

**NPTEL MOOC**

# **PROGRAMMING, DATA STRUCTURES AND ALGORITHMS IN PYTHON**

**Week 4, Lecture 4**

**Madhavan Mukund, Chennai Mathematical Institute**

**<http://www.cmi.ac.in/~madhavan>**



# Quicksort

- \* Choose a pivot element
  - \* Typically the first value in the array
- \* Partition **A** into lower and upper parts with respect to pivot
- \* Move pivot between lower and upper partition
- \* Recursively sort the two partitions



# Quicksort in Python

```
def Quicksort(A,l,r): # Sort A[l:r]
    if r - l <= 1:    # Base case
        return ()

    # Partition with respect to pivot, a[l]
    yellow = l+1

    for green in range(l+1,r):
        if A[green] <= A[l]:
            (A[yellow],A[green]) = (A[green],A[yellow])
            yellow = yellow + 1

    # Move pivot into place
    (A[l],A[yellow-1]) = (A[yellow-1],A[l])

    Quicksort(A,l,yellow-1) # Recursive calls
    Quicksort(A,yellow,r)
```



# Analysis of Quicksort

## Worst case

- \* Pivot is either maximum or minimum
  - \* One partition is empty
  - \* Other has size  $n-1$
  - \* 
$$T(n) = T(n-1) + n = T(n-2) + (n-1) + n$$
$$= \dots = 1 + 2 + \dots + n = O(n^2)$$
- \* Already sorted array is worst case input!



# Analysis of Quicksort

But ...

- \* Average case is  $O(n \log n)$ 
  - \* All permutations of  $n$  values, each equally likely
  - \* Average running time across all permutations
- \* Sorting is a rare example where average case can be computed



# Quicksort: randomization

- \* Worst case arises because of fixed choice of pivot
  - \* We chose the first element
  - \* For any fixed strategy (last element, midpoint), can work backwards to construct  $O(n^2)$  worst case
- \* Instead, choose pivot **randomly**
  - \* Pick any index in  $\text{range}(0, n)$  with uniform probability
- \* **Expected running time** is again  $O(n \log n)$



# Quicksort in practice

- \* In practice, Quicksort is very fast
  - \* Typically the default algorithm for in-built sort functions
    - \* Spreadsheets
    - \* Built in sort function in programming languages



# Stable sorting

- \* Sorting on multiple criteria
- \* Assume students are listed in alphabetical order
- \* Now sort students by marks
  - \* After sorting, are students with equal marks still in alphabetical order?
- \* Stability is crucial in applications like spreadsheets
  - \* Sorting column B should not disturb previous sort on column A



# Stable sorting ...

- \* Quicksort, as described, is not stable
  - \* Swap operation during partitioning disturbs original order
- \* Merge sort is stable if we merge carefully
  - \* Do not allow elements from right to overtake elements from left
  - \* Favour left list when breaking ties