Quick sort,
Properties of sorting algorithms

The Quicksort Algorithm

- Most sorting libraries use some variation of the *Quicksort algorithm*, which was developed by Tony Hoare in the 1950s.
- The Quicksort algorithm consists of two phases:

1. *Partition*. In the partition phase, the elements of the array are reordered so that the array begins with a set of "small" elements and ends with a set of "big" elements, where the distinction between "small" and "big" is made relative to an element called the *pivot*, which appears at the boundary between the two regions.
2. *Sort*. In the sort phase, the Quicksort algorithm is applied recursively to the "small" and "big" subarrays, which leaves the entire array sorted.

Partitioning the Array

touching lives

0	1	2	3	4	5	6	7	8	9
209	503	946	367	987	838	209	838	987	946

1. Select the first element as the *pivot* and set it aside.
2. Keep two indices into the remaining elements, starting at each end.
3. Advance the indices until the left index is larger than the pivot and the right index is smaller.
4. Exchange the elements at the two indices.
5. Repeat the process until the indices coincide.
6. Swap the pivot and the index.

The QuickSort Implementation


```
void quickSort(int arr[], int left, int right) {
      int i = left, j = right, tmp;
      int pivot = arr[(left + right) / 2];

      /* partition */
      while (i <= j) {
            while (arr[i] < pivot) i++;
            while (arr[j] > pivot) j--;
            if (i <= j) {
                  tmp = arr[i]; arr[i] = arr[j]; arr[j] = tmp;
                  i++; j--;
            }
      };

      /* recursion */
      if (left < j)
            quickSort(arr, left, j);
      if (i < right)
            quickSort(arr, i, right);
}
```

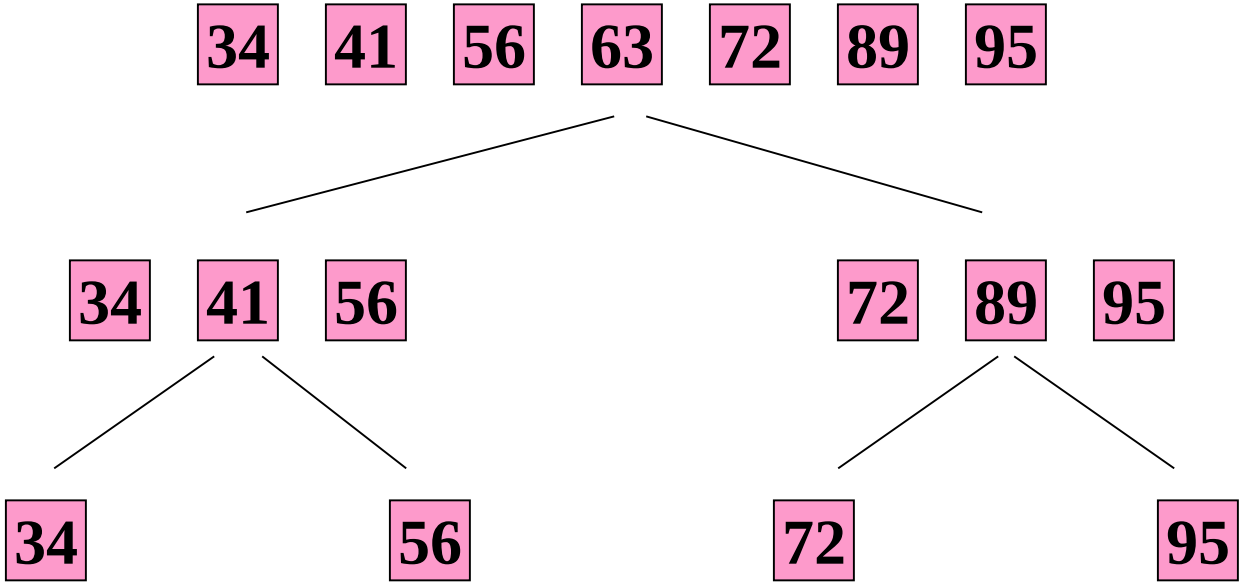
Complexity Analysis

- The number of comparisons in the top-level call is *N.*
- The sum of the comparisons in the two recursive calls at the next level is also *N.*
- The sum of the comparisons in the four recursive calls beneath these is also *N*, etc.
- Thus, the total number of comparisons equals *N* * the number of times the list must be subdivided.

How Many Times Will the Array Be Subdivided?

- It depends on the data and on the choice of the pivot element.
- Ideally, when the pivot is the median on each call, the list is subdivided lg N times.
- Best-case behavior is O(*N* log *N*)

Call Tree For a Best Case

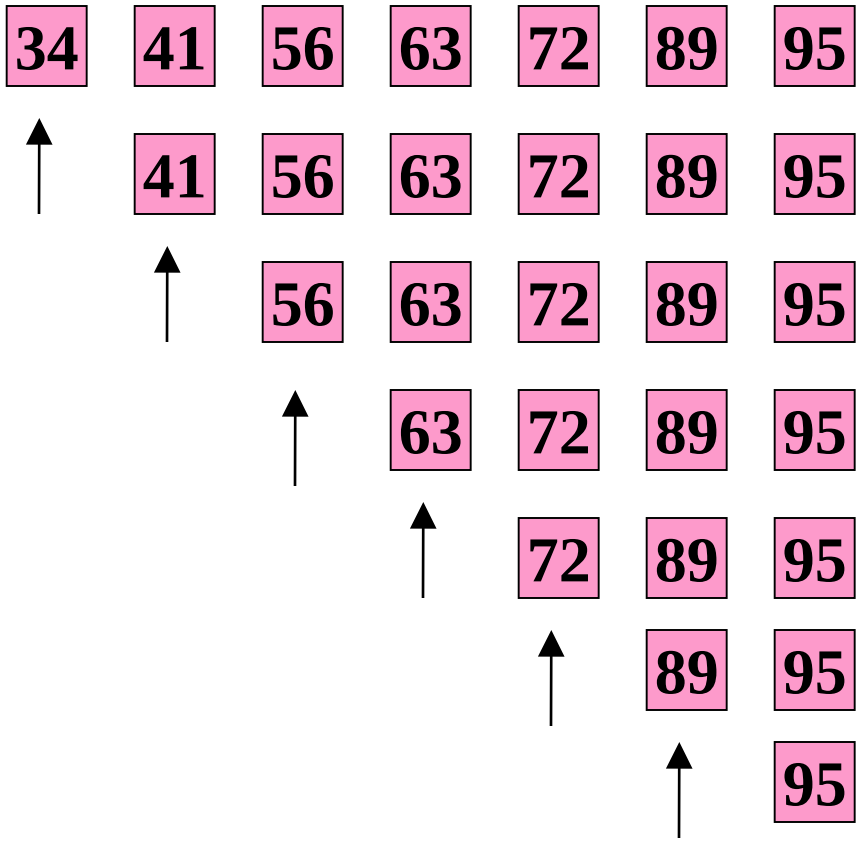


We select the midpoint element as the pivot.
The median element happens to be at the midpoint on each call.
But the list was already sorted!

Worst Case

- What if the value at the midpoint is near the largest value on each call?
- Or near the smallest value on each call?
- Then there will be approximately *n* subdivisions, and quick sort will degenerate to O(n^2) running time

Call Tree For a Worst Case



We select the first element as the pivot.
The smallest element happens to be the first one on each call.
n subdivisions!

Other Methods of Selecting the Pivot Element

- Pick a random element
- Pick the median of the first three elements
- Pick the median of the first, middle, and last elements
- Pick the median element of all n elements - *No!!* It requires an O(*n*) algorithm just to find the median

Average Case

- Assume that data is initially arranged in random order (every permutation is equally likely to occur, with probability 1/n!)
- Quicksort will take O(*n lg n*) average-case time, using any of the previous strategies for choosing the pivot
- Analysis of average-case is complicated

Mergesort compared to Quicksort

- Mergesort:
  - Just splits (no order)
  - Always splits exactly in half
  - Merge after the recursion
- Quicksort:
  - Partitions based on a pivot value
  - Split can be uneven
  - Finished after the recursion

Properties of Sorts

- In-place vs. Out of place
- Stability
- Comparison based

In Place vs Out of Place

- How much additional space (memory usage) is needed for the sort?
  - O(1) or O(lg N) space is called In-place
  - O(N) is called Out-of-place
  - More than O(N) is called wasteful.

Which of these sorting algorithms are in-place?

Bubble sort? Selection sort? Insertion sort?

Merge sort? Quick sort?

Stability

- A sorting algorithm is stable if whenever there are two records R and S with the same key, and R appears before S in the original list, then R will always appear before S in the sorted list.
- In other words:
  Stable sorts maintain the order of equal keys.

Which of these sorting algorithms are stable?

Bubble sort? Selection sort? Insertion sort?

Merge sort? Quick sort?

Comparison Based Sorting

- Every sorting algorithm we've seen so far uses comparisons to sort, i.e. if (a < b) :
  - It's not possible to sort "faster" than O(N lg N) time using comparisons
- There exist sorting algorithms not based on comparisons.
  - All of these algorithms only work for restricted types and ranges of data