Quick sort

Quick Sort Idea

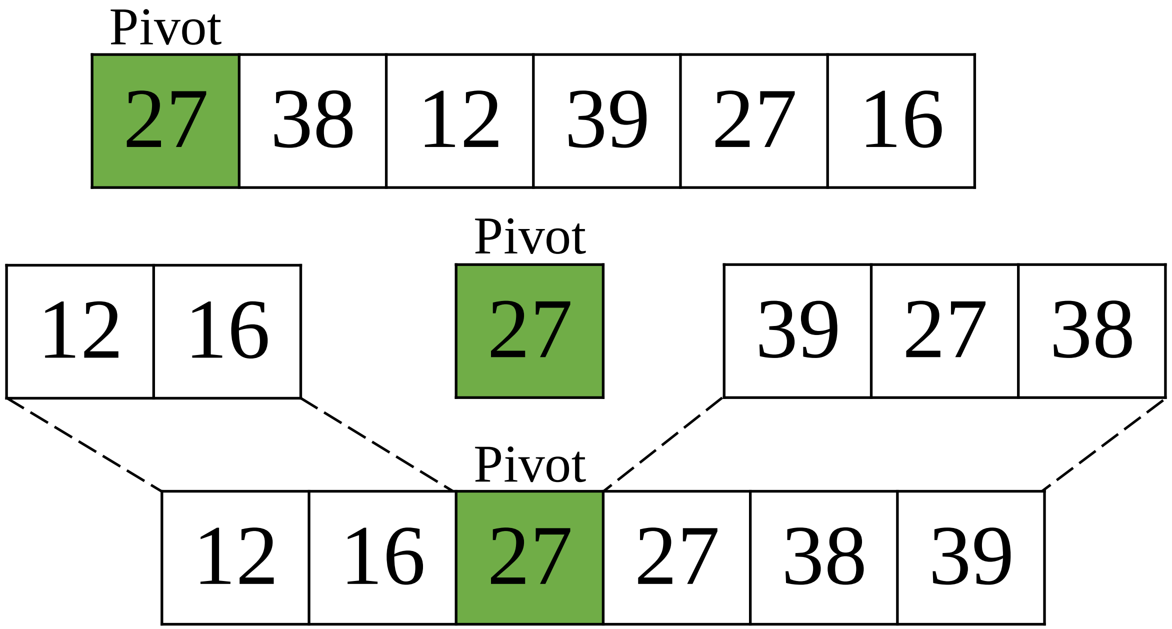
- Quick Sort is a divide-and-conquer algorithm:
  - Divide step:
    - Choose an item p (known as pivot) and partition the items of $a[i \dots j]$ into two parts:
      - Items that are smaller than p.
      - Items that are greater than or equal to p.
    - Recursively sort the two parts.
  - Conquer step:
    - Do nothing!
- In comparison, Merge Sort spends most of the time in conquer step but very little time in divide step.

Quick Sort
Divide Step Example

Choose first element as pivot

Partition **arr** about the pivot 27

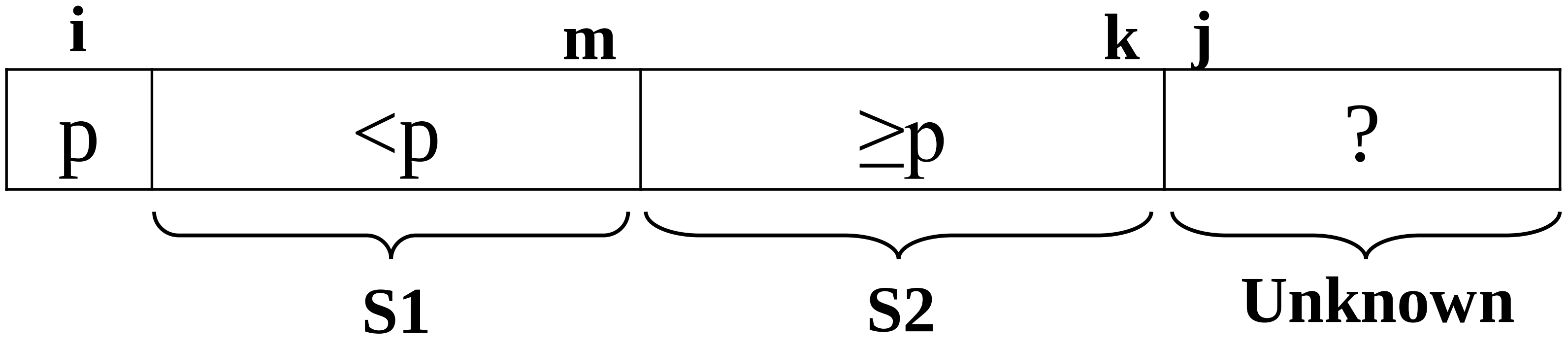
Recursively sort the two parts



Notice anything special about the position of pivot in the final sorted items?

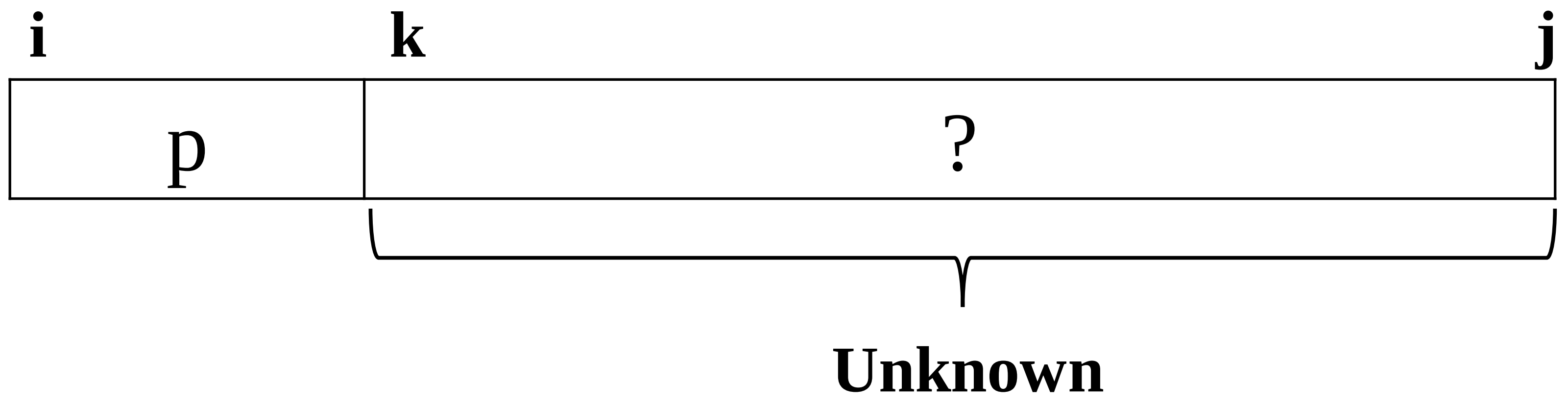
Quick Sort
Partition Algorithm

- To partition $a[i \ldots j]$, we choose $a[i]$ as the pivot p.
  - Why choose $a[i]$? Are there other choices?
- The remaining items (i.e. $a[i + 1 \ldots j]$) are divided into 3 regions:
  - $S1 = a[i + 1 \ldots m]$ where items $< p$.
  - $S2 = a[m + 1 \ldots k]$ where item $\geq p$.
  - Unknown (unprocessed) = $a[k + 1 \ldots j]$, where items are yet to be assigned to $S1$ or $S2$.



Quick Sort
Partition Algorithm

- Initially, regions $S1$ and $S2$ are empty:
  - All items excluding p are in the unknown region.
- For each item $a[k]$ in the unknown region:
  - Compare $a[k]$ with p:
    - If $a[k] \;>=\; p$, put it into $S2$.
    - Otherwise, put $a[k]$ into $S1$.



Quick Sort
Partition Algorithm

- Case 1: if a[k] >= p:

- If a[k]=y >= p:

	S1	S2	
p	< p	x $\geq$ p	y ?
i	m		k j

- Increment k:

	S1	S2	
p	< p	x $\geq$ p y	?
i	m		k j

Quick Sort
Partition Algorithm

- Case 2: if $a[k] < p$

- If $a[k]=y < p$:

	S1	S2	
p	$< p$	x $\geq p$	y ?
i	m		k j

Quick Sort
Partition Algorithm

- Case 2: if a[k] < p

- Increment m:

	S1	**S2**	
p	< p x	≥ p	y ?
i	**m**		**k** **j**

Quick Sort
Partition Algorithm

- Case 2: if a[k] < p

- Swap x and y:

	S1	**S2**	
p	< p y	≥ p	x ?
i	**m**		**k** **j**

Quick Sort
Partition Algorithm

- Case 2: if a[k] < p

- Increment k:

	S1		S2		
p	< p	y	$\geq$ p	x	?
i		m		k	j

Quick Sort
Partition Example

Pivot	Unknown				
27	38	12	39	27	16

We take as pivot first element, and compare is with first element from Unknown section.

Quick Sort
Partition Example

Pivot	S_2	Unknown			
27	38	12	39	27	16

- As the first element from Unknown section is greater then out pivot, we simply increment unknown section pointer, and 38 is become first element of $S2$ section.

Quick Sort
Partition Example

Pivot	S_2	Unknown			
27	38	12	39	27	16

- Then we continue with the following element from Unknown section, and swap it with 38 to create $S1$ section

Quick Sort
Partition Example

Pivot	S_1	S_2	Unknown		
27	12	38	39	27	16

- Then we continue with the following element from Unknown section, and swap it with 38 to create $S1$ section

Quick Sort
Partition Example

Pivot	S_1	S_2		Unknown	
27	12	38	39	27	16

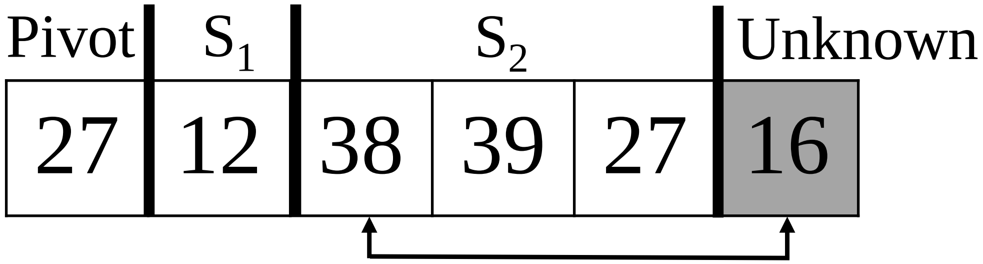
Now we take the first element from Unknown section again, and increment S2 section.

Quick Sort
Partition Example

Pivot	S_1		S_2		Unknown
27	12	38	39	27	16

Now we take the first element from Unknown section again, and increment S2 section.

Quick Sort
Partition Example



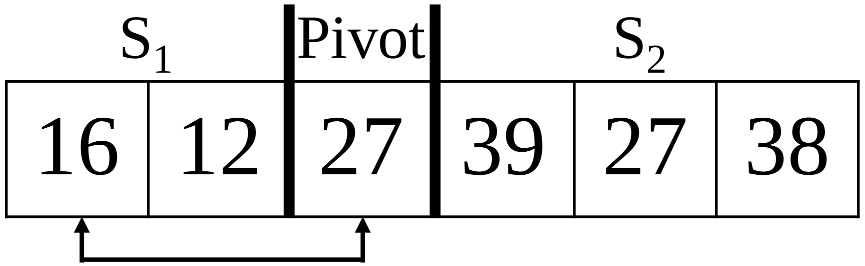
- Now we take the last element from Unknown section, and swap it with first element from $S2$ section.

Quick Sort
Partition Example

Pivot	S_1		S_2		
27	12	16	39	27	38

- Now we take the last element from Unknown section, and swap it with first element from $S2$ section.

Quick Sort
Partition Example



- And swapping pivot with last element of $S1$, to place it between $S1$ and $S2$.

Quick Sort Implementation

```
void quick_sort(vector<int>& arr, int i, int j)
{
    if (i == j)
        return;
    int pivot = partition(arr, i, j);
    quick_sort(arr, i, pivot);
    quick_sort(arr, pivot + 1, j);
}

void quick_sort(vector<int>& arr)
{
    quick_sort(arr, 0, arr.size());
}
```

Quick Sort
Partition implementation

```cpp
int partition(vector<int>& arr, int i, int j)
{
    int pivot = i;
    int small_index = i;
    int big_index = i;
    for (int k = i + 1; k < j; ++k)
    {
        if (arr[k] >= arr[pivot])
        {
            ++big_index;
        }
        else
        {
            ++small_index;
            swap(arr[small_index], arr[k]);
            ++big_index;
        }
    }
    swap(arr[pivot], arr[small_index]);
    pivot = small_index;
    return small_index;
}
```

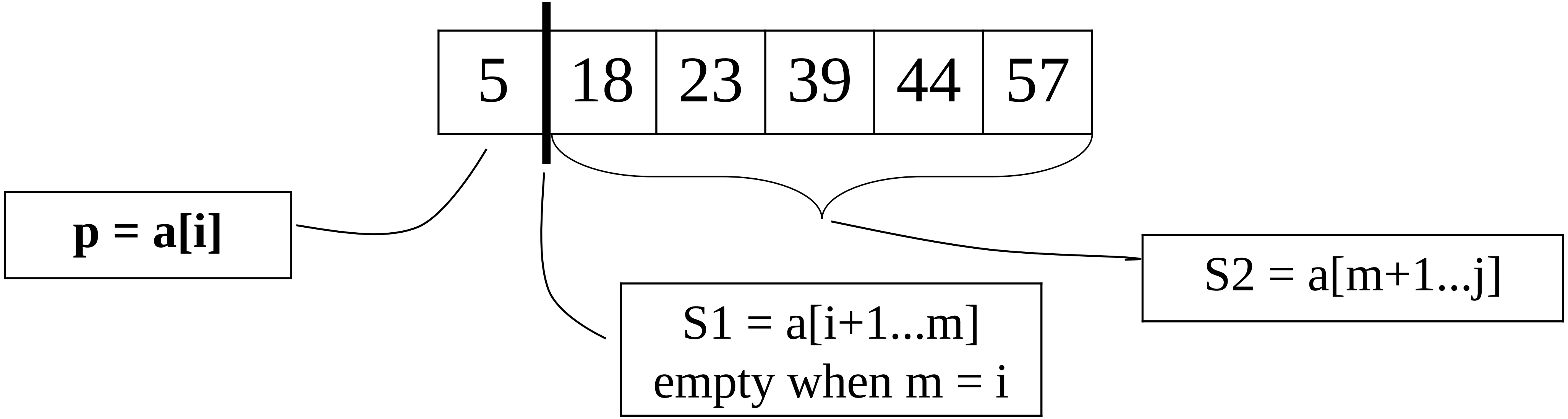
Quick Sort
Partition Analysis

- There is only a single for-loop:
  - Number of iterations = number of items, n, in the unknown region.
  - Complexity is O(n).

- Similar to Merge Sort, the complexity is then dependent on the number of times partition() is called.

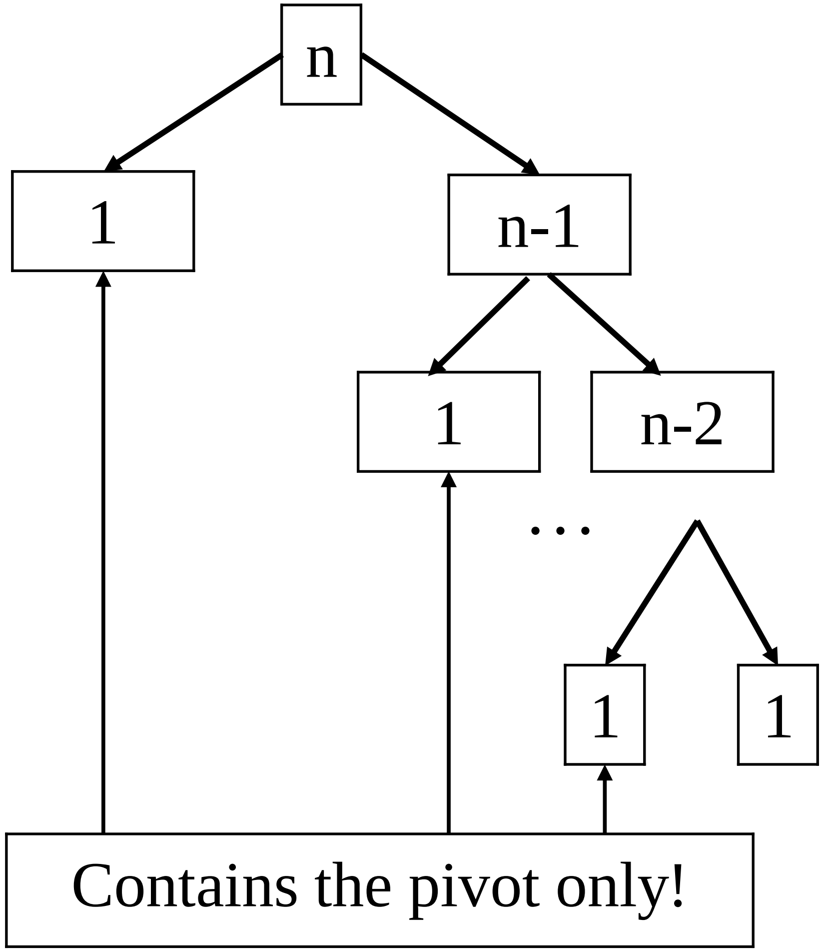
Quick Sort
Worst Case Analysis

- When the array is already in ascending order



- S1 is empty, while S2 contains every item except the pivot

Quick Sort Analysis



Total no. of levels = n

As each partition takes linear time, the algorithm in its worst case has n levels and hence it takes time n+(n-1)+…+ = $O(n^2)$

Quick Sort
Best/Average Case Analysis

- Best case occurs when partition always splits the array into two equal halves:
  - Depth of recursion is $\log(n)$.
  - Each level takes n or fewer comparisons, so complexity is $O(n \cdot \log(n))$.

- In practice, worst case is rare, and on the average we get some good splits and some bad ones:
  - Average time is also $O(n \cdot \log(n))$.