Lecture 20, 29 October 2024

Madhavan Mukund

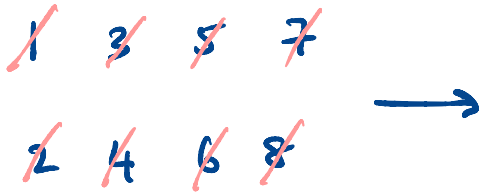
https://www.cmi.ac.in/~madhavan

Programming and Data Structures with Python

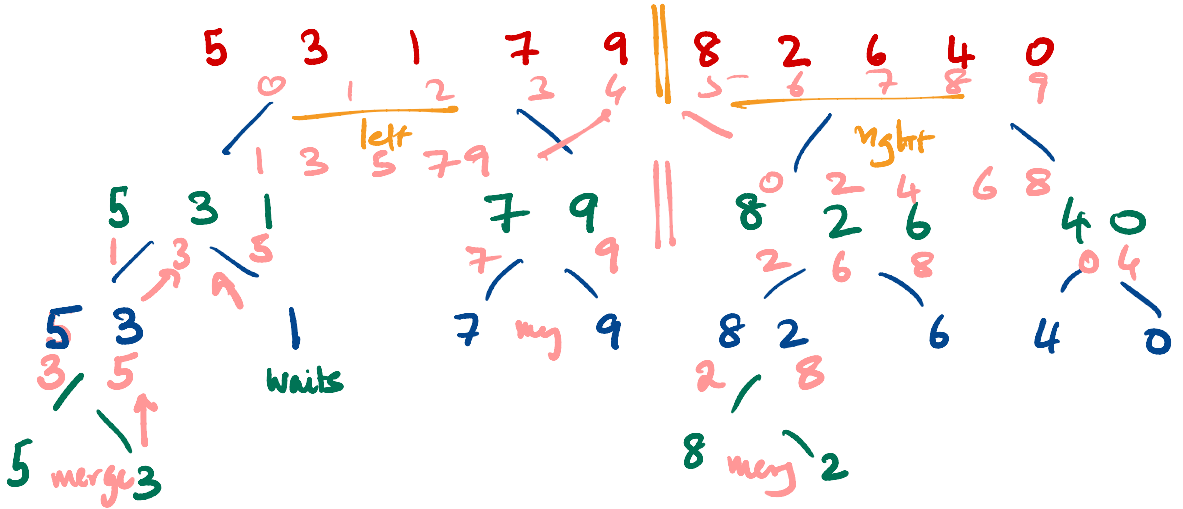
Lecture 20, 29 Oct 2024

Merge sort

$O(n \log n)$ sort?



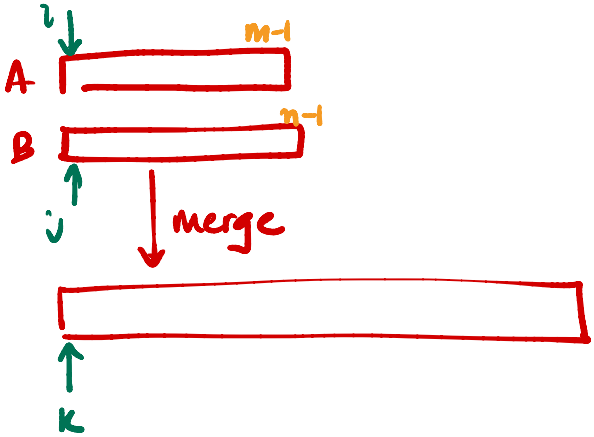
Merge sort



Merging sorted lists

```
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)
```

Boundary cases



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)
```

Base case

no gap!

Merge sort analysis

Let $T(n)$ be time to merge sort input of size n

$$T(n) = 2T(n/2) + n$$

└ Merge

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

Merge sort analysis

$$T(n) = 2T(n/2) + n$$

$$= 2\left[2T(n/4) + n/2\right] + n \times 1$$

Level 3 $$= 2\left[2\left[2T(n/8) + n/4\right] + n/2\right] + n$$

2^3 2^2 $\times 2^2$ n $\times 2 = n$

$$2^3 T\left(n/2^3\right) + 3n$$

Merge sort analysis

After k steps

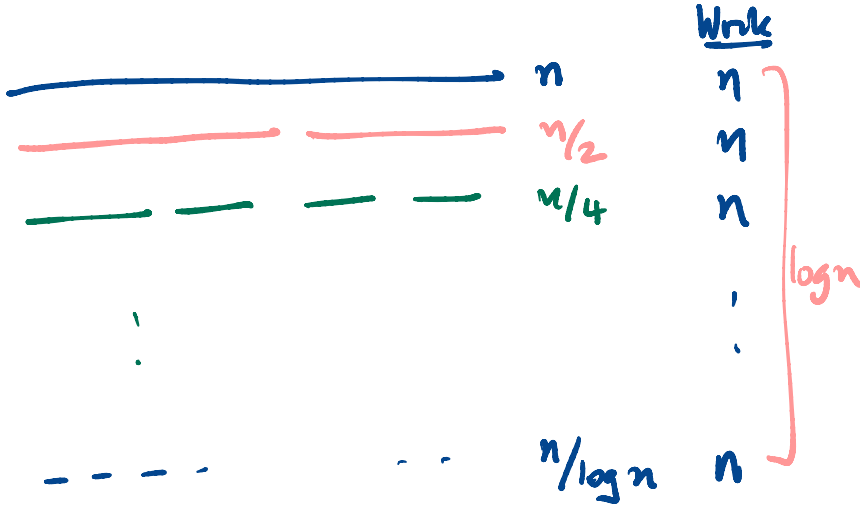
$$2^k T\left(\frac{n}{2^k}\right) + kn \qquad k = \log_2 n$$

$$2^{\log n} T(1) + (\log n)\, n \qquad \frac{n}{2^k} = 1$$

$$n \cdot 1 + n \cdot \log n \longrightarrow O(n \log n)$$

Merge sort analysis

Pictorally



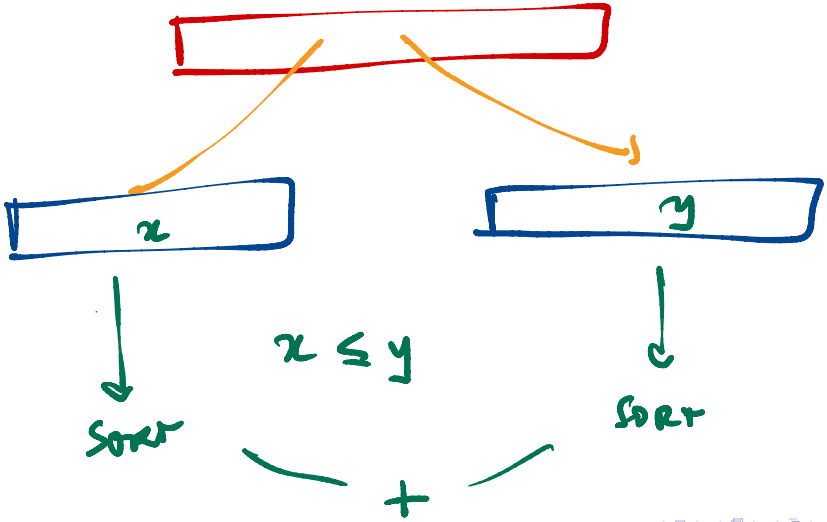
Merge sort analysis

Divide & Conquer

Break up into disjoint subproblems

Combine subpart solutions

Divide and conquer without merging



Quicksort

C.A.R. Hoare "Tony"

List → split into Lower & Upper

How?

Pick some value in the list - PIVOT

l 0..n-1

```
pivot = l[0]
lower = [x for x in l[1:] if x < pivot]
upper = [y for y in l[1:] if y ≥ pivot]
```

Quicksort

```
def quicksort(l):
    if len(l)<=1:
        return(l)
    pivot = l[0]
    lower = [x for x in l[1:] if x < pivot]
    upper = [y for y in l[1:] if y ≥ pivot]
    return( quicksort(lower) + [pivot] + quicksort(upper))
```

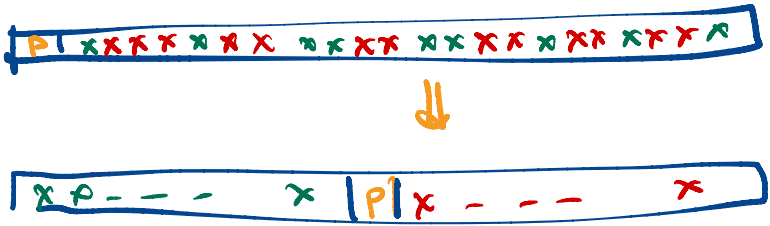
Quicksort

Extra space

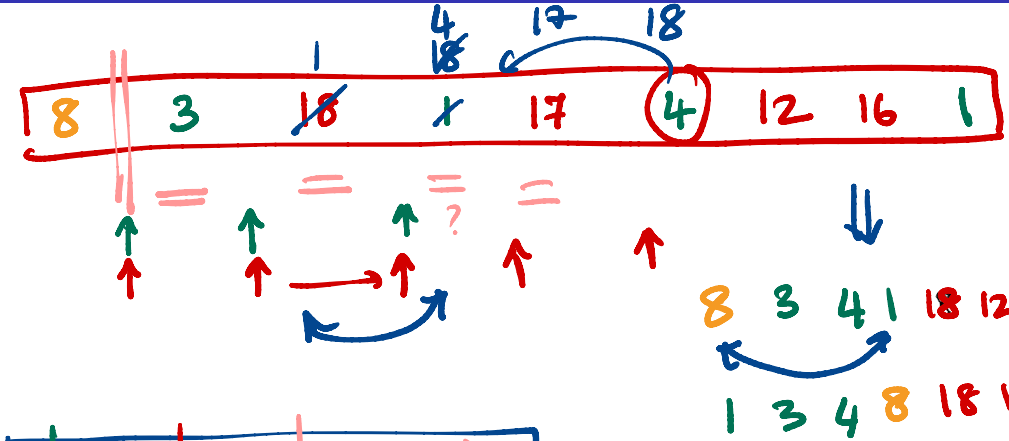
Merge $A, B \longrightarrow C$

Quicksort $\longrightarrow$ lower, upper

Partitioning



Partitioning



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)
```

l..r

L[l] is pivot

Partitioning

Quicksort analysis

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

	?	?
If pivot is smallest	0	$n-1$
or largest	$n-1$	0

$T(n) = T(n-1) + n$ Same as insertion sort

$= 1 + \cdots + n-1 + n = O(n^2)$

Average case

Sorting: Only order matters

Input of size n — permutation of $[1..n]$

Expected time of quicksort is $O(n \log n)$

Randomized quicksort

For any fixed strategy to choose pivot

- Position largest/smallest value there
- Produce worst case n^2 input

Pick $i \in [0 .. n-1]$ uniformly at random

Use $L[i]$ as pivot $\rightarrow$ Swap $L[i]$ & $L[0]$, proceed as before

Stable sorting

First sort by roll number

then sort by marks

Equal marks → sorted by roll number

Stable sorting

Selection sort

2 2 1

1 2 2

Merge

$A[i] \leq B[j]$ is stable

$A[i] < B[j]$ is not

Stable sorting

Theoretical lower bound of $O(n \log n)$

Assuming only compare & swap