Quickselect

Shusen Wang

Select the smallest element

55	8	29	68	3	41	32	12	53	17

Select the smallest element

55	8	29	68	3	41	32	12	53	17

```
int min(int arr[], int n) {
    int i = 0;
    int minVal = arr[0];
    for (i=1; i<n; i++) {
        if (arr[i] < minVal)
            minVal = arr[i];
    }
    return minVal;
}
```

Select the smallest element

55	8	29	68	3	41	32	12	53	17

```
int min(int arr[], int n) {
    int i = 0;
    int minVal = arr[0];
    for (i=1; i<n; i++) {
        if (arr[i] < minVal)
            minVal = arr[i];
    }
    return minVal;
}
```

Select the smallest element

55	8	29	68	3	41	32	12	53	17

```
int min(int arr[], int n) {
    int i = 0;
    int minVal = arr[0];
    for (i=1; i<n; i++) {
        if (arr[i] < minVal)
            minVal = arr[i];
    }
    return minVal;
}
```

Select the k-th smallest element

55	8	29	68	3	41	32	12	53	17

the 3rd smallest

Naïve Algorithms

Brute-force: $O(nk)$ time

55	8	29	68	3	41	32	12	53	17

the smallest

- $O(n)$ for finding the smallest.

Brute-force: $O(nk)$ time

55	8	29	68	~~8~~	41	32	12	53	17

the smallest

- $O(n)$ for finding the smallest.

Brute-force: $O(nk)$ time

55	8	29	68	~~3~~	41	32	12	53	17

the 2^{nd} smallest

- $O(n)$ for finding the smallest.
- $O(n)$ for finding the 2^{nd} smallest.

Brute-force: $O(nk)$ time

55	~~8~~	29	68	~~3~~	41	32	12	53	17

the 2^{nd} smallest

- $O(n)$ for finding the smallest.
- $O(n)$ for finding the 2^{nd} smallest.

Brute-force: $O(nk)$ time

55	~~8~~	29	68	~~3~~	41	32	12	53	17

the 3rd smallest

- $O(n)$ for finding the smallest.
- $O(n)$ for finding the 2nd smallest.
- $O(n)$ for finding the 3rd smallest.

Brute-force: $O(nk)$ time

55	~~8~~	29	68	~~3~~	41	32	~~12~~	53	17

the 3^{rd} smallest

- $O(n)$ for finding the smallest.
- $O(n)$ for finding the 2^{nd} smallest.
- $O(n)$ for finding the 3^{rd} smallest.

Brute-force: $O(nk)$ time

55	~~8~~	29	68	~~3~~	41	32	~~12~~	53	17

- $O(n)$ for finding the smallest.
- $O(n)$ for finding the 2nd smallest.
- $O(n)$ for finding the 3rd smallest.

$\vdots$

- $O(n)$ for finding the k-th smallest.

$O(nk)$ time in total

Sorting: $O(n \log n)$ time

55	**8**	**29**	**68**	**3**	**41**	**32**	**12**	**53**	**17**

- $O(n \log n)$ time for sorting the array.

Sorting: $O(n \log n)$ time

3	**8**	**12**	**17**	**29**	**32**	**41**	**53**	**55**	**68**
0	1	2	3	4	5	6	7	8	9

the 3rd smallest

- $O(n \log n)$ time for sorting the array.
- After sorting, the k-th smallest element is at the $(k-1)$-th position.
- $O(1)$ time for finding the k-th smallest element.

Quickselect: $O(n)$ time

Example 1: Find the 4th smallest

55	8	29	68	3	41	32	12	53	17

Step 1: Picking a pivot.

- Heuristic:

$$\text{pivot} = \text{median}(\text{left}, \text{center}, \text{right})$$

Example 1: Find the 4th smallest

55	8	29	68	3	$\boxed{41}$	32	12	53	17
left					center				right

Pick it as pivot!

Step 1: Picking a pivot.

- Heuristic:

$$\text{pivot} = \text{median}(\text{left}, \text{center}, \text{right})$$

Example 1: Find the 4th smallest

12	8	29	32	3	17	41	55	53	68

Group 1: $\{x \mid x \leq 41\}$.

Group 2: $\{x \mid x \geq 41\}$.

Step 2: Partition.

Example 1: Find the 4^{th} smallest

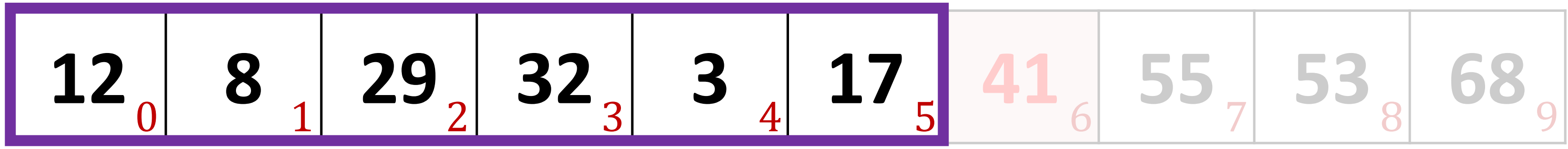
12	8	29	32	3	17	41	55	53	68
0	1	2	3	4	5	6	7	8	9

The 4^{th} smallest is on the left

`pivot is the 7th smallest`

Step 2: Partition.

Example 1: Find the 4^{th} smallest



The 4^{th} smallest is on the left

Step 3: Recursion.

- Search the 4^{th} smallest in the left part.

Example 2: Find the 9th smallest

0	1	2	3	4	5	6	7	8	9
12	8	29	32	3	17	41	55	53	68

pivot is the 7th smallest

Step 3: Recursion.

Example 2: Find the 9th smallest

12	8	29	32	3	17	41	**55**	**53**	**68**
0	1	2	3	4	5	6	7	8	9

The 9th smallest is on the right

Step 3: Recursion.

- Find the 9th smallest element.
- Pivot is the 7th smallest element.
- Search the $9 - 7 = 2$ smallest in the right part.

Example 3: Find the 7th smallest

12	8	29	32	3	17	**41**	55	53	68
0	1	2	3	4	5	6	7	8	9

pivot is the 7th smallest

Example 3: Find the 7th smallest

12 0	**8** 1	**29** 2	**32** 3	**3** 4	**17** 5	**41** 6	**55** 7	**53** 8	**68** 9

Found!

Step 3: Return the value of pivot.

Summary of Quickselect

Inputs: `arr` (array) and k (select the k-th smallest).

Summary of Quickselect

Inputs: `arr` (array) and k (select the k-th smallest).

1. Pick a pivot.
2. Partition the array into two parts (left and right).
   - Suppose the pivot is at the i-th position
   - Let $p = i + 1$. (Pivot is the p-th smallest element.)

Summary of Quickselect

Inputs: `arr` (array) and k (select the k-th smallest).

1. Pick a pivot.
2. Partition the array into two parts (left and right).
   - Suppose the pivot is at the i-th position
   - Let $p = i + 1$. (Pivot is the p-th smallest element.)
3. Recursion:
   - If $k == p$ ==> Return the value of pivot, i.e., `arr[i]`.
   - If $k < p$ ==> Find the k-th smallest in the left part.
   - If $k > p$ ==> Find the $(k - p)$th smallest in the right part.

Time Complexity

Time Complexity (Simplified)

Assume pivot is the median.

- $T(n)$: Time complexity for size-n array.
- Sizes of the left and right parts are both $\frac{n}{2}$.
- Recurrence relation:

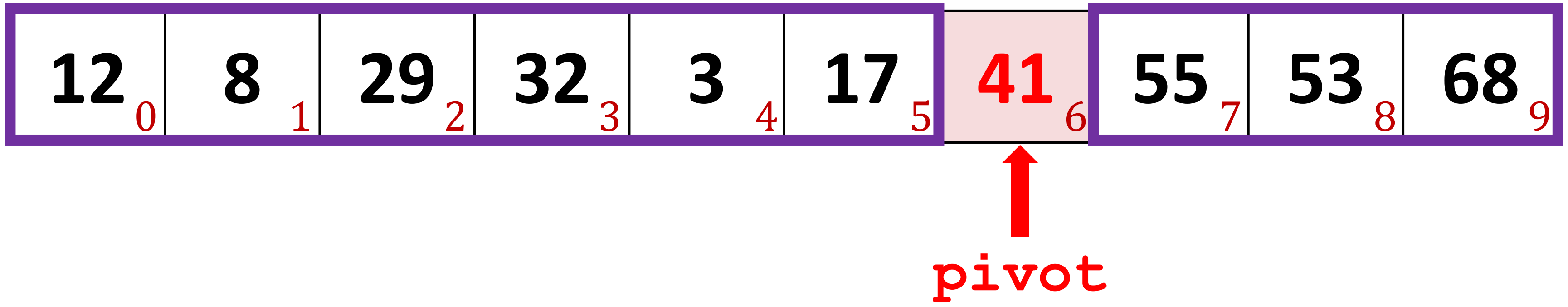
$$T(n) \;=\; T(n/2) \;+\; c\,n.$$

- ➔ $T(n) = O(n)$.

Time Complexity (Average Case)

Assume the data is randomly shuffled.

- The pivot's position can be any of $\{0, 1, 2, \cdots, n-1\}$ (with equal probability).



Time Complexity (Average Case)

Assume the data is randomly shuffled.

- The pivot's position can be any of $\{0, 1, 2, \cdots, n-1\}$ (with equal probability).
- The expected time complexity is $O(n)$.

Thank You!

Implementation

```
int select(int arr[], int left, int right, int k) {
     if (left+10 > right) { // for short array
           return naiveAlgorithm(arr, left, right, k);
     }
     else { // for long array
          int j = selectpivot(arr, left, right);// pivot position
          swap(arr, j, right-1); //put pivot in the end
          int i = partition(arr, left, right);
          swap(arr, i, right-1); // restore pivot
          // now, pivot is at the i-th position
          int p = i + 1; // pivot is the p-th smallest
          if (k == p) return arr[i];
          if (k < p) return select(arr, left, i - 1, k);
          if (k > p) return select(arr, i + 1, right, k-p);
     }
}
```

```
int select(int arr[], int left, int right, int k) {
    if (left+10 > right) { // for short array
        return naiveAlgorithm(arr, left, right, k);
    }
    else { // for long array
        int j = selectpivot(arr, left, right);// pivot position
        swap(arr, j, right-1); //put pivot in the end
        int i = partition(arr, left, right);
        swap(arr, i, right-1); // restore pivot
        // now, pivot is at the i-th position
        int p = i + 1; // pivot is the p-th smallest
        if (k == p) return arr[i];
        if (k < p) return select(arr, left, i - 1, k);
        if (k > p) return select(arr, i + 1, right, k-p);
    }
}
```

```
int select(int arr[], int left, int right, int k) {
      if (left+10 > right) { // for short array
            return naiveAlgorithm(arr, left, right, k);
      }
      else { // for long array
            int j = selectpivot(arr, left, right);// pivot position
            swap(arr, j, right-1); //put pivot in the end
            int i = partition(arr, left, right);
            swap(arr, i, right-1); // restore pivot
            // now, pivot is at the i-th position
            int p = i + 1; // pivot is the p-th smallest
            if (k == p) return arr[i];
            if (k < p) return select(arr, left, i - 1, k);
            if (k > p) return select(arr, i + 1, right, k-p);
      }
}
```

```
int select(int arr[], int left, int right, int k) {
    if (left+10 > right) { // for short array
        return naiveAlgorithm(arr, left, right, k);
    }
    else { // for long array
        int j = selectpivot(arr, left, right);// pivot position
        swap(arr, j, right-1); //put pivot in the end
        int i = partition(arr, left, right);
        swap(arr, i, right-1); // restore pivot
        // now, pivot is at the i-th position
        int p = i + 1; // pivot is the p-th smallest
        if (k == p) return arr[i];
        if (k < p) return select(arr, left, i - 1, k);
        if (k > p) return select(arr, i + 1, right, k-p);
    }
}
```

Proof of the simplified case

Time Complexity (Simplified)

- Time complexity: $T(n) \;=\; T\left(\frac{n}{2}\right) \;+\; c\,n.$
- Thus, $T(n) \;=\; T\left(\frac{n}{2}\right) \;+\; c\,n$

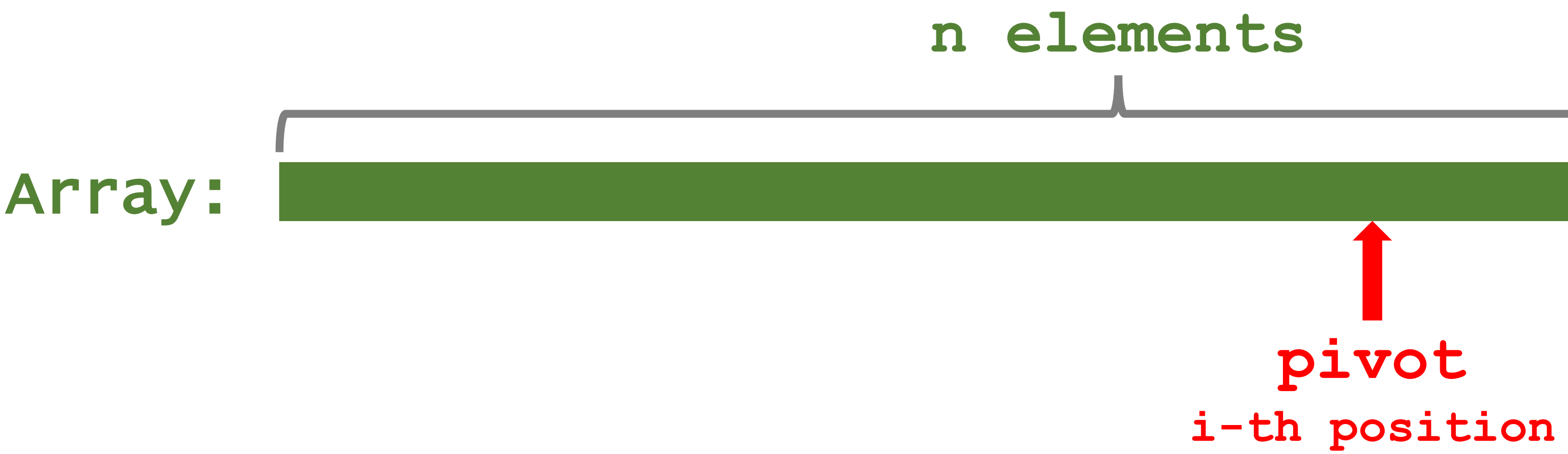
$$
\begin{aligned}
&= T\left(\frac{n}{4}\right) + \frac{cn}{2} + cn \\
&= T\left(\frac{n}{8}\right) + \frac{cn}{4} + \frac{cn}{2} + cn \\
&= \cdots \\
&= c \cdot \left(1 + 2 + 4 + 8 + \cdots + \frac{n}{4} + \frac{n}{2} + n\right) \\
&= c(2n - 1).
\end{aligned}
$$

Proof of the average-case time complexity

Time Complexity (Average Case)

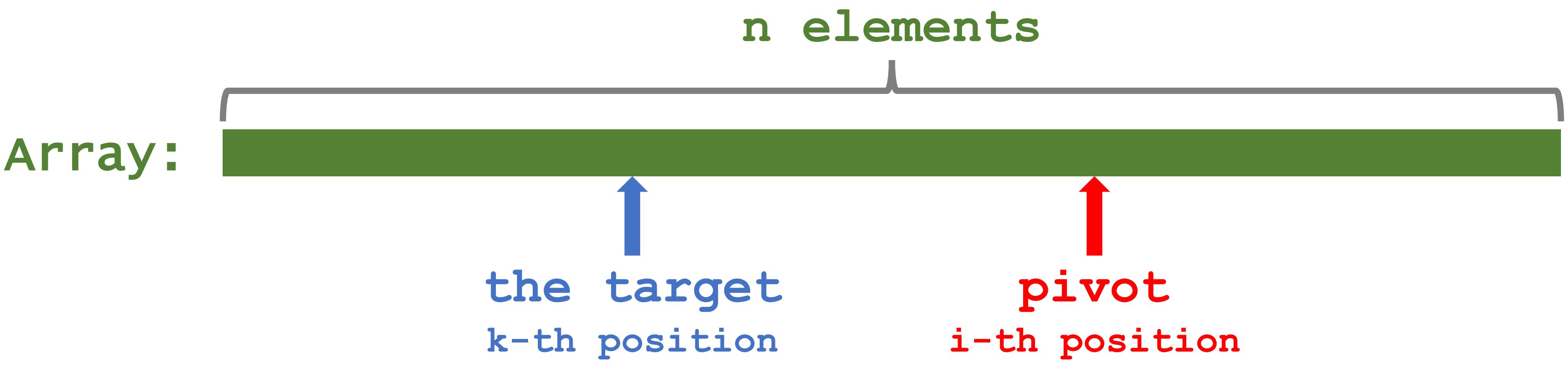
Assume the data is randomly shuffled.

- Let p_i be the probability that the pivot is at the i-th position.
- Assume $p_0 = p_1 = p_2 = \cdots = p_{n-1} = \frac{1}{n}$.



Time Complexity (Average Case)

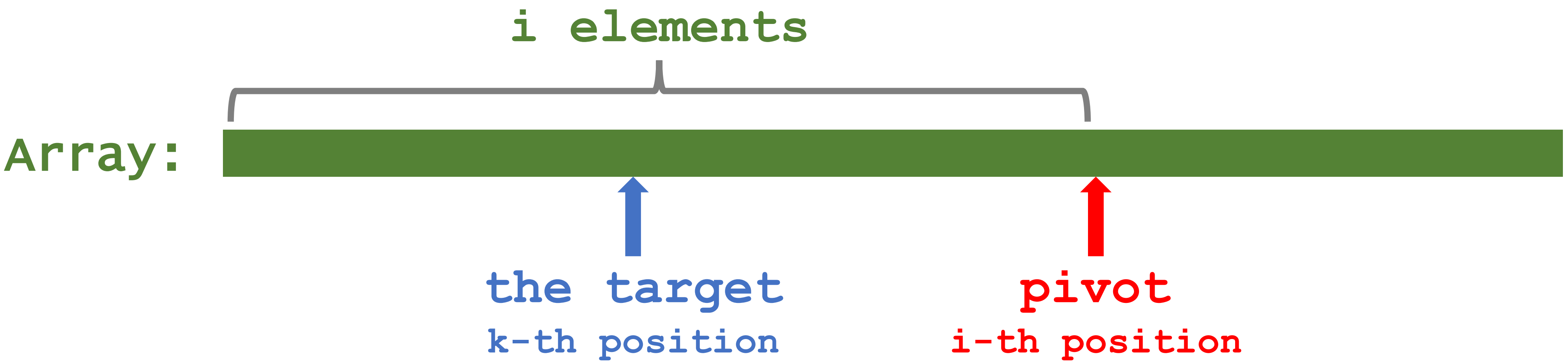
- The pivot is in the left of the target: $\mathbb{P}[i < k] = \frac{k}{n}$.
- The pivot is in the right of the target: $\mathbb{P}[i > k] = \frac{n-k}{n}$.



Time Complexity (Average Case)

- $T(n) = \boxed{\frac{n-k}{n} \cdot T(i)} + \frac{k}{n} \cdot T(n-i) + cn.$

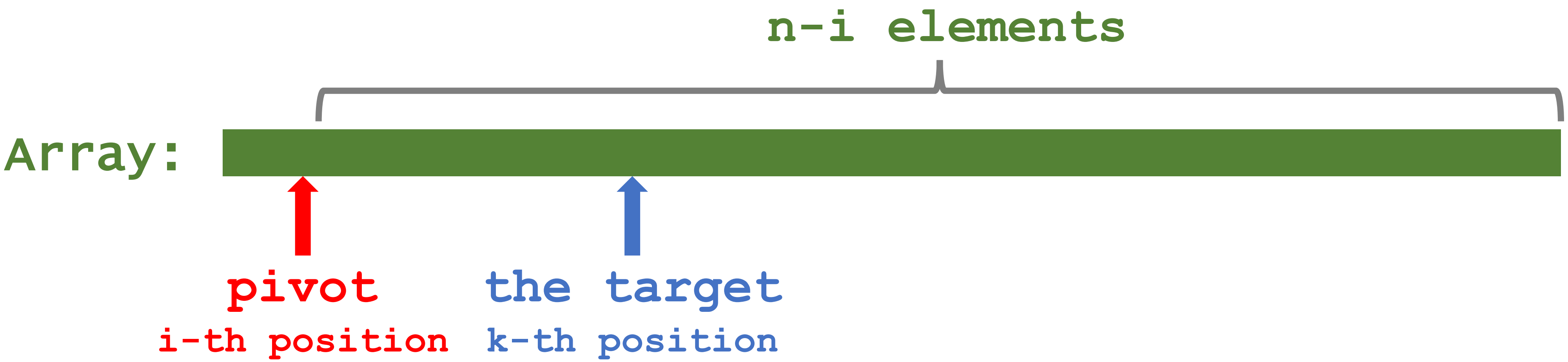
The pivot happens to be in the right.



Time Complexity (Average Case)

- $T(n) = \frac{n-k}{n} \cdot T(i) + \boxed{\frac{k}{n} \cdot T(n-i)} + cn.$

The pivot happens to be in the left.



Time Complexity (Average Case)

- $T(n) = \frac{n-k}{n} \cdot T(i) + \frac{k}{n} \cdot T(n-i) + \boxed{cn.}$

Time of partition

Time Complexity (Average Case)

- $T(n) \; = \; \frac{n-k}{n} \cdot T(i) + \; \frac{k}{n} \cdot T(n-i) \; + \; cn.$
- $\mathbb{E}[T(n)] = cn + \sum_{i=0}^{n-1} p_i \cdot \left[\frac{n-k}{n} \cdot T(i) + \; \frac{k}{n} \cdot T(n-i)\right]$

$$= cn + \sum_{i=0}^{n-1} \frac{1}{n} \cdot \left[\frac{n-k}{n} \cdot T(i) + \; \frac{k}{n} \cdot T(n-i)\right]$$

$$= cn + \frac{n-k}{n} \cdot \sum_{i=0}^{n-1} \frac{1}{n} \cdot T(i) + \frac{k}{n} \cdot \sum_{i=0}^{n-1} \frac{1}{n} \cdot T(n-i)$$

$$= cn + \frac{1}{n} \cdot \sum_{i=0}^{n-1} T(i).$$

Time Complexity (Average Case)

- $T(n) = \frac{n-k}{n} \cdot T(i) + \frac{k}{n} \cdot T(n-i) + cn.$
- $\mathbb{E}[T(n)] = cn + \sum_{i=0}^{n-1} p_i \cdot \left[\frac{n-k}{n} \cdot T(i) + \frac{k}{n} \cdot T(n-i)\right]$

$$= cn + \sum_{i=0}^{n-1} \frac{1}{n} \cdot \left[\frac{n-k}{n} \cdot T(i) + \frac{k}{n} \cdot T(n-i)\right]$$

$$= cn + \frac{n-k}{n} \cdot \sum_{i=0}^{n-1} \frac{1}{n} \cdot T(i) + \frac{k}{n} \cdot \sum_{i=0}^{n-1} \frac{1}{n} \cdot T(n-i)$$

$$= cn + \frac{1}{n} \cdot \sum_{i=0}^{n-1} T(i).$$

If $T(n) = 2cn,$ then the two sides are equal.