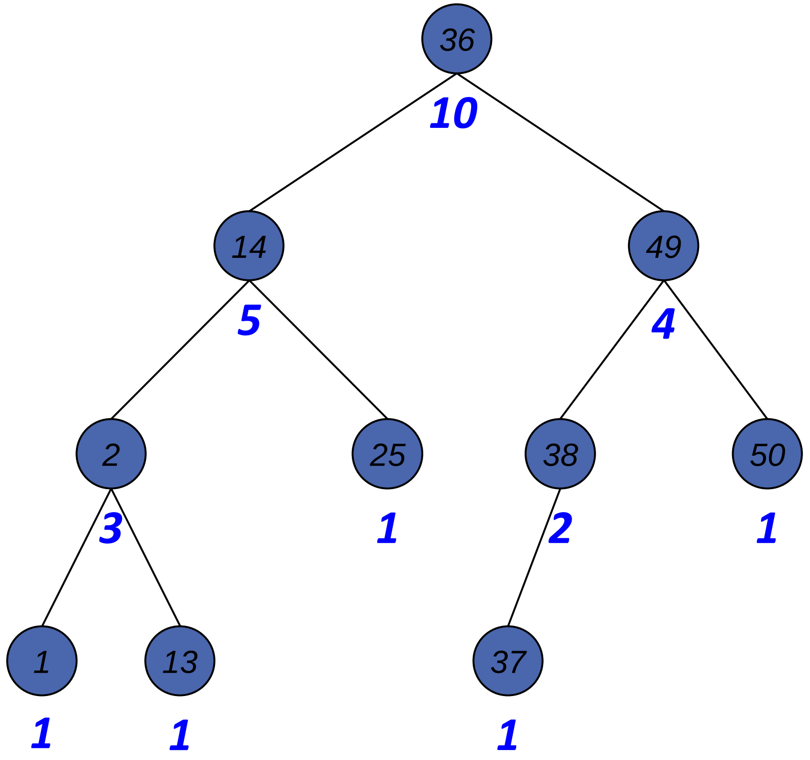
Dynamic Selection

- To find the 7th smallest element, should both subtrees of "36" be checked?
- What information will be helpful?



Dynamic Selection

- To find the 7th smallest element, should both subtrees of “36” be checked?
- What information will be helpful?
  – Size of the tree.



Exercise 2

- Implement the following function

```
int FindKthElement(bstADT t, int k)
```

- The definition of bstCDT is as follows

```
struct bstCDT {
    int root_val;
    bstADT left;
    bstADT right;
    int size;
};
```

Exercise 2 Solution

```c
int FindKthElement(bstADT t, int k){
    int i;
    if (t->left != NULL)
        i = t->left->size + 1;
    else i = 1;
    if (k == i)
        return t->root_val;
    else if (k < i)
        return FindKthElement(t->left, k);
    else
        return FindKthElement(t->right, k - i);
}
```

A *test* to stop or continue

An *end case*

a *recursive call*

Dynamic Selection

- The complexity is $O(\log n)$ (bounded by tree depth)
- Question: How to update substree sizes when inserting or deleting?
  – Try by yourself.

Q&A