Divide and Conquer

Data and File Structures Laboratory

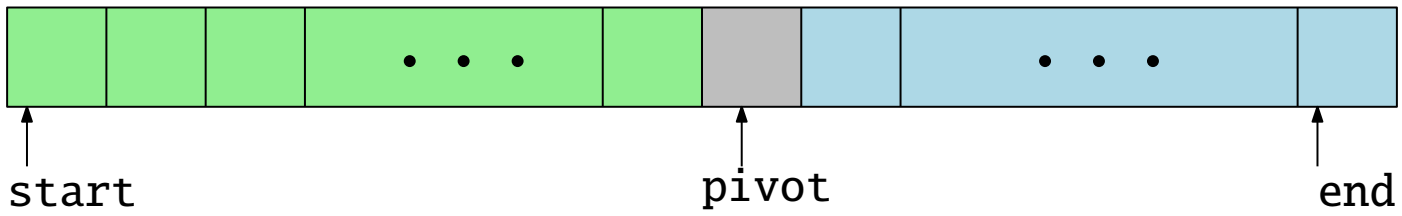
http://www.isical.ac.in/~dfslab/2020/index.html

Standard algorithms

- Binary search
- Mergesort
- Insertion sort

Quicksort

```
def quicksort(A, start, end) :
    if end > start : # sort only if more than 1 element
        pivot = partition(A, start, end) # O(n)
        quicksort(A, start, pivot-1)
        quicksort(A, pivot+1, end)
```

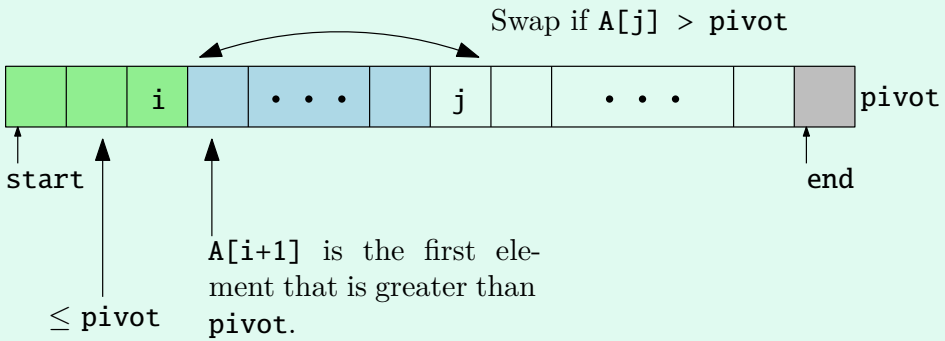


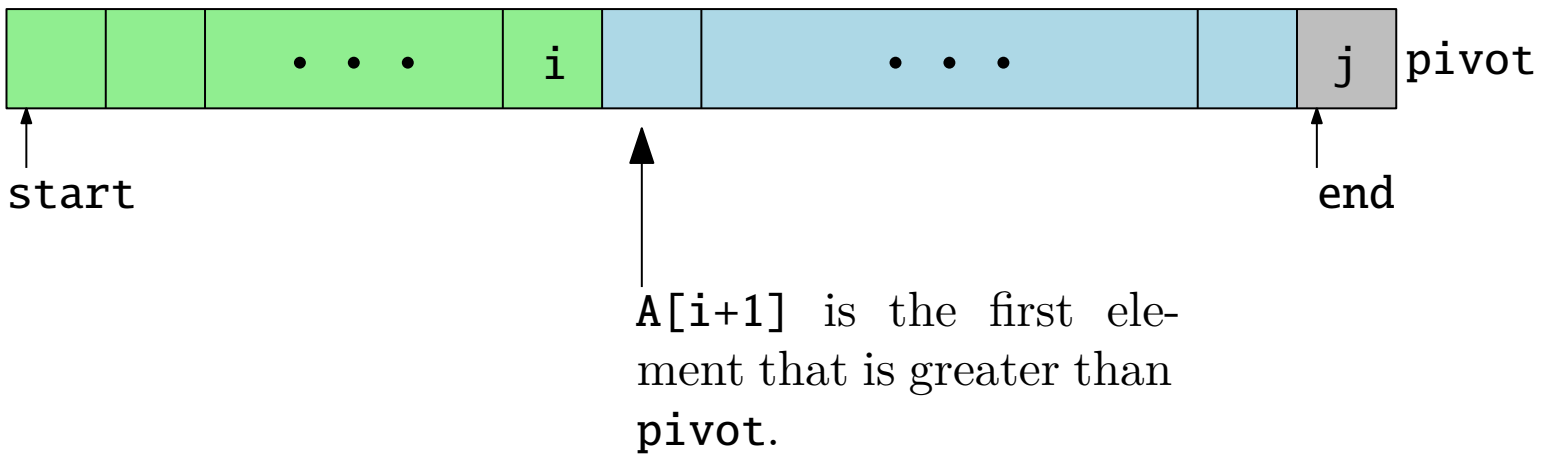
`partition()`

- picks a `pivot` (first / last element, or randomly)
- partitions list into 2 parts with `pivot` element in between
- *returns correct index* of `pivot` in sorted order

Quicksort

```
def partition(A, start, end) :
    pivot = A[end]
    i = start-1
    for j in range(start, end) :
        if A[j] <= pivot :
            i = i+1
            A[i], A[j] = A[j], A[i]
    A[i+1], A[end] = A[end], A[i+1]
    return i+1
```



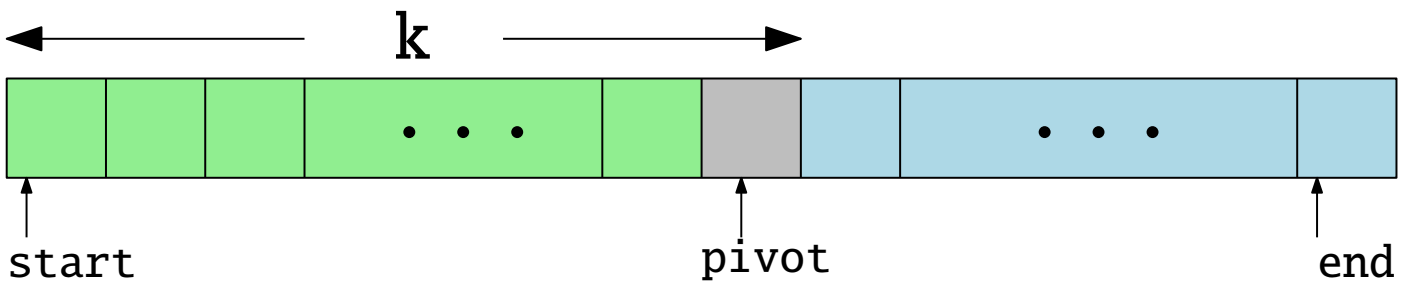


Randomised median finding

Reference: CLR 9.2,9.3

```
def randomised_select(A, start, end, i) :
    if end - start + 1 < i : # not enough elements
        return A[end]

    pivot = partition(A, start, end)
    k = pivot - start + 1
    if k == i : return A[pivot]
    if k < i : return randomised_select(A, pivot+1, end, i-k)
    else : return randomised_select(A, start, pivot-1, i)
```



Python implementation of deterministic median finding algorithm: https://stackoverflow.com/questions/24101524/finding-median-of-list-in-python

Problems – I

1. **Longest ascending subsequence problem.** A *subsequence* of a list L of length n is a list of the form $L[i_1], L[i_2], \ldots, L[i_m]$, where $0 \leq i_1 < i_2 < \cdots < i_m < n$. Write a program that takes a list L of distinct natural numbers as input, and finds the length of the longest subsequence of L whose elements are in increasing order.

   Your algorithm should run in quadratic time.

2. **Round-robin tournament design.** Suppose there are $n = 2^k$ teams in a round-robin tournament. Every team must play every other team exactly once, and each team can play at most one match per day. Write a program to compute a schedule that will enable the tournament to be completed in the minimum time.

3. Write a program to compute a^n for a given *base* a and an exponent $n \in \mathbb{N}$ using the minimum number of multiplications.