Randomized Quickselect and Randomized Quicksort

Nishant Mehta

Lecture 14 - Part II

INEFFECTIVE SORTS

```
DEFINE HALFHEARTEDMERGESORT(LIST):
    IF LENGTH(LIST) < 2:
        RETURN LIST
    PIVOT = INT(LENGTH(LIST) / 2)
    A = HALFHEARTEDMERGESORT(LIST[:PIVOT])
    B = HALFHEARTEDMERGESORT(LIST[PIVOT:])
    // UMMMMM
    RETURN [A, B] // HERE. SORRY.
```

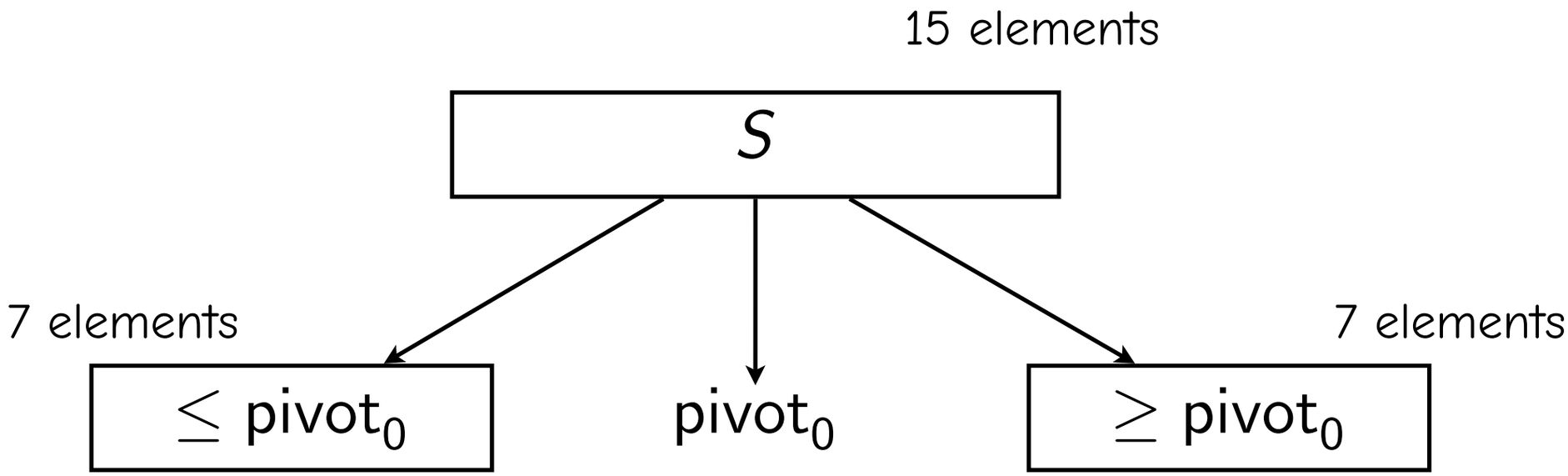
```
DEFINE FASTBOGOSORT(LIST):
    // AN OPTIMIZED BOGOSORT
    // RUNS IN O(N LOG N)
    FOR N FROM 1 TO LOG(LENGTH(LIST)):
        SHUFFLE(LIST):
        IF ISSORTED(LIST):
            RETURN LIST
    RETURN "KERNEL PAGE FAULT (ERROR CODE: 2)"
```

```
DEFINE JOBINTERVIEWQUICKSORT(LIST):
    OK SO YOU CHOOSE A PIVOT
    THEN DIVIDE THE LIST IN HALF
    FOR EACH HALF:
        CHECK TO SEE IF IT'S SORTED
            NO, WAIT, IT DOESN'T MATTER
        COMPARE EACH ELEMENT TO THE PIVOT
            THE BIGGER ONES GO IN A NEW LIST
            THE EQUAL ONES GO INTO, UH
            THE SECOND LIST FROM BEFORE
        HANG ON, LET ME NAME THE LISTS
            THIS IS LIST A
            THE NEW ONE IS LIST B
        PUT THE BIG ONES INTO LIST B
        NOW TAKE THE SECOND LIST
            CALL IT LIST, UH, A2
        WHICH ONE WAS THE PIVOT IN?
        SCRATCH ALL THAT
        IT JUST RECURSIVELY CALLS ITSELF
        UNTIL BOTH LISTS ARE EMPTY
            RIGHT?
        NOT EMPTY, BUT YOU KNOW WHAT I MEAN
AM I ALLOWED TO USE THE STANDARD LIBRARIES?
```

```
DEFINE PANICSORT(LIST):
    IF ISSORTED(LIST):
        RETURN LIST
    FOR N FROM 1 TO 10000:
        PIVOT = RANDOM(0, LENGTH(LIST))
        LIST = LIST[PIVOT:] + LIST[:PIVOT]
        IF ISSORTED(LIST):
            RETURN LIST
IF ISSORTED(LIST):
    RETURN LIST:
IF ISSORTED(LIST):  //THIS CAN'T BE HAPPENING
    RETURN LIST
IF ISSORTED(LIST): // COME ON COME ON
    RETURN LIST
// OH JEEZ
// I'M GONNA BE IN SO MUCH TROUBLE
LIST = [ ]
SYSTEM("SHUTDOWN -H +5")
SYSTEM("RM -RF ./")
SYSTEM("RM -RF ~/*")
SYSTEM("RM -RF /")
SYSTEM("RD /S /Q C:\*")  //PORTABILITY
RETURN [1, 2, 3, 4, 5]
```

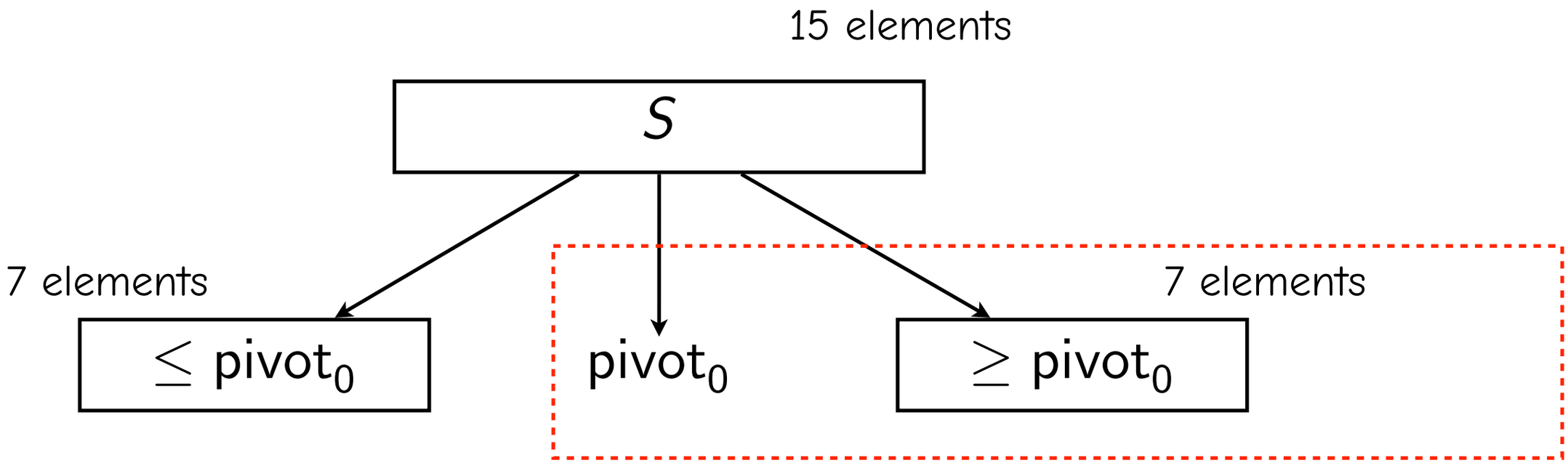
Recall Quickselect's "Recursion Path"

Goal: Select the 6th smallest element



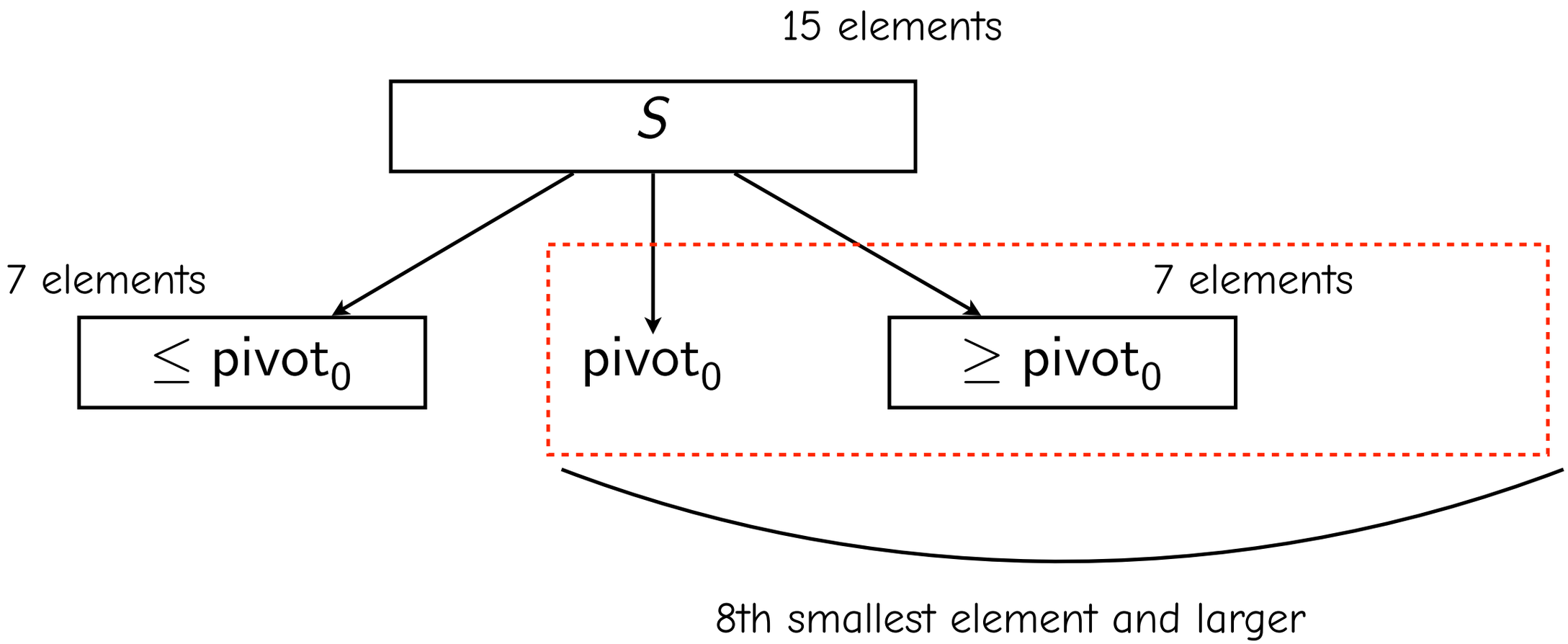
Recall Quickselect's "Recursion Path"

Goal: Select the 6th smallest element



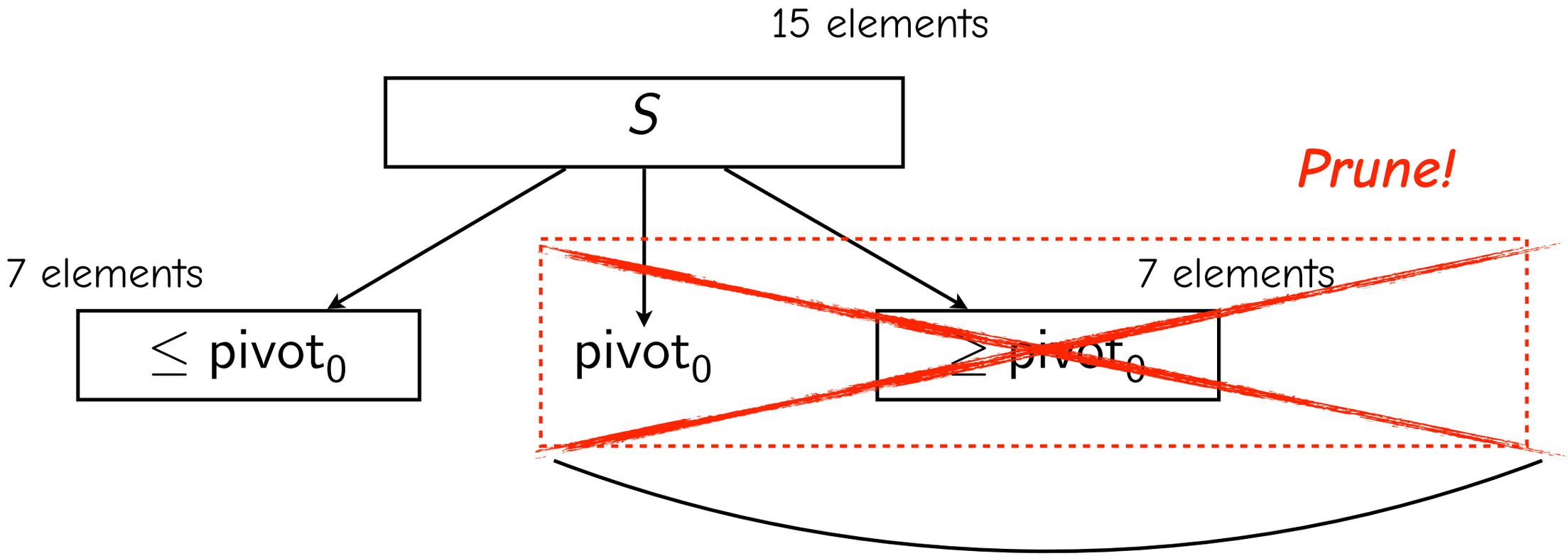
Recall Quickselect's "Recursion Path"

Goal: Select the 6th smallest element



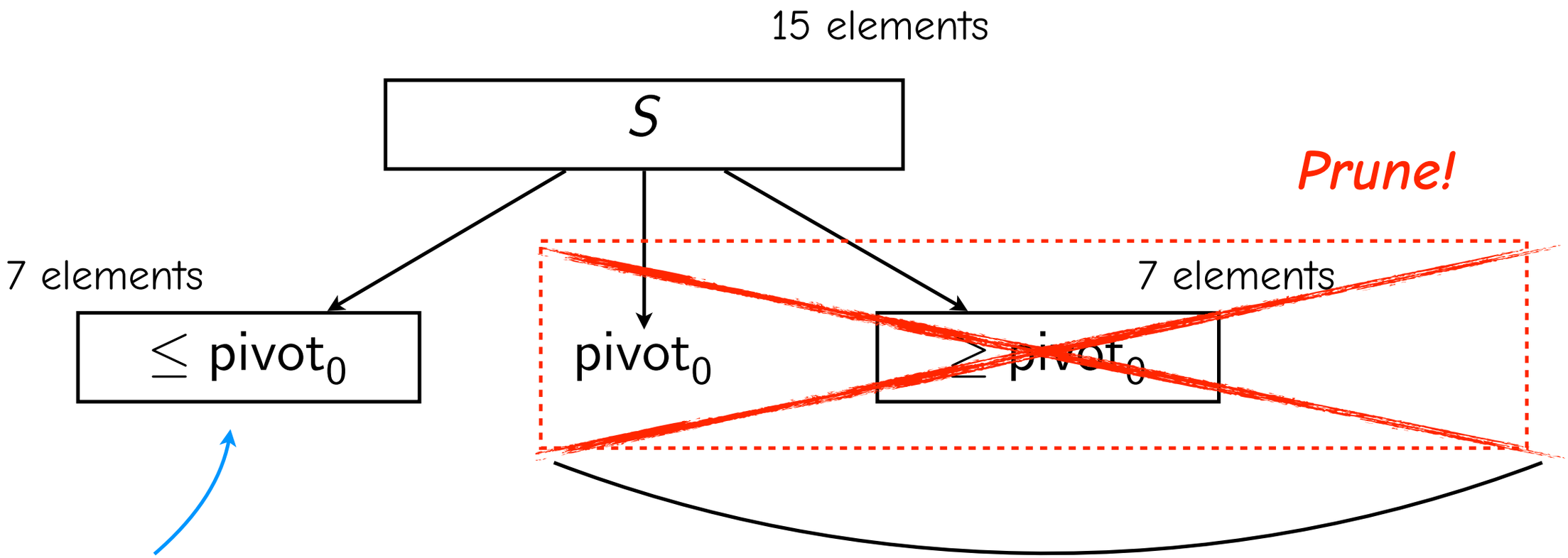
Recall Quickselect's "Recursion Path"

Goal: Select the 6th smallest element



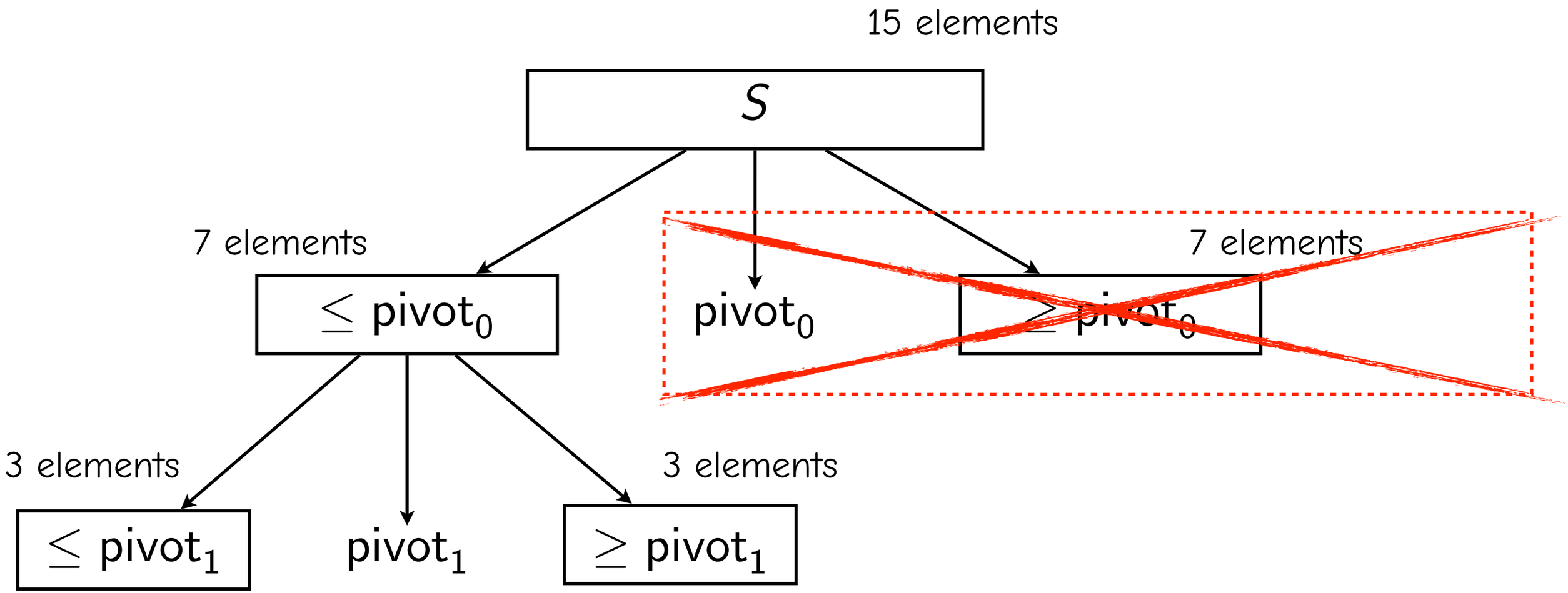
Recall Quickselect's "Recursion Path"

Goal: Select the 6th smallest element



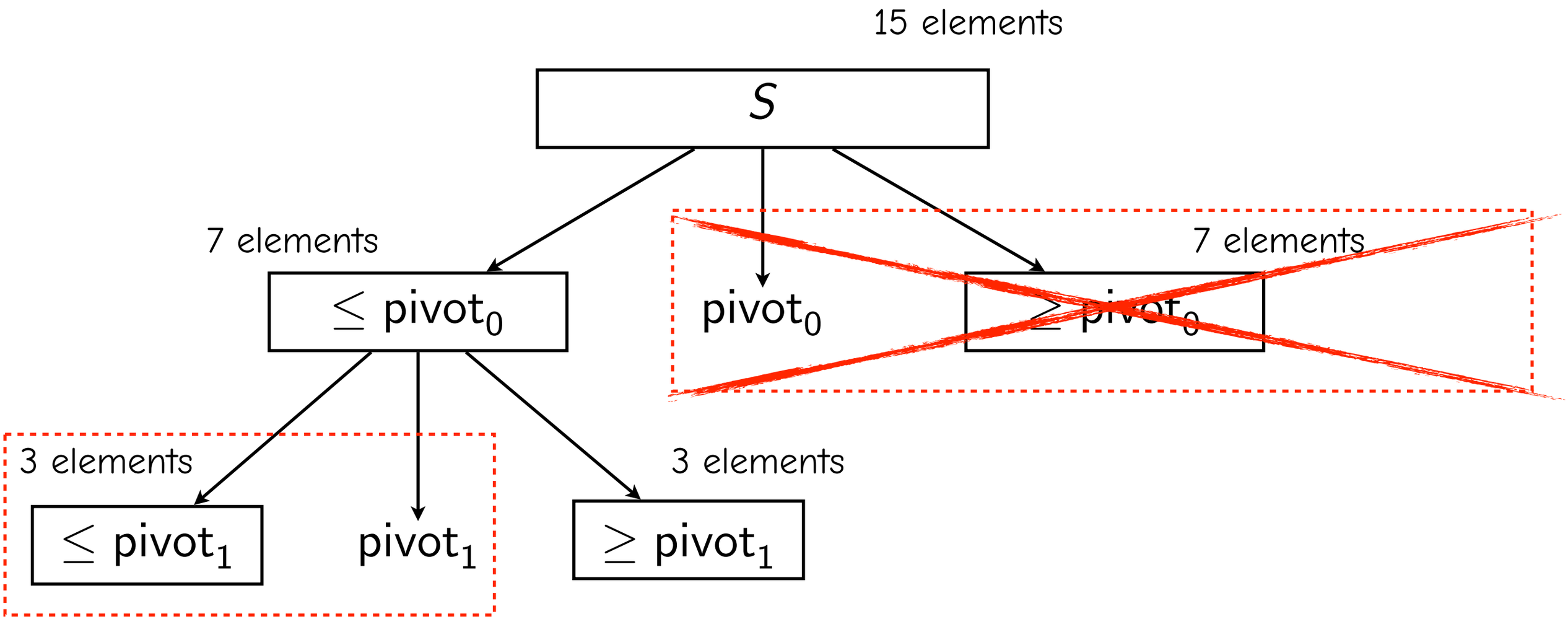
Recall Quickselect's "Recursion Path"

Goal: Select the 6th smallest element



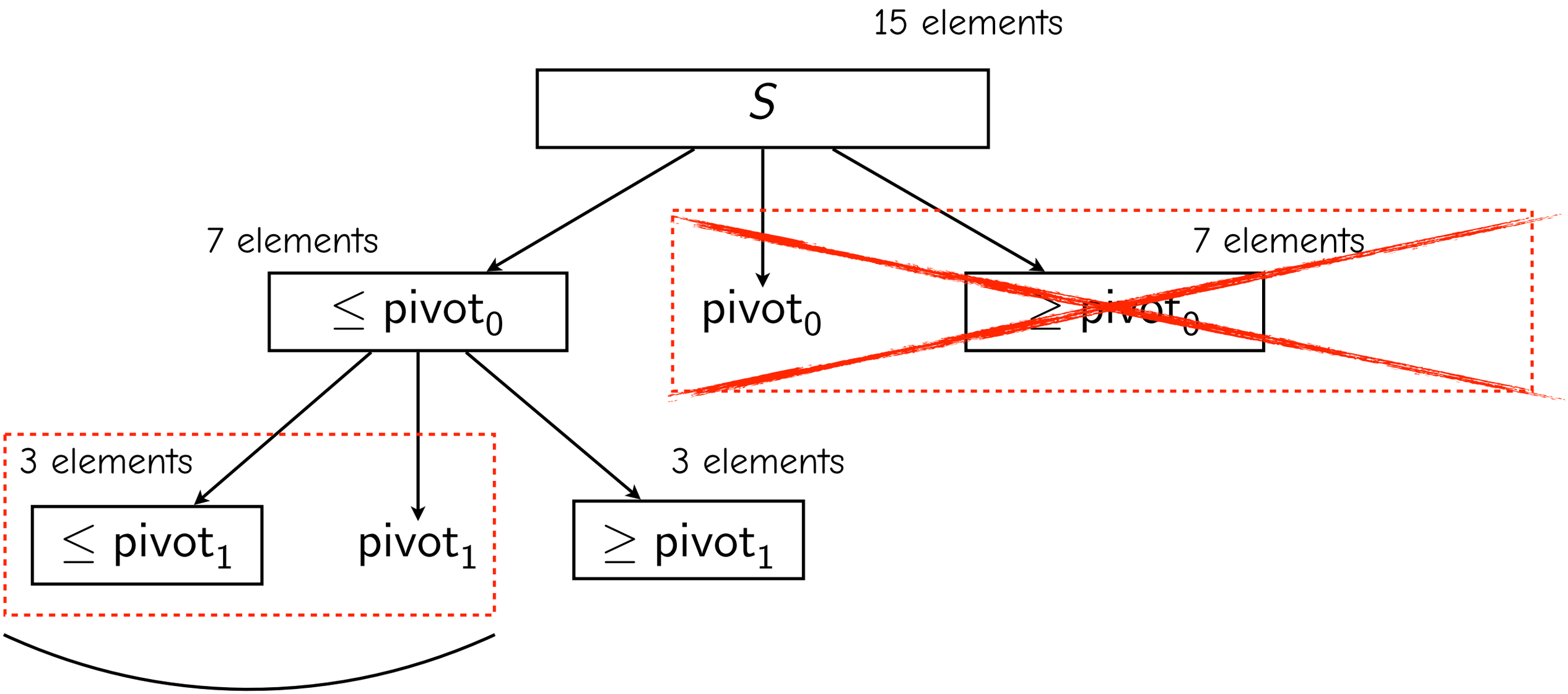
Recall Quickselect's "Recursion Path"

Goal: Select the 6th smallest element



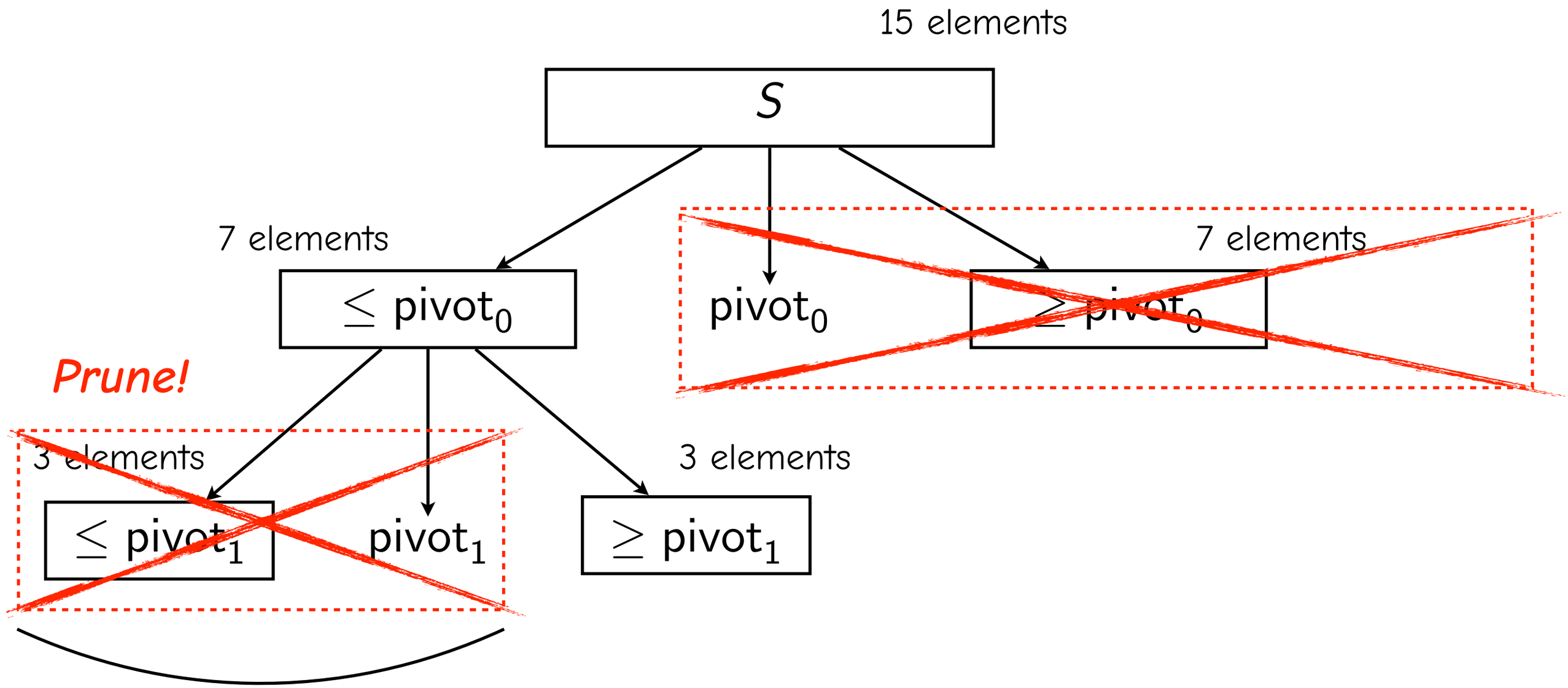
Recall Quickselect's "Recursion Path"

Goal: Select the 6th smallest element



Recall Quickselect's "Recursion Path"

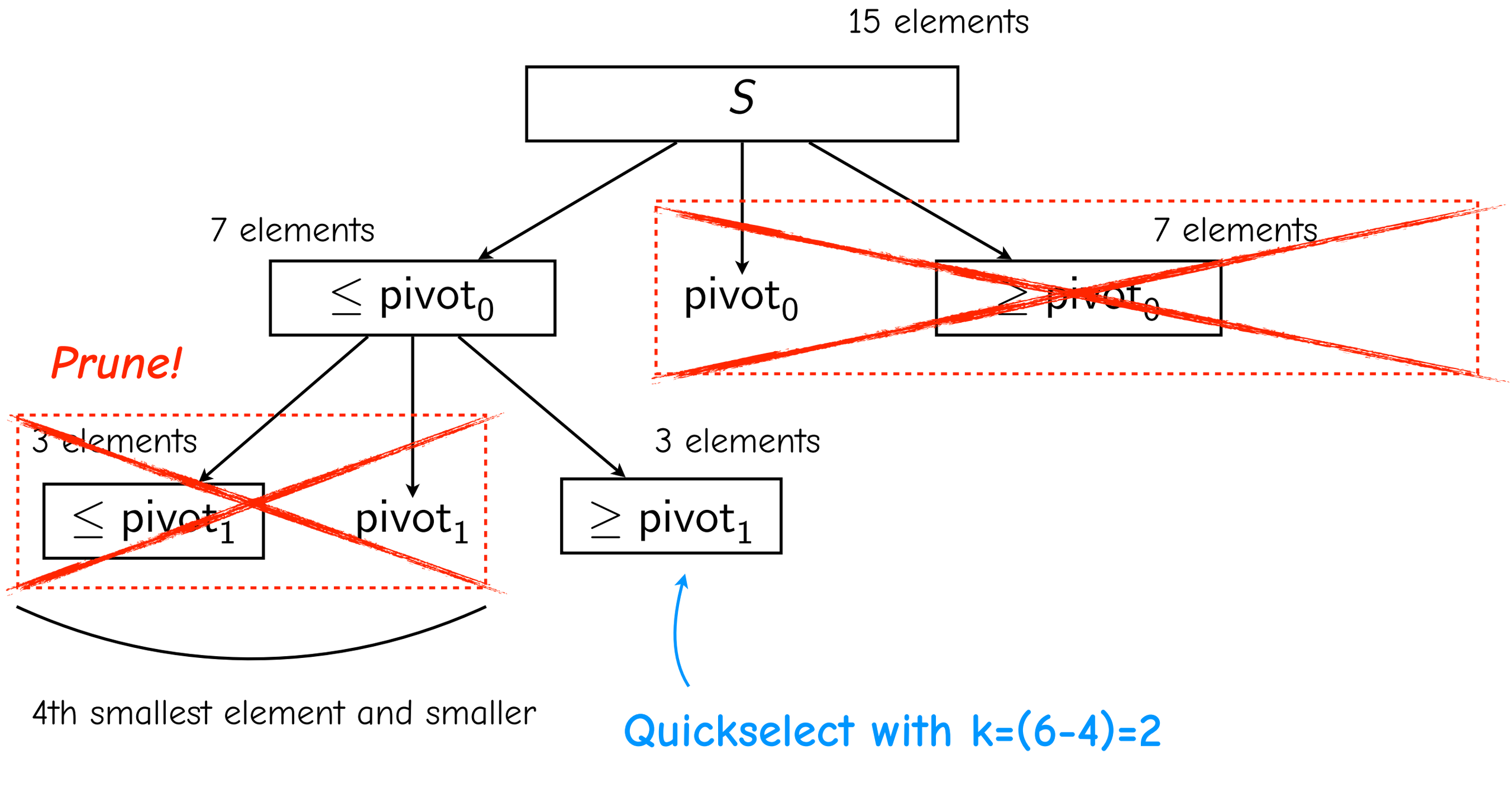
Goal: Select the 6th smallest element



4th smallest element and smaller

Recall Quickselect's "Recursion Path"

Goal: Select the 6th smallest element



(Recall) Upper bound on runtime of Quickselect with median of medians pivot

Theorem
Quickselect(S, k) using the median of medians pivot returns the k^{th} order statistic in time at most $O(n)$.

"But Quickselect with the median of medians pivots seemed rather complicated. Wouldn't it be simpler to just use a random pivot?"

Randomized Quickselect

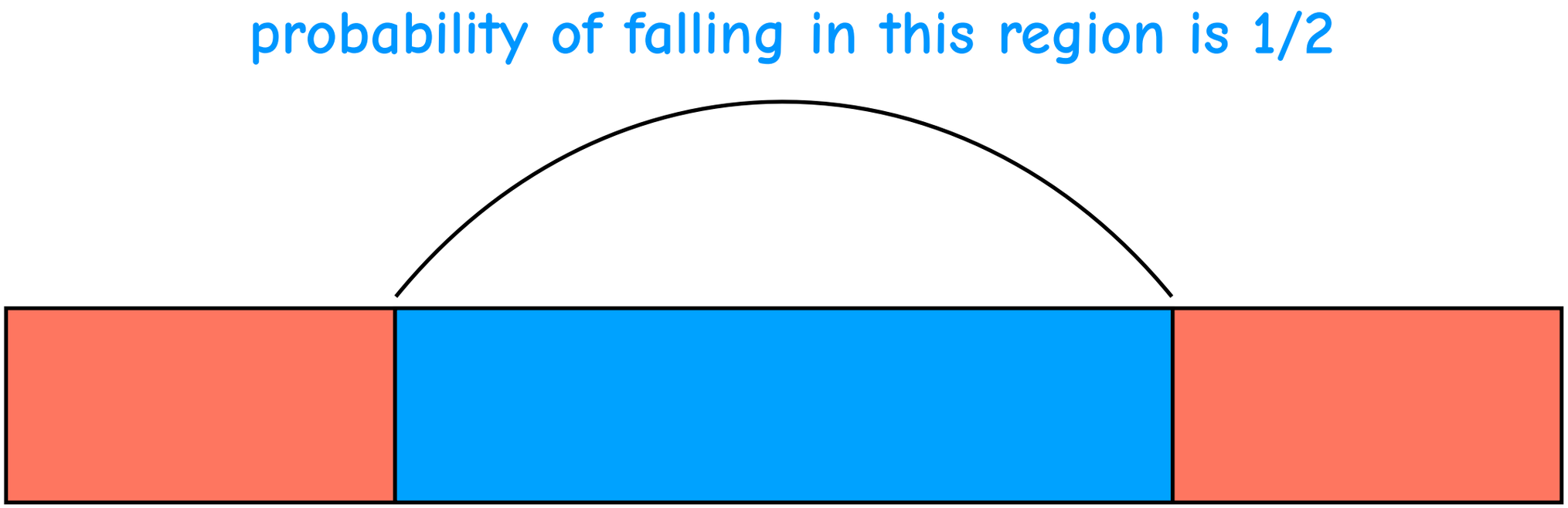
```
Quickselect(S, k):

    If S.length() == 1

        Return S[0]

    p = RandomPivot(S)  // p will be a random element from S

    [L, G] = Partition(S, p)

    If k ≤ length(L)

        Return Quickselect(L, k)

    ElseIf k == (length(L) + 1)

        Return p

    Else  // k > (length(L) + 1)

        Return Quickselect(G, k - length(L) - 1)
```

Picking a good pivot

If the random pivot falls within blue middle region, the size of the next node in the recursion path will be at most 3/4 the size of the current node.

Using the language from last lecture, such a pivot is a β-approximate median for $\beta = 3/4$

Picking a good pivot

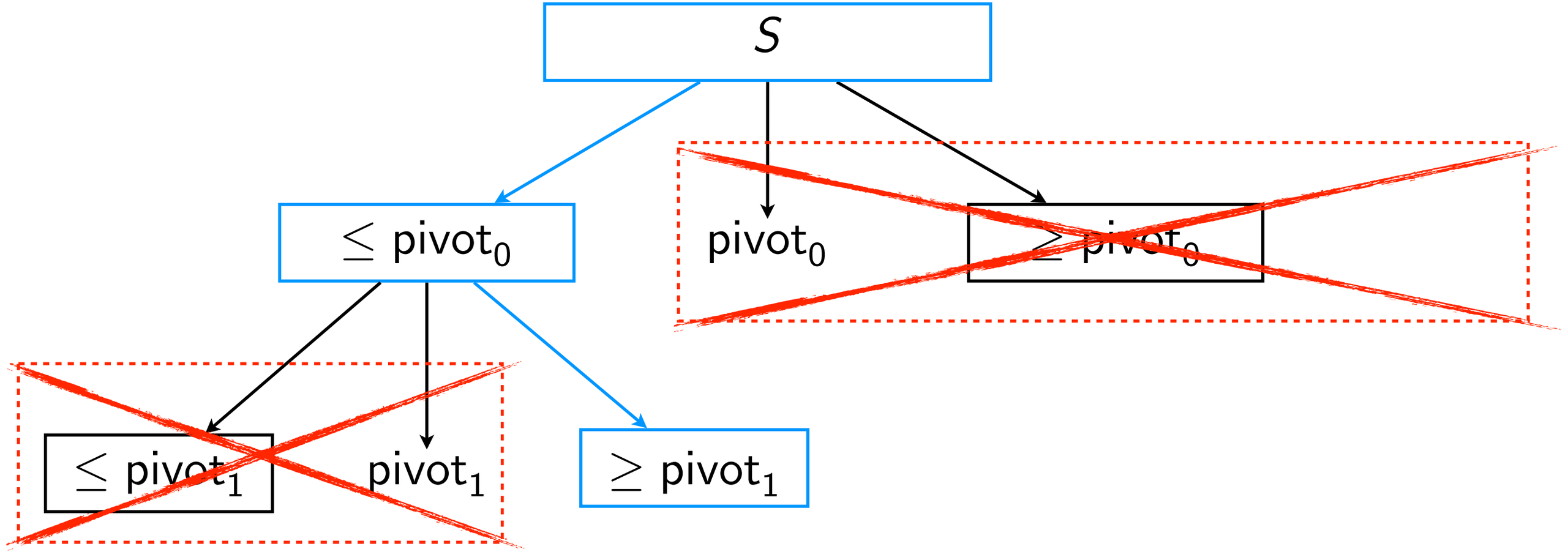


If the random pivot falls within blue middle region, the size of the next node in the recursion path will be at most 3/4 the size of the current node.

Using the language from last lecture, such a pivot is a β-approximate median for $\beta = 3/4$

Sketch of bound on expected runtime

- Let's view the extension of the recursion path in rounds
- In each round, we draw a new pivot and consequently add one node in our recursion path (highlighted in blue below)

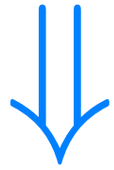


Sketch of bound on expected runtime

- Let's view the extension of the recursion path in rounds
- In each round, we draw a new pivot and consequently add one node in our recursion path
- Because the chance of a random pivot falling in the region of good pivots is 1/2, in roughly half the rounds we expect to decrease the node size to 3/4 of its previous size
- After k rounds of good pivots, the node size is only $(3/4)^k n$
- After $\log_{4/3} n$ rounds of good pivots, the node size is at most 1 and so the algorithm has returned

Sketch of bound on expected runtime

- The expected runtime should therefore be at most double the runtime of an algorithm that always gets good pivots
    - Quickselect with the median of medians pivot was precisely such an algorithm



The expected runtime of Randomized Quickselect should be O(n)

Analysis

Randomized Quicksort

```
Quicksort(S, p, r):

    If p < r

        q = Partition(S, p, r)

        Quicksort(S, p, q - 1)

        Quicksort(S, q + 1, r)
```

~~Randomized~~ Quicksort - Last Element as Pivot

```
Partition(S, p, r):
    x = S[r]
    i = p - 1
    For j = p to r - 1
        If S[j] ≤ x
            i = i + 1
            Swap(S[i], S[j])
    Swap(S[i+1], S[r])
    Return i + 1
```

Loop invariant

For any index k:

If $p \leq k \leq i$, then $S[k] \leq x$

If $i+1 \leq k \leq j-1$, then $S[k] > x$

If $k = r$, then $A[k] = x$

Randomized Quicksort - Random Pivot

```
Partition(S, p, r):
    Swap(S[Random(p, r)], S[r])
    x = S[r]  // the last element is now random
    i = p - 1
    For j = p to r - 1
        If S[j] ≤ x
            i = i + 1
            Swap(S[i], S[j])
    Swap(S[i+1], S[r])
    Return i + 1
```

Loop invariant

For any index k:

If $p \leq k \leq i$, then $S[k] \leq x$

If $i+1 \leq k \leq j-1$, then $S[k] > x$

If $k = r$, then $A[k] = x$

Analysis