Data Structures & Algorithms

Subodh Kumar

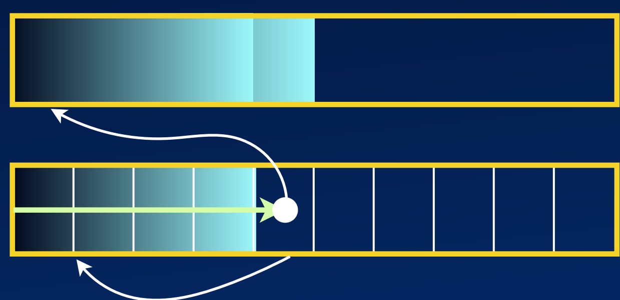
(subodh@iitd.ac.in, Bharti 422)

Dept of Computer Sc. & Engg.

Sorting

Insertion Sort

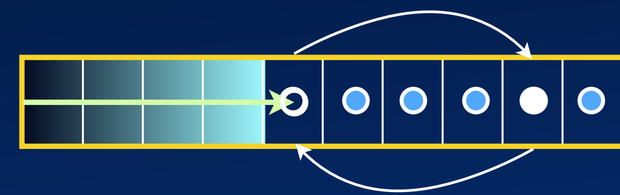
Stable



Insert next in the right place

Selection Sort

Unstable

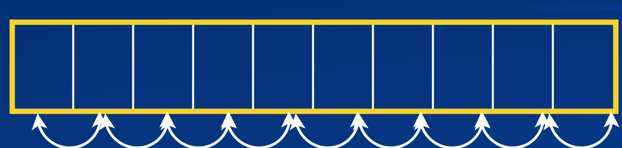


Which remaining element is next

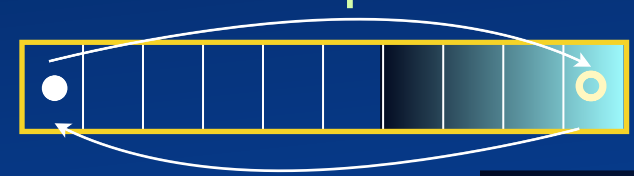
Can come from a Heap

Bubble Sort

Stable



Heap Sort



Unstable

T/F?

- The following sorting of (key, value) pairs is stable.
  - sorting is by the integer key

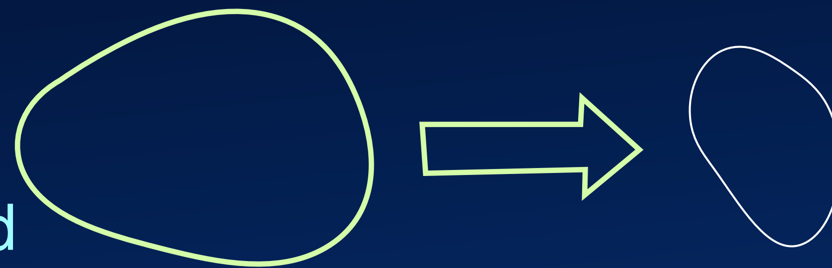
[(1, ‘king’) (11, ‘pawn’) (5, ‘knight') (1, ‘queen’) (9, ‘bishop’)]

→

[(1, 'queen') (1, 'king') (5, 'knight') (9, 'bishop') (11, ‘pawn')]

Email: col106quiz@cse.iitd.ac.in
Format: f

Search for Solution



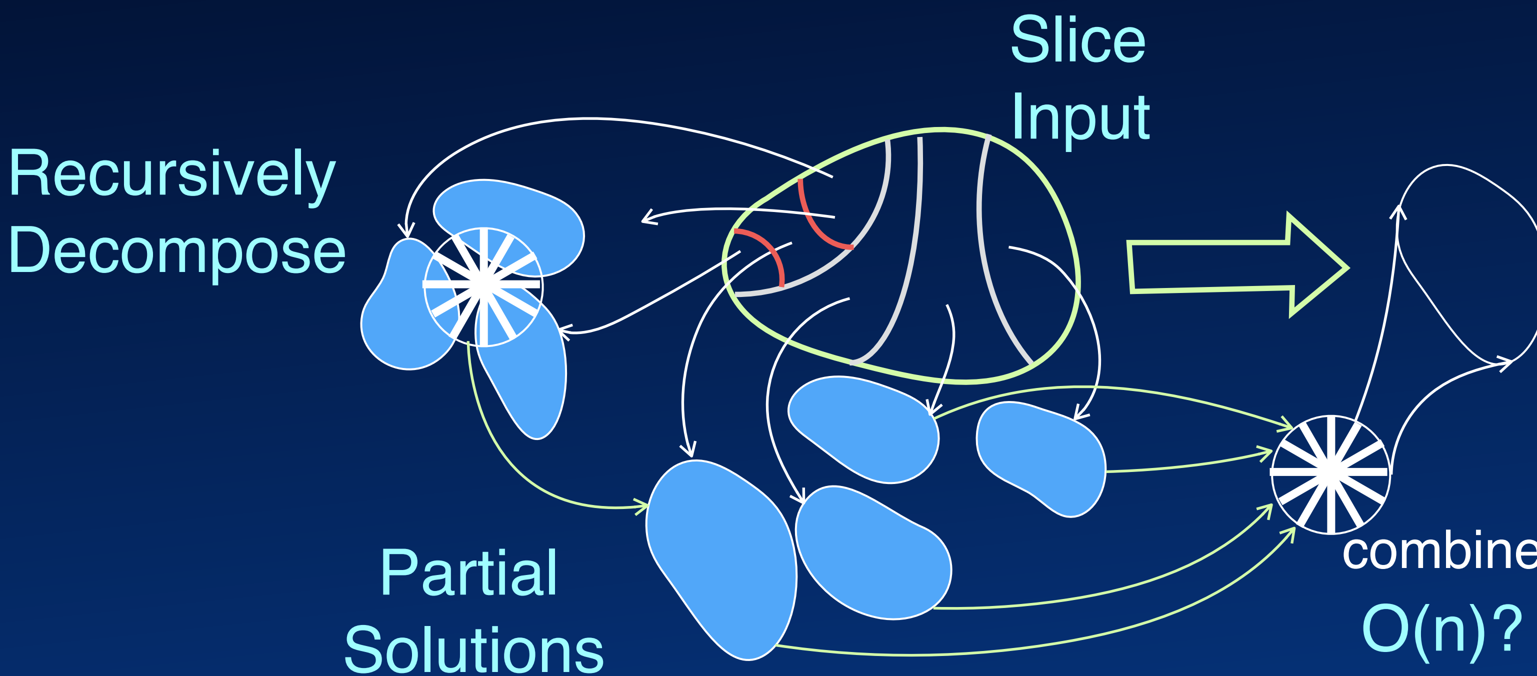
Systematically enumerated
Solution space

doit(input, output):
  Implicitly enumerate possible output
  successively eliminate impossible ones
  output = what remains

Eliminate
fraction of space

Recursively
Decompose

Divide and Conquer

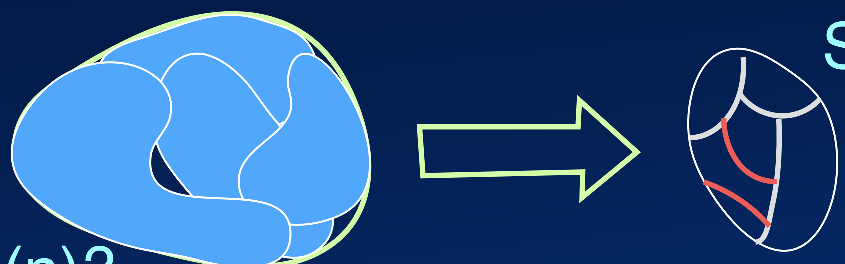


Roughly
zed

```
doit(input, output):
    list = Partition(input)
    for e in list partials.push(doit(e))
    output = combine(partials)
```

Divide and Conquer

Slices of Solution



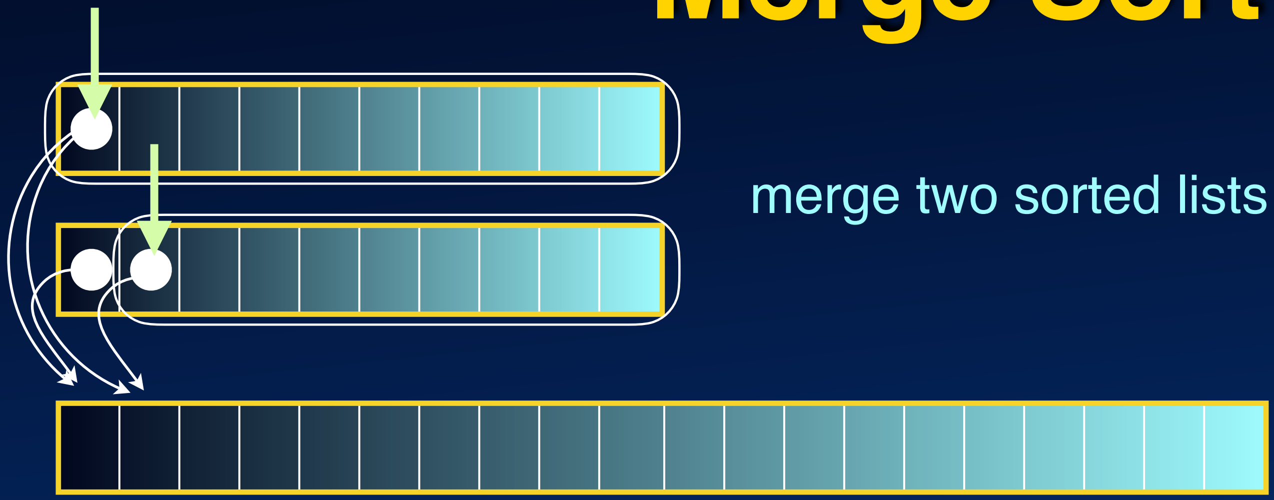
O(n)?

Roughly Equal Sized

Recursively Decompose

```
doit(input, output):
    list = Partition(output)
    foreach e in list, find its relevant input
        doit(relevant input, e)
```

Merge Sort



merge two sorted lists

```
while(i < size1 && j < size2):
    if(list1[i] before list2[j])
    result[next++] = list1[i++]
    else result[next++] = list2[j++]
```

Both i and j must reach their ends

```
while(i < size1):
    result[next++] = list1[i++];
while(j < size2):
    result[next++] = list1[j++];
```

O(size1+size2)

Merge Sort

- **Sort(Array[0..n/2]**
- **Sort(Array[n/2..n]**
- **Merge(Array[0..n/2], Array[n/2..n])**

$$
\begin{aligned}
T(n) &\le 2\,T(\lceil n/2 \rceil) + O(n) \\
&\le 2\,T(1+n/2) + O(n) \\
&\le 2\,[2T(1+n/4) + O(n/2)] + O(n) \\
&= 4\,T(1+n/4) + 2\,O(n/2) + O(n) \\
&= 4\,T(1+n/4) + O(n) + O(n) \\
&\le 8\,T(1+n/8) + 4O(n/4) + O(n) + O(n) \\
&\le 2^{i}\,T(1+n/2^{i}) + i\,O(n) \\
&\le 2^{\lg n}\,T(1+1) + \lg n\,O(n);\ T(2) = O(1) \\
&= O(n) + O(n \lg n) = O(n \lg n)
\end{aligned}
$$

Time Complexity of Merge Sort

- **Worst-case complexity of merge sort is:**
  a) $\Theta(n^2)$
  b) $\Theta(n \log n)$
  c) $\Theta(n)$
  d) $\Theta(\log n)$
- **Average complexity of merge sort is:**
  a) $\Theta(n^2)$
  b) $\Theta(n \log n)$
  c) $\Theta(n)$
  d) $\Theta(\log n)$

Email: col106quiz@cse.iitd.ac.in
Format: a,a

Merge Sort

- **Sort(Array[0..n/2]**
- **Sort(Array[n/2..n]**
- **Merge(Array[0..n/2], Array[n/2..n])**

Sort(A, 0, n-1)

```
Sort(A, s, e):
   if(s < e):
      Sort(A, s, (s+e)/2)
      Sort(A, 1+(s+e)/2, e)
      B = new Array of size e-s+1
      Copy(A[s:e]) to B[])
      Merge(B[first half], B[second half], A[s:e])
```

Merge Sort

- Sort(Array[0..n/2]
- Sort(Array[n/2..n]
- Merge(Array[0..n/2], Array[n/2..n])

Sort(A, B, 0, n-1)

```
Sort(A, temp, s, e):
    if(s < e)
        Sort(A, temp, s, (s+e)/2)
        Sort(A, temp, 1+(s+e)/2, e)
        Copy(A[s:e]) to temp[s:e])
        Merge(temp[s:(s+e)/2], temp[1+(s+e)/2:e], A[s:e])
```

Merge Sort

- **Sort(Array[0..n/2]**
- **Sort(Array[n/2..n]**
- **])**

```
SortintoB(A, B, s, e):
   if(s < e)
      SortintoA(A, B, s, (s+e)/2)
      SortintoA(A, B, 1+(s+e)/2, e)
      Merge(A[s:(s+e)/2], A[1+(s+e)/2:e], B)
  else B[s] = A[s]
```

```
Sort?(A, B, 0, n-1)
```

```
SortintoA(A, B, s, e):
   if(s < e)
      SortintoB(A, B, s, (s+e)/2)
      SortintoB(A, B, 1+(s+e)/2, e)
      Merge(B[s:(s+e)/2], B[1+(s+e)/2:e], A)
```

Merge Sort

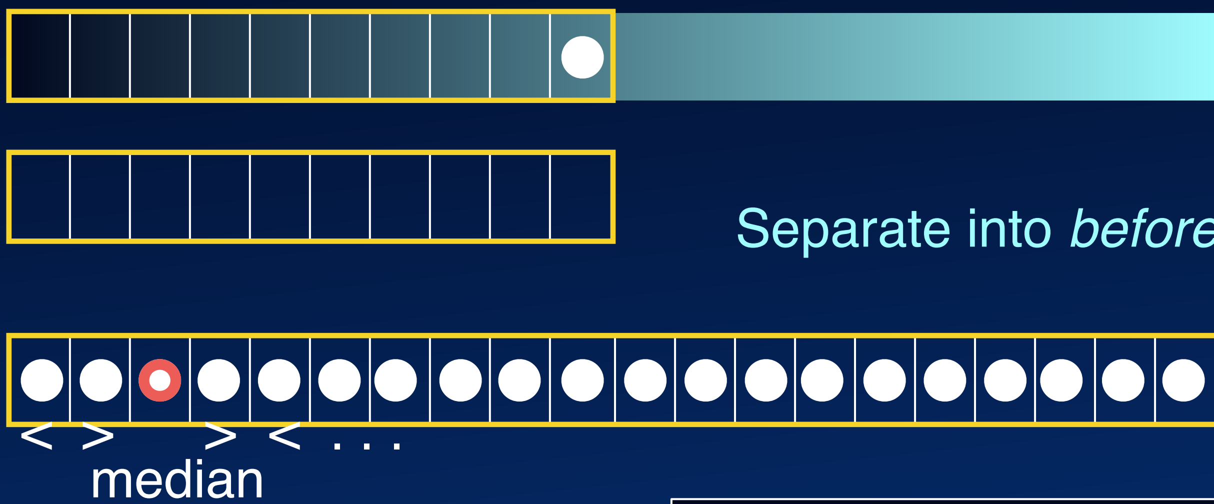
- Parallel?
- $T(n) = T(n/2) + O(n)$

T/F

- Merge sort is stable

Format: t
mailto: col106quiz@cse.iitd.ac.in

Quick Sort



Separate into *before* and *after* sets

O(n)

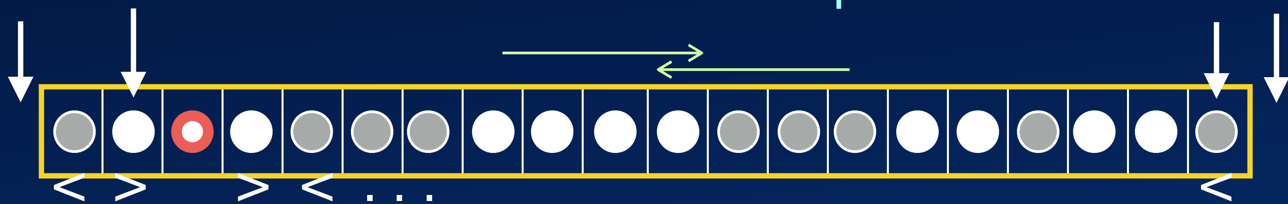
```
sort(list):
    while(i < n):
        if(elem[i] before median)
            append to the first part
        else
            append to the second part
    sort(first part)
    sort(second part)
```

$$T(n) = 2T(n/2) + O(n)$$
$$= O(n \log n)$$

Quick Sort

Duplicate Keys?

Separate into *before* and *after* sets   In-place



O(n)

```
partition(A, s, e, pivot):
  repeat:
    do s++
    while(elem[s] before median)
    do e--
    while(elem[e] after median)
    if(s < e) swap(@s, @e)
    else break out of loop
```

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

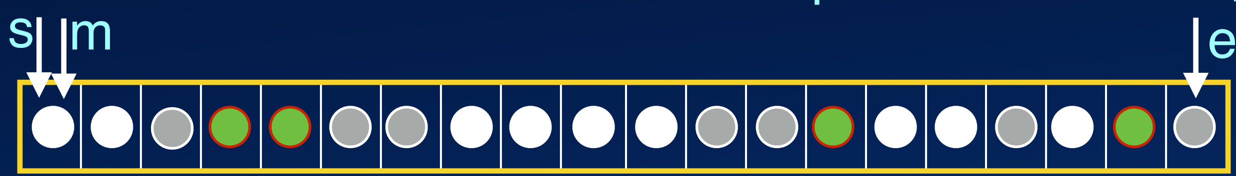
$= O(n \log n)$

Quick Sort

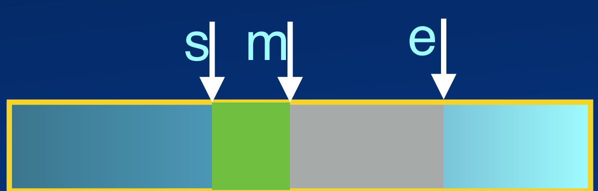
Duplicate Keys: bring to middle

O(n)

Separate into *before*, *equal*, and *after* sets



s separates *before* & *equal*
m separates *equal* & *unchecked*
e separates *unchecked* & *after*



Swap m with e, if m is *before*

Swap m with s, if m is *after*

```
partition(A, s, e, pivot):
    m = s;
    while m ≤ e:
        if A[m] < pivot:
            swap A[s] and A[m]
            s++; m++;
        else if A[m] > pivot:
            swap A[m] and A[e]
            e--;
        else: // A[m] == pivot
            m++;
    return s, m
```

$$T(n) = 2T(n/2) + O(n)$$
$$= O(n \log n)$$

T/F?

Average complexity of quick sort is $\Theta(n^2)$

Format: f
mailto: col106quiz@cse.iitd.ac.in

Lower bound on Sorting

Comparison Model

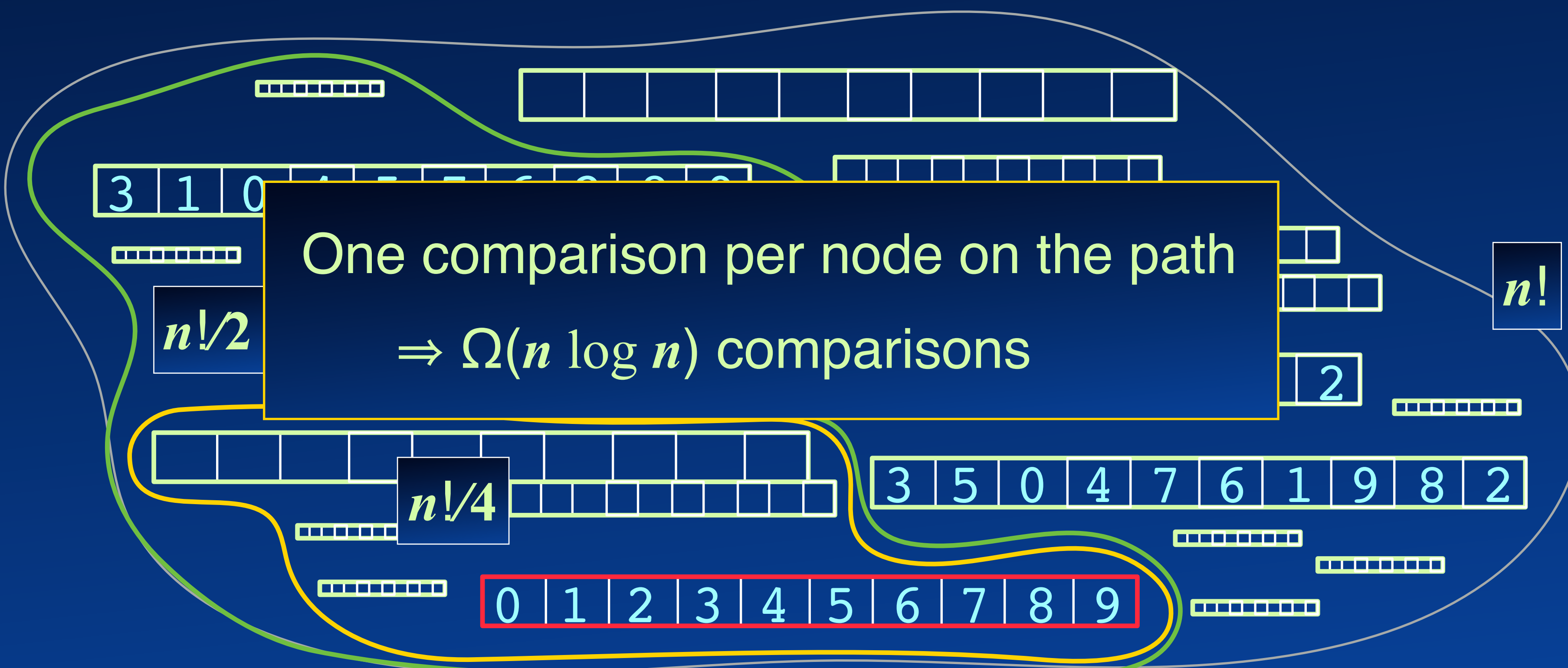
Compare i, j

i before j

Compare k, l

k before l

Minimum Height
of this decision tree = $\log(n!)$



One comparison per node on the path

$\Rightarrow \Omega(n \log n)$ comparisons

Q-Sort Average Complexity

probability (Good pivot) = 1/2

Divides into a ratio with balance more than 1/4 : 3/4

$\Rightarrow$ Recursion depth = $\log_{4/3} n$

if good pivot at every level

Expect half the pivots are good

$\Rightarrow$ Expected depth = 2 $\log_{4/3} n$ = O(log n)

probability (Complexity is O(n log n)) $\geq$ $\boxed{1 - 1/n}$

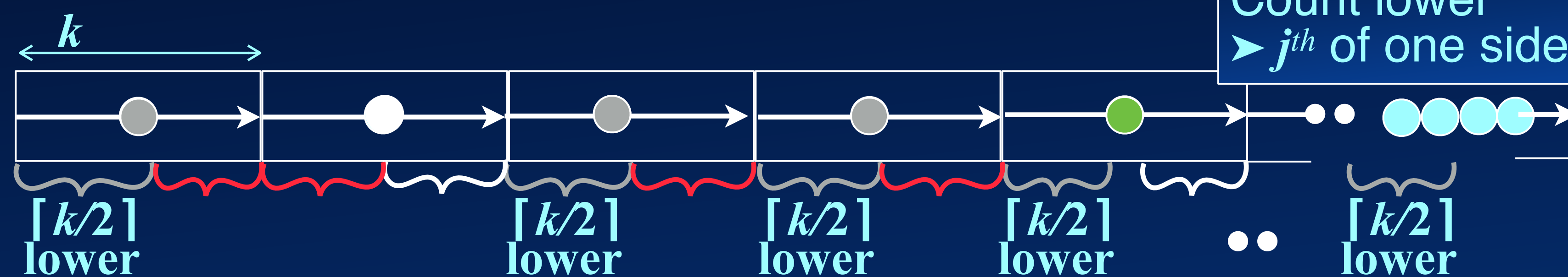
Linear separation time at each level (O(n)) $\Rightarrow$

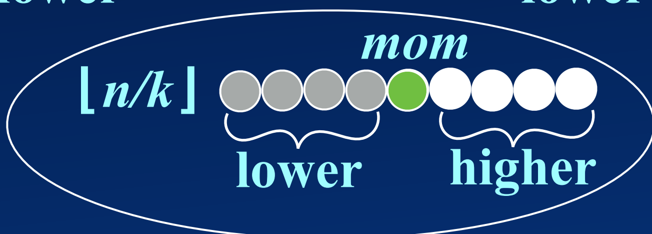
Expected time complexity is O(n log n)

Median Find (Select)

$T(n) = O(k \log k) + O(n/k) + T(n/k) + O(n) + T(\lceil 3n/4 \rceil)$

sort groups of k
➤ mom
Count lower
➤ j^{th} of one side





$\Rightarrow \lceil k/2 \rceil * \lfloor n/2k \rfloor \Rightarrow n/4$ are definitely $\leq$ *mom*

$\sim n/4$ are definitely $\geq$ *mom*

Count the "≤" ones: p of them

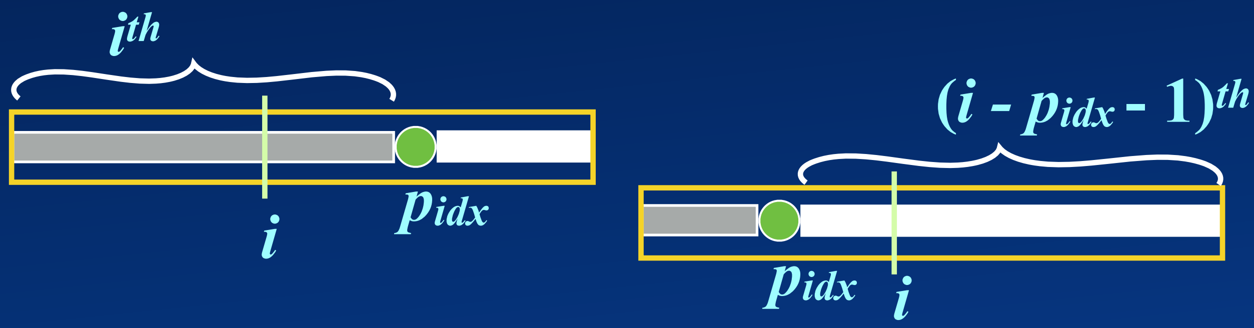
$\Rightarrow$ *mom* is the p^{th} element

Find i^{th} element in the *lower* set if $(p > i)$

Find $(i\text{-}p)^{th}$ element in the *higher* set if $(p < i)$

Quick Select

- Pick random pivot
- Partition array
  - < pivot
  - > pivot



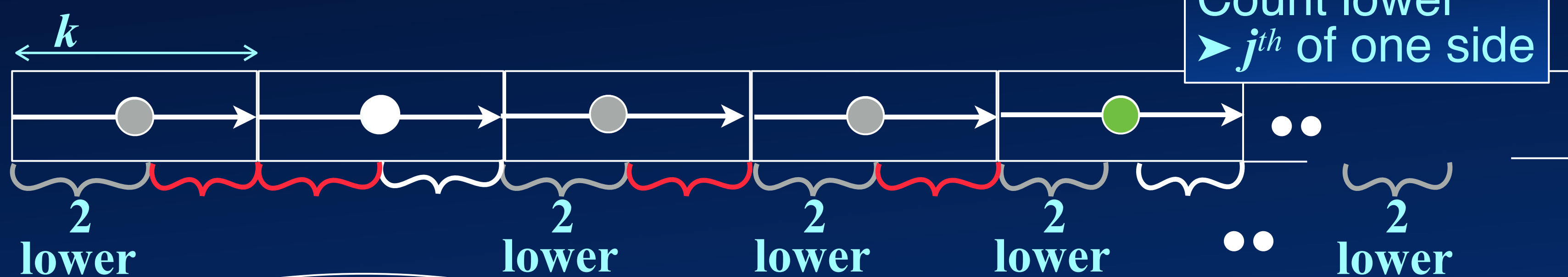
Expected linear complexity

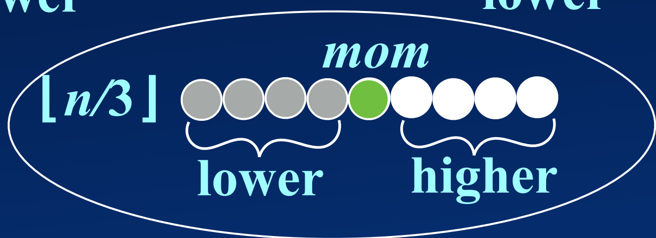
Expected T(n) = O(n) + T(n/2)

Median Find (Select)

$T(n) = O(n) + T(n/3) + T(\lceil 2n/3 \rceil)$

sort groups of 3
➤ mom
Count lower
➤ j^{th} of one side





$\Rightarrow$ $\lfloor n/3 \rfloor$ are definitely $\leq$ ***mom***
$\lfloor n/3 \rfloor$ are definitely $\geq$ ***mom***

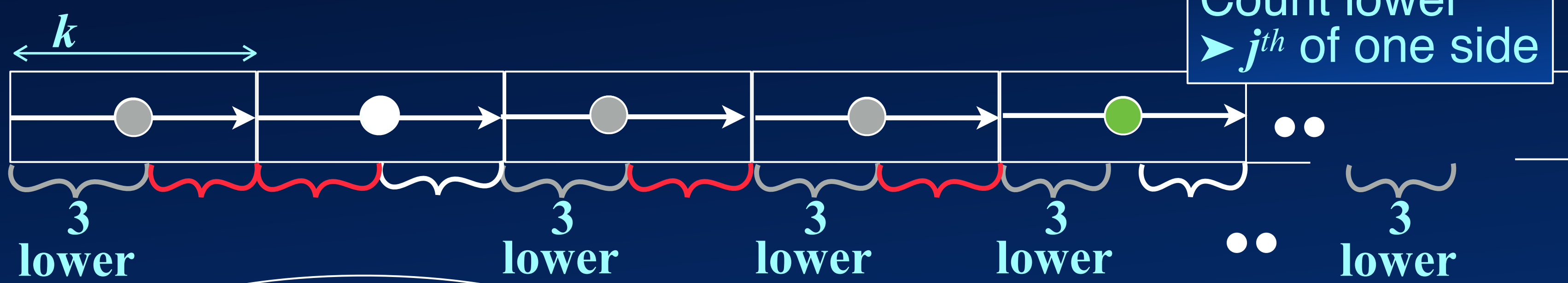
Count the "≤" ones: p of them
$\Rightarrow$ ***mom*** is the p^{th} element

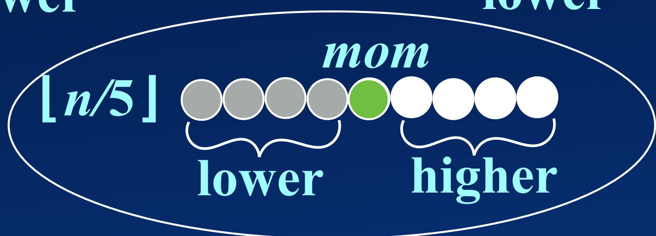
Find i^{th} element in the *lower* set	if $(p > i)$
Find $(i\text{-}s)^{th}$ element in the *higher* set	if $(p < i)$

Median Find (Select)

$T(n) = O(n) + T(n/5) + T(\lceil 7n/10 \rceil)$

sort groups of 5
➤ mom
Count lower
➤ j^{th} of one side





$\Rightarrow$ $\lfloor 3n/10 \rfloor$ are definitely $\leq$ ***mom***
$\lfloor 3n/10 \rfloor$ are definitely $\geq$ ***mom***

Count the "≤" ones: p of them
$\Rightarrow$ ***mom*** is the p^{th} element

Find i^{th} element in the *lower* set if $(p > i)$
Find $(i\text{-}s)^{th}$ element in the *higher* set if $(p < i)$

Median-Find Analysis

$T(n) = O(k \log k) + O(n/k) + T(n/k) + O(n) + T(3n/4)$

$T(\boldsymbol{n}) = O(\boldsymbol{n}) + T(\boldsymbol{n}/5) + T(7\boldsymbol{n}/10)$

$T(\boldsymbol{n}) \le c\boldsymbol{n} + T(\boldsymbol{n}/5) + T(7\boldsymbol{n}/10) \quad \forall\ \boldsymbol{n} \ge \boldsymbol{n}_0$

Dropping c and $\boldsymbol{n} \ge \boldsymbol{n}_0$ from notation for succinctness:

$T(\boldsymbol{n}) \le \boldsymbol{n} + \underline{\boldsymbol{n}/\mathbf{5} + T(\boldsymbol{n}/25) + T(7\boldsymbol{n}/50)} + \underline{7\boldsymbol{n}/10 + T(7\boldsymbol{n}/50) + T(49\boldsymbol{n}/100)}$

$\le \boldsymbol{n} + \boldsymbol{n}/\mathbf{5} + 7\boldsymbol{n}/10 + T(\boldsymbol{n}/25) + 2T(7\boldsymbol{n}/50) + T(49\boldsymbol{n}/100)$

$\le \boldsymbol{n} + \boldsymbol{n}/\mathbf{5} + 7\boldsymbol{n}/10 + (1/\mathbf{5})^2\boldsymbol{n} + T(\boldsymbol{n}/125) + T(7\boldsymbol{n}/250)$

$+ 2(1/\mathbf{5})(7/10)\boldsymbol{n} + 2T(7\boldsymbol{n}/250) + 2T(49\boldsymbol{n}/500)$

$+ (7/10)^2\boldsymbol{n} + T(49\boldsymbol{n}/500) + T(343\boldsymbol{n}/1000)$

$\le \boldsymbol{n} + \boldsymbol{n}/5 + 7\boldsymbol{n}/10 + (1/5)^2\boldsymbol{n} + (7/10)^2\boldsymbol{n} + 2(1/\mathbf{5})(7/10)\boldsymbol{n}$

$+ T(\boldsymbol{n}/\mathbf{5}^3) + T((7/10)^3\boldsymbol{n}) + 3T((7/10)(1/5)^2\boldsymbol{n}) + 3T((7/10)^2(1/5)\boldsymbol{n})$

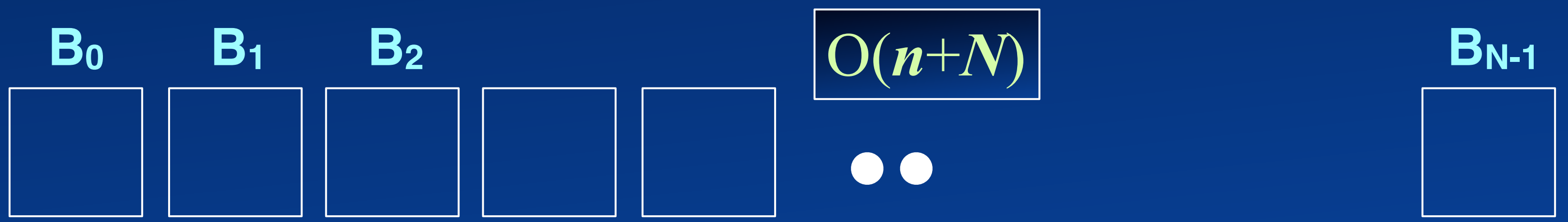
$\le \boldsymbol{n} + \boldsymbol{n}[(1/5 + 7/10) + (1/5 + 7/10)^2 + (1/5 + 7/10)^3 \ldots]$

$\le \boldsymbol{n} + \boldsymbol{n}[(9/10) + (9/10)^2 + (9/10)^3 \ldots]$

$\le C\boldsymbol{n}\ \forall\ \boldsymbol{n} \ge \boldsymbol{n}_0$

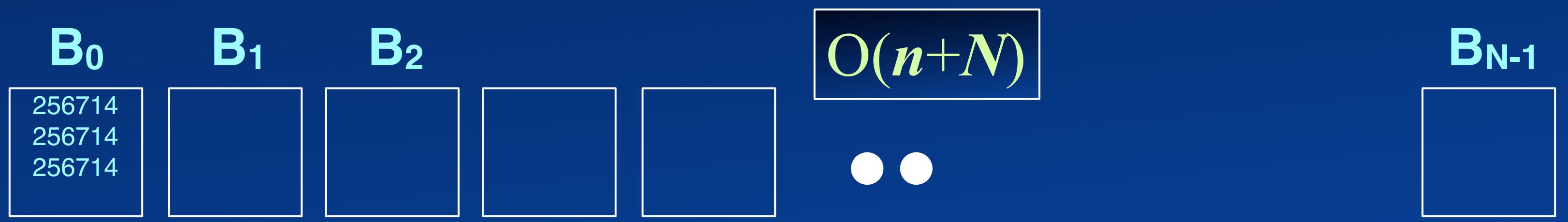
Bucket Sort

- Each key goes into a bucket based on some property of the key
- Key → Bucket mapping takes O(1) time
- Keys in bucket i are before those in bucket j
  - if $i < j$
- In general, consider n keys and N buckets
  - What if all keys map in the range $0{:}N\text{-}1$?



Radix Sort

- Decompose each key into k parts
  - Each part maps to the range 0:N-1
- Bucket-sort based on part 0
- For each bucket:
  - Bucket-sort based on part 1
  - i.e., create sub-buckets
- and so on ..



Radix Sort

$O(k(n+N))$

- Decompose each key into k parts
  - Each part maps to the range 0:N-1
- Stably Bucket-sort based on part k-1
- Stably Bucket-sort based on part k-2
  - and so on ..

123	212	212	123
234	322	213	124
233	123	214	212
212	233	322	213
124	213	123	214
214	234	124	233
213	124	233	234
322	214	234	322