Lecture 22, 3 November 2026

Madhavan Mukund

https://www.cmi.ac.in/~madhavan

Programming and Data Structures with Python

Lecture 22, 3 Nov 2026

Quicksort, partitioning

Choose a pivot/splitter — e.g. `l[0]`

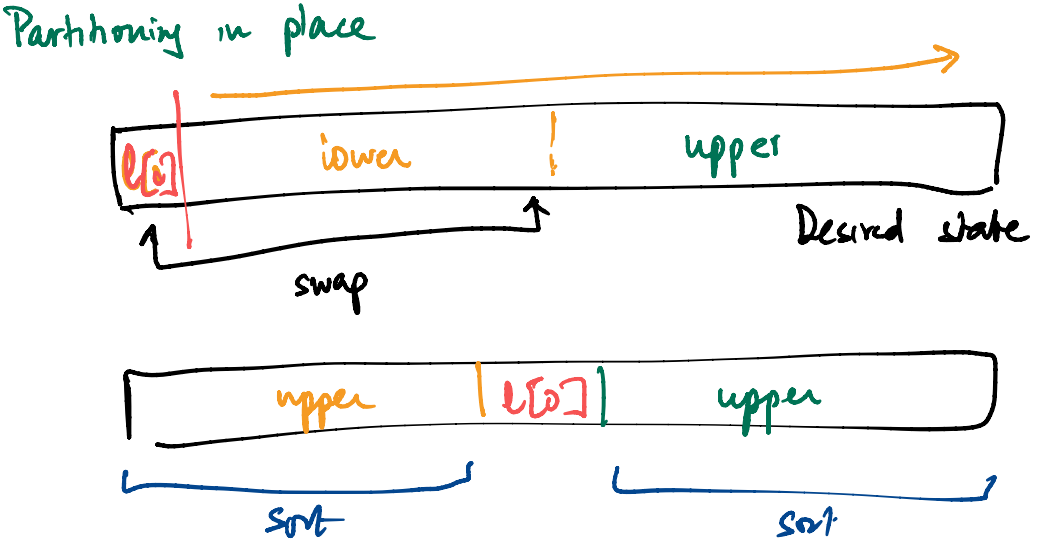
```
lower = [x | for x in l[1:] if x <= l[0]]
upper = [x | for x in l[1:] if x > l[0]]
```

```
sort(lower) + [l[0]] + sort(upper)
```

No merge required

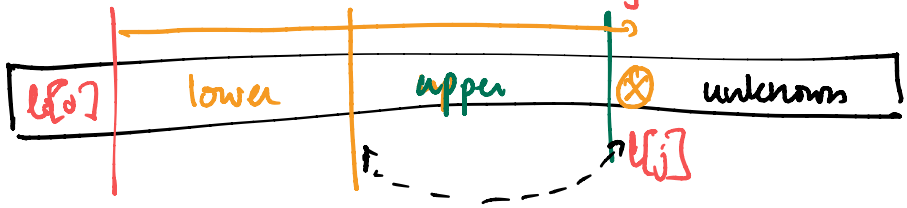
- Extra space ✓
- Recursion

Quicksort, partitioning



Quicksort, partitioning

Partition — Some Polish name starting with B



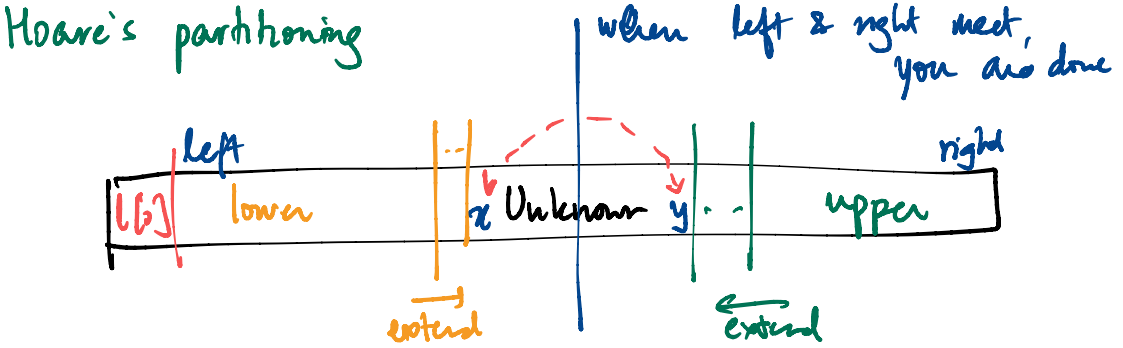
$l[j] > l[0]$ — extend upper to right

$\leq$ — swap leftmost upper with $l[j]$

effective extends lower

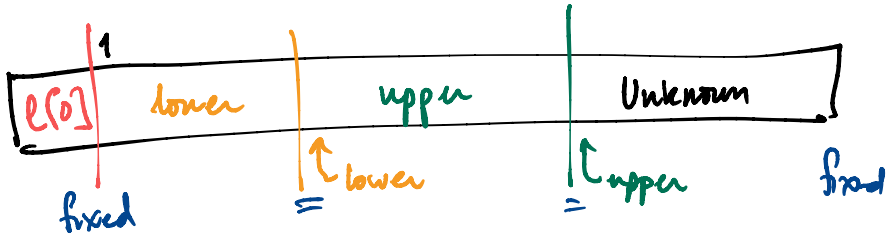
effective shifts upper by 1 place

Quicksort, partitioning



Quicksort, partitioning

Programming the partition

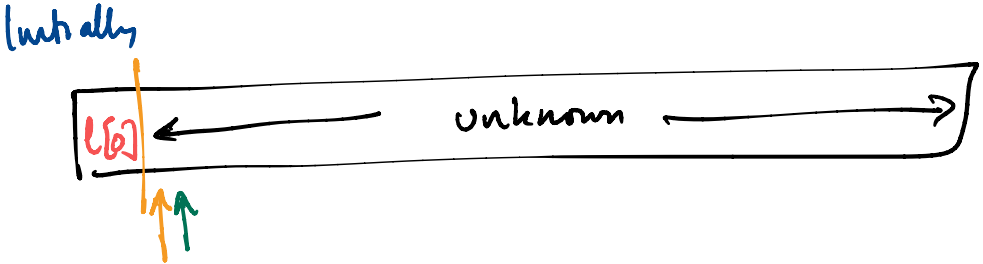


lower segment = l[1 : lower]

upper segment = l[lower : upper]

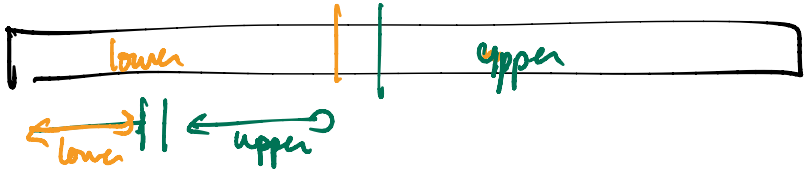
unknown = l[upper :]

Quicksort, partitioning



Quicksort, partitioning

In general, we are sorting an arbitrary slice of l

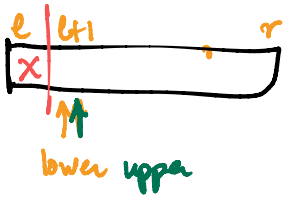


Sort l[left : right] - splitter is l[left]

Quicksort code

```
def quicksort(L,l,r):  # Sort L[l:r]
  if (r - l <= 1):
    return(L)
  (pivot,lower,upper) = (L[l],l+1,l+1)
  for i in range(l+1,r):
    if L[i] > pivot:  # Extend upper segment
      upper = upper+1
    else: # Exchange L[i] with start of upper segment
      (L[i], L[lower]) = (L[lower], L[i])
      # Shift both segments
      (lower,upper) = (lower+1,upper+1)
  # Move pivot between lower and upper
  (L[l],L[lower-1]) = (L[lower-1],L[l])
  lower = lower-1
  # Recursive calls
  quicksort(L,l,lower)
  quicksort(L,lower+1,upper)
  return(L)
```

] length of slice ≤ 1



] Challenge → remove recursion

Quicksort analysis

Cannot assume anything about l[0]

Worst case, l[0] is min or max

lower = []

upper = l[1:]

$T(0) = T(1) = 1$

$T(n) = T(\text{lower}) + T(\text{upper}) + n$

$T(\text{lower})$: 0 — $T(\text{upper})$: $n-1$

Quicksort analysis

Worst case recurrence

$$T(0) = T(1) = 1$$

$$T(n) = T(n-1) + n$$

$$= T(n-2) + (n-1) + n$$

$$= 1 + 2 \ldots + n = O(n^2)$$

Quicksort analysis

Worst case?

List already sorted

How common is worst case?

"Average" case?

Quicksort analysis

Sorting a list of n elements

- Can assume a permutation of $1, 2, \ldots, n$
- Assume uniform distribution

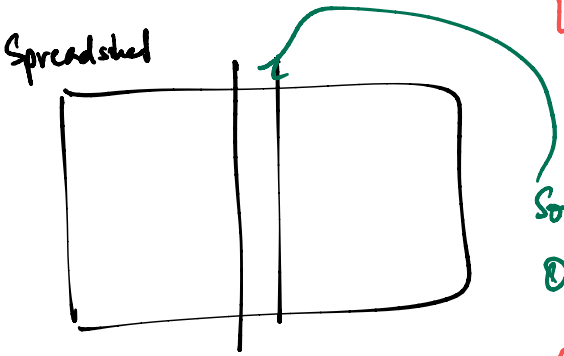
Expected running time is $O(n \log_2 n)$

Suppose we choose $l[mid]$ as pivot? $mid = n//2$

Reverse engineer a bad input

Quicksort analysis

In practice, quicksort is very fast



- In place, iterative
- Sort by column, ascending
- Oops, I meant descending
- Check already sorted

Randomized quicksort

Choose a pivot position in l to r-1 at random

Start by randomly choosing pivot.

Swap L[0], L[pivot] and proceed as before

Again expected running time is $O(n \log_2 n)$

Sorting in practice

Unix – used quicksort as default sort

qsort( ) in C

Hybrid strategy

Python – Tim Sort – Merge sort upto a point

Then insertion sort

– works well when sorted

look for "sorted runs"

Sorting in practice

No of swaps ?

Algorithms to minimize no/distance of swaps

Sorting in practice

Have the list of enrolled students, sorted by roll number

Assign marks

Sort by marks to assign grades

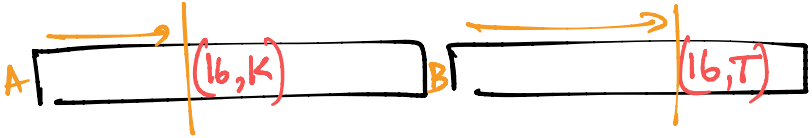
Students with same marks

Do they remain in roll number order?] Stability

Sorting in practice

Stable sorting

Merge



A[i] <= B[j] → move A[i] to C — STABLE

(16,K) (16,T)

A[i] < B[j] → move A[i] to C — UNSTABLE

Sorting in practice

Quicksort?

Swaps all over the place

Hard (if not impossible) to make it stable