Lecture 21, 29 October 2026

Madhavan Mukund

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

Programming and Data Structures with Python

Lecture 21, 29 Oct 2026

Merge sort

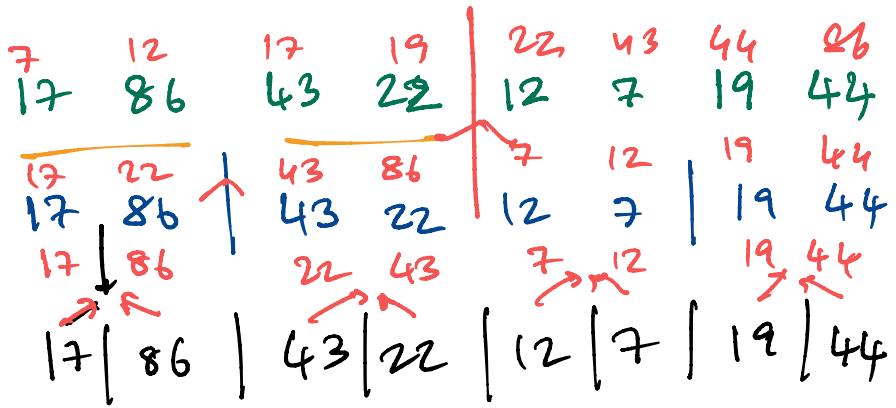
Merge 2 sorted list of size n in $O(n)$ time

To sort a list

Recursively sort two halves

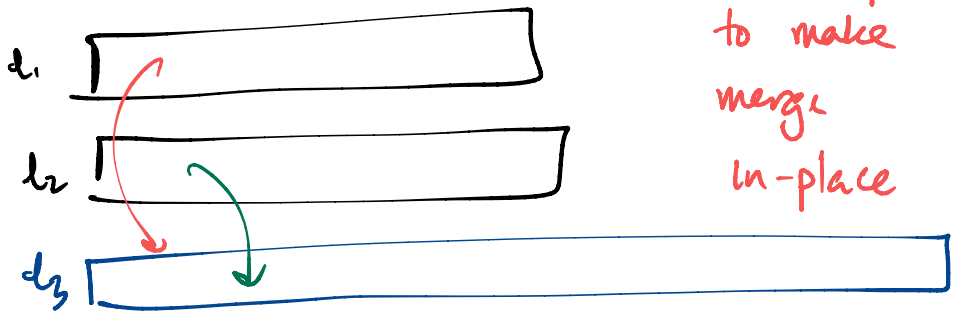
Merge them

Merge sort


17 12 17 19 22 43 44 86
17 86 43 22 12 7 19 44
17 22 43 86 7 12 19 44
17 86 43 22 12 7 19 44
17 86 22 43 7 12 19 44
17 86 43 22 12 7 19 44

Merge sort

Merge l_1 & l_2



Not easy to make merge in-place

Merge sort

Not easy to avoid recursive calls

Merge sort

```
def merge(A,B):
  (m,n) = (len(A),len(B))
  (C,i,j,k) = ([],0,0,0)
  while k < m+n:
    if i == m:
      C.extend(B[j:])
      k = k + (n-j)
    elif j == n:
      C.extend(A[i:])
      k = k + (m-i)
    elif A[i] < B[j]:
      C.append(A[i])
      (i,k) = (i+1,k+1)
    else:
      C.append(B[j])
      (j,k) = (j+1,k+1)
  return(C)
```

```
def mergesort(A):
  n = len(A)

  if n <= 1:
     return(A)

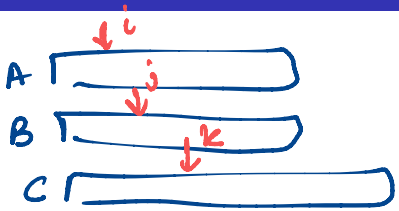
  L = mergesort(A[:n//2])
  R = mergesort(A[n//2:])

  B = merge(L,R)

  return(B)
```

Recursive

New list!



Merge sort analysis

$TM(n)$ = Merging n elements $\quad |A| = |B| = n$, $= 2n$

$TS(n)$ = Sorting n elements

$$TS(0) = TS(1) = 1$$

$$TS(n) = 2 \cdot TS(n/2) + TM(n/2)\ n$$

Merge sort analysis

$$TS(n) = 2 \cdot TS(n/2) + n$$

$$2 \cdot [2 \cdot TS((n/2)/2) + n/2] + n$$

$$2^2 \cdot TS(n/2^2) + 2n$$

$$2^2 [2 \cdot TS(n/2^3) + n/2^2] + 2n$$

$$2^3 \cdot TS(n/2^3) + 3n$$

Merge sort analysis

After expanding k times

$$TS(n) = 2^k TS\left(n/2^k\right) + k \cdot n$$

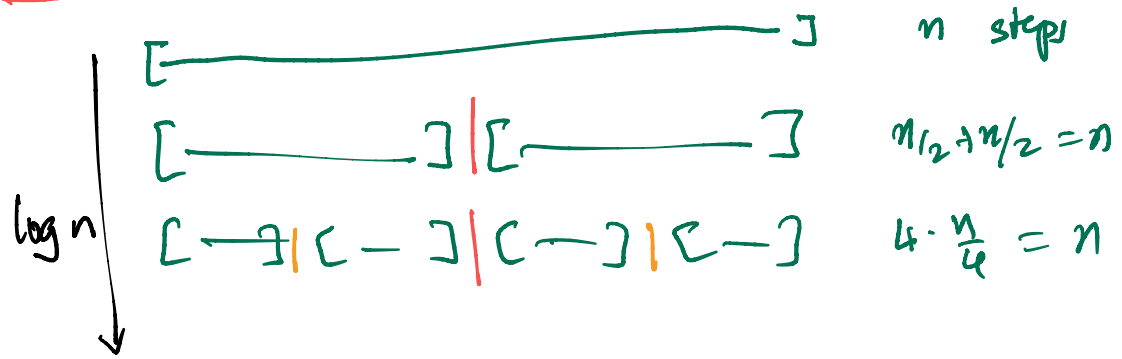
$$TS(n) = n \cdot TS(1) + (\log_2 n) \cdot n \qquad \frac{n}{2^k} \Rightarrow 1$$

$$= n + n \log_2 n \qquad k = \log_2 n$$

$$= O(n \log_2 n)$$

Merge sort analysis

Another way to see this



Divide and conquer without merging

1 3 5 7 2 4 6 8

What if everything on left was smaller than everything on the right How?

Answer = sort(left) + sort(right)

Divide and conquer without merging

Suppose we knew the median value

Left = $\leq$ median

Right = $>$ median

⇓ One pass

$O(n)$

```
left = [ x | for x in l if x <= median]
right = [ x | for x in l if x > median]
```

Divide and conquer without merging

How do we know the median?

Sort & check middle value

Compromise

Pick some value and use it to split

Typically choose $l[0]$ to split

Split $l[1:]$ wrt $l[0]$

Divide and conquer without merging

left (or lower) = [x | for x in l[1:] if x <= l[0]]

right (or upper) = [x | for x in l[1:] if x > l[0]]

Sort (lower)
Sort (upper)

Splitter ≡ Pivot

Sort (lower) + [l[0]] +
Sort (upper)

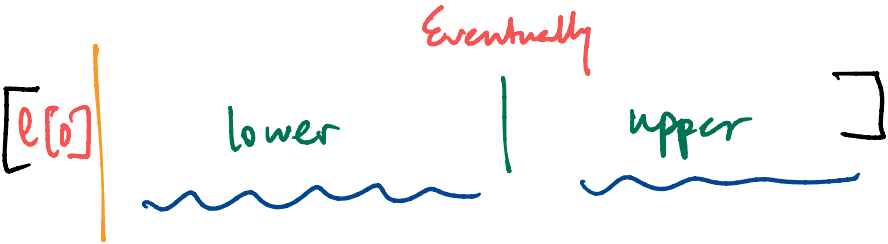
Divide and conquer without merging

If pivot/splitter is approx median every time

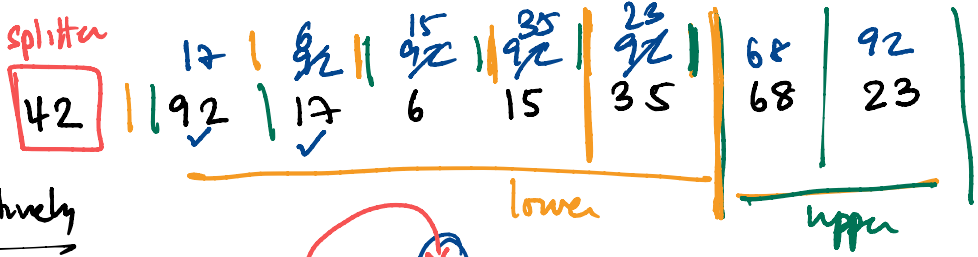
$$O(n \cdot \log n)$$

Space?

Quicksort = C.A.R. Hoare



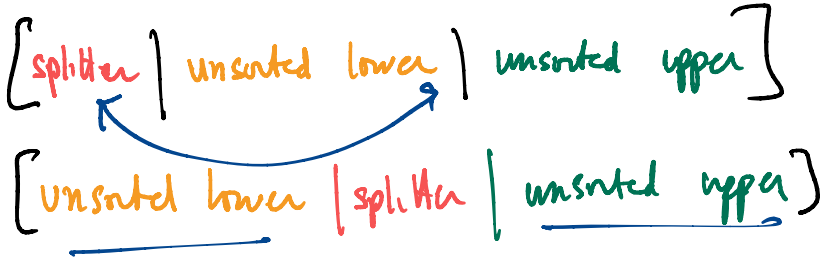
Quicksort


splitter
42
17
92
92
17
15
6
85
15
23
35
68
68
92
23
lower
upper
Inductively
lower
upper
x
unexplored
empty eventually
x ≤ splitter – swap with first in upper
x > splitter – expand upper

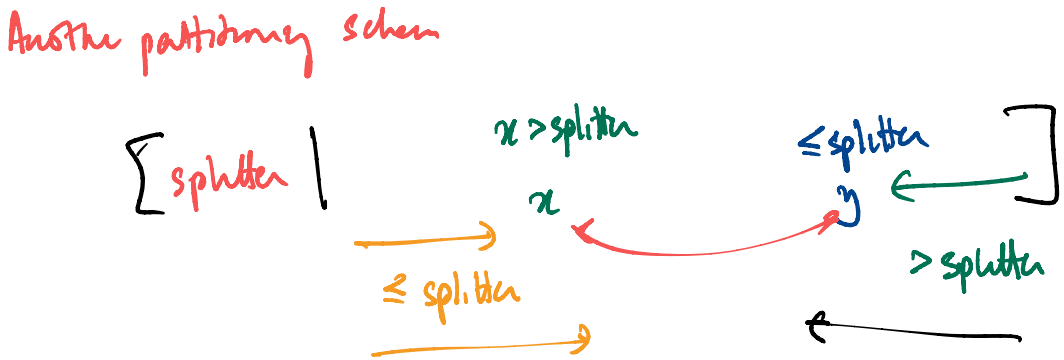
Quicksort

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

Before recursion



Quicksort


Another partitioning scheme
splitter
x > splitter
x
≤ splitter
≤ splitter
y
> splitter