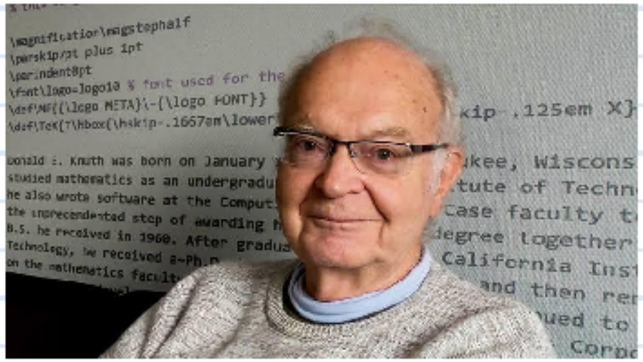
CS2700 : PROGRAMMING and DATA STRUCTURES.

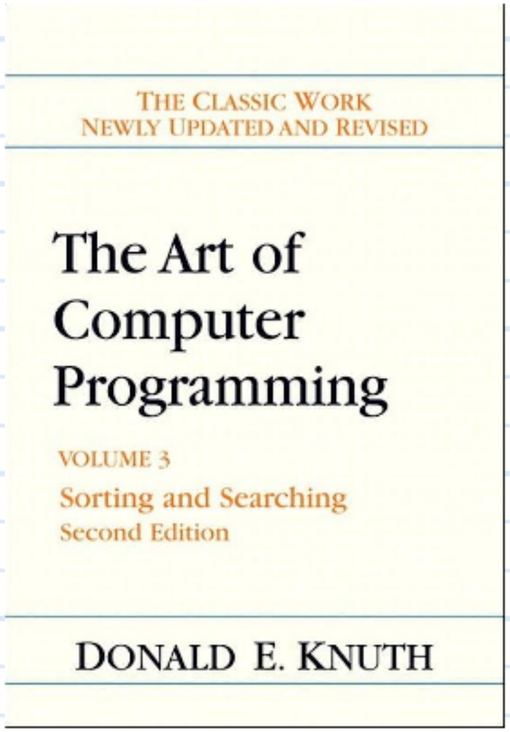
TODAY : TWO SORTING ALGORITHMS.

1. HEAPSORT

2. QUICKSORT

SORTING : A FUNDAMENTAL PRIMITIVE.





- NEED TO OPTIMIZE SORTING ALGOs CANT BE EMPHASISED ENOUGH
- A RANGE OF SORTING ALGOS
  - different properties satisfied
- what algos have you studied?

SORTING : A FUNDAMENTAL PRIMITIVE.

INPUT ; AN ARRAY OF INTEGERS

OUTPUT : INPUT SORTED IN (INCREASING) ORDER

POSSIBLE CONSTRAINTS / VARIANTS

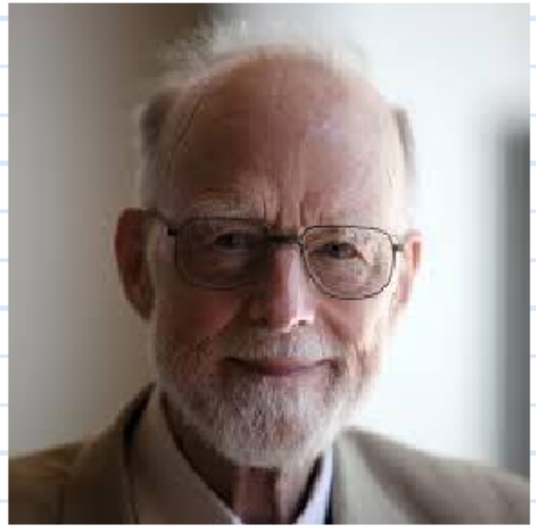
HEAP SORT

O(n) · Initialize a min heap with array elements.

O(n log n) · Repeatedly remove min element (using deletemin) and store it in an aux. array (which is the output)

Running time : O(n log n)

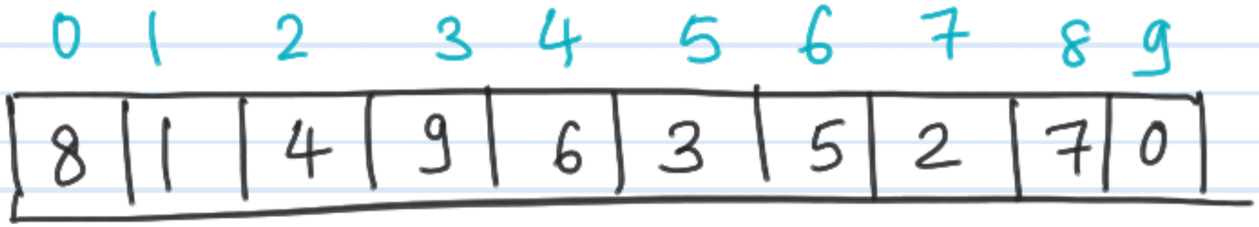
QUICK SORT



- A very fast in practice algo
- Needs careful implementation
- Has $O(n^2)$ worst case running time

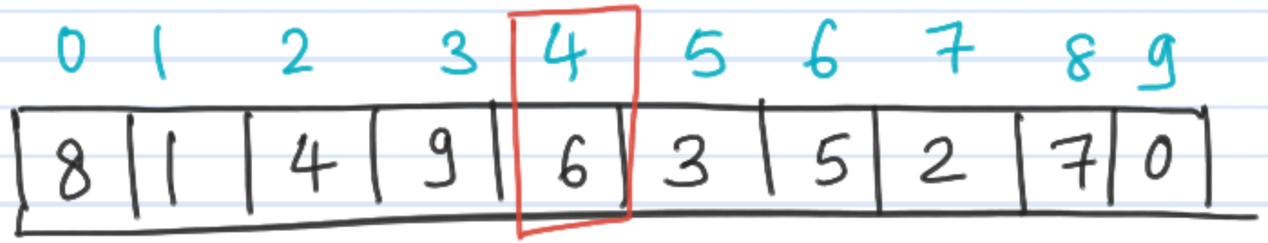
  $O(n \log n)$ average case complexity.

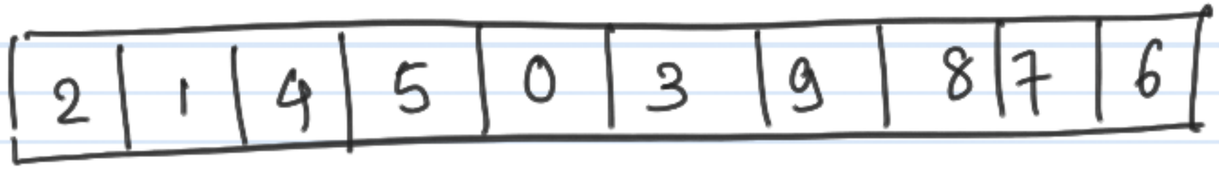
QUICK SORT : BASIC IDEA



- pick a pivot [let us say A[4]]
- find position of pivot in sorted array [how??]

QUICK SORT : BASIC IDEA



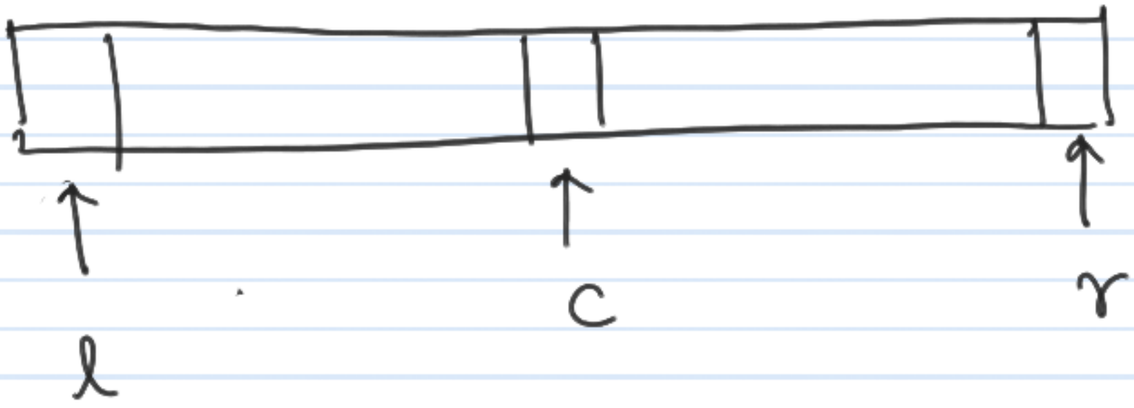


- pick a pivot [let us say A[4]]
- find position of pivot in sorted array [how??]
  - partition the array
- recursively sort left, sort right.

QUICK SORT

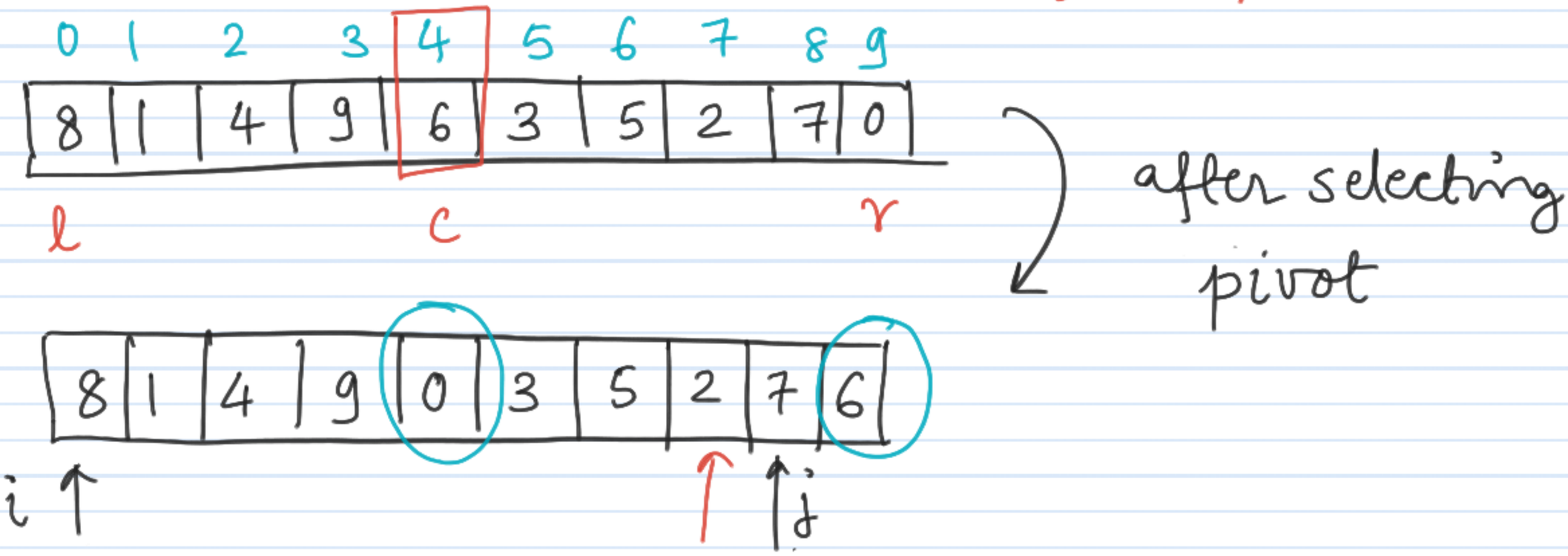
- Find a pivot
- partition the array based on pivot
  - multiple choices for seleting pivot,
    - first / last element
    - center element
    - random index
    - median of 3 elements

QUICK SORT : Selecting a pivot



- sort A[l], A[c], A[r]
- pick middle of the 3 as pivot
- get pivot out of the way.
  ↳ this is a side effect.

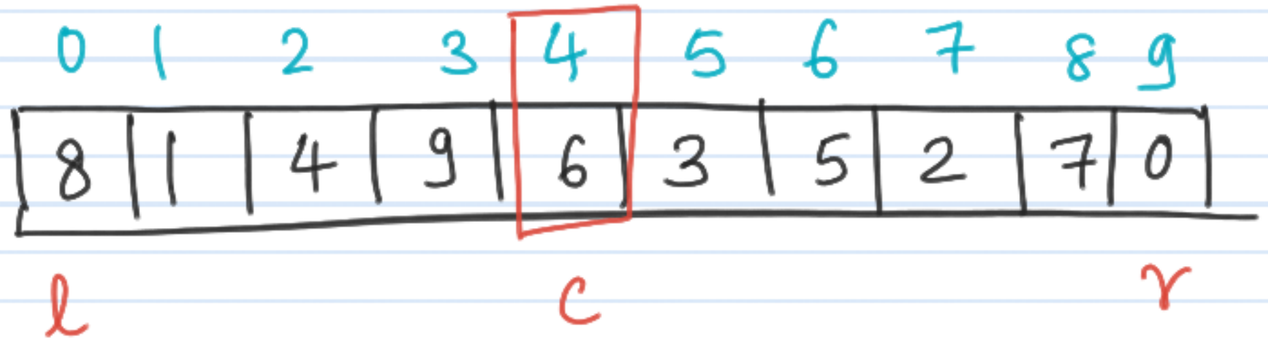
QUICK SORT : selecting a pivot



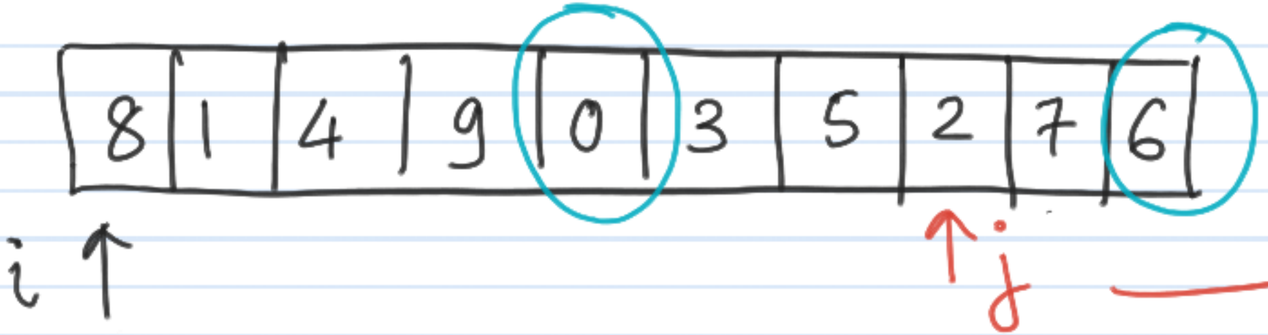
Partition :

- move all "small" elements to "left"
- move all "large" elements to "right"

QUICK SORT : selecting a pivot



after selecting pivot

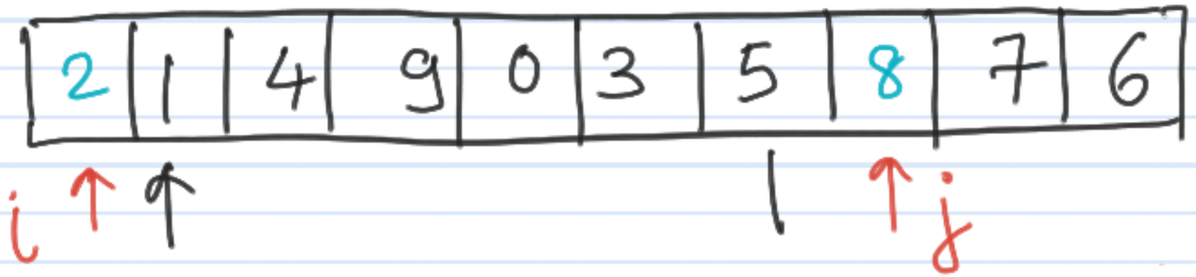


→ swap A[i] A[j]

Partition :

- move all "small" elements to "left"
- move all "large" elements to "right"

QUICK SORT : selecting a pivot




```
while ( A[i]  < pivot) {  i++;  }
while ( A[j]  > pivot) {  j--;  }
if (i < j)
        ( swap A[i], A[j])
else ____ ??
```

```
Qsort ( A[] , left , right) {

    Pivot = getPivot ( A , left , right)
                ↳ has side effect.

    i = left ;  j = right - 1;
    while ( 1 ) {
        while (A[i] < pivot) i++;
        while ( A[j] > pivot) j --;
        if( i < j) swap (_ _) else ———
    }
    swap ( A[i] , A[right]); Qsort (A, —— , —— );
}                            Qsort (A, — ,—— );
```

```
Qsort ( A[] , left , right) {

    Pivot = getPivot ( A , left , right)
                ↳ has side effect.

    i = left ;  j = right - 1;
    while ( 1 ) {
        while (A[i] < pivot) i++;
        while ( A[j] > pivot) j --;
        if( i < j ){swap (_ _) } else ———
    }
    swap ( A[i] , A[right]); Qsort (A, —, —);
}                            Qsort (A, — ,— );
```

Running time analysis for quicksort

$$T(n) = T(i) + T(n-i-1) + c \cdot n$$

$i = |S| \rightarrow$ determined by the pivot

Best case : pivot happens to be the median

Worst case : pivot happens to be smallest or largest

Write down recurrences for both.

Running time analysis for quicksort

Best case : $T(n) = 2T(n/2) + c \cdot n$

Worst case : $T(n) = T(n-1) + c \cdot n$

One more sorting algorithm : Mergesort

Basic idea : Suppose we have 2 sorted arrays

A[ ] = 1, 13, 24, 26

B[ ] = 2, 15, 27, 38

can we created a single "merged" array out of these ?

One more sorting algorithm : Mergesort

Basic idea : Suppose we have 2 sorted arrays

A[ ] = 1, 13, 24, 26

B[ ] = 2, 15, 27, 38

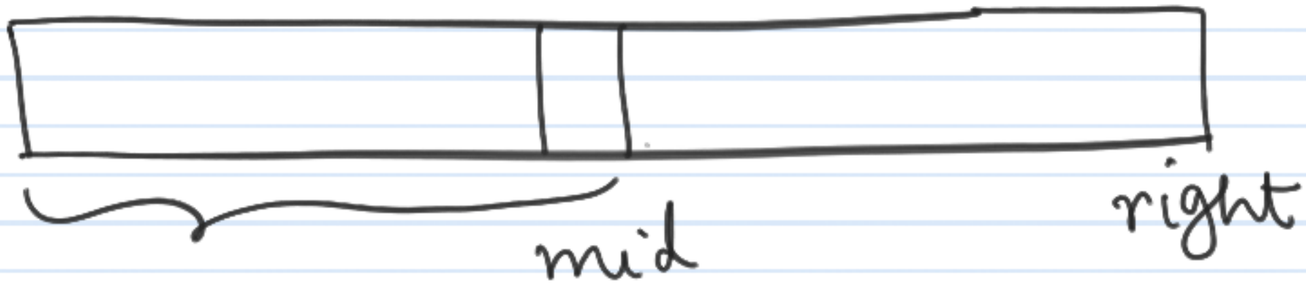
- C[ ] = [1, 2, 13, 15, ...]
- Compare A[i], B[j]
- Populate C[k]
- Which indices advance?

Mergesort : Recursive part

```
void msort ( int A [], int left, int right ) {

    if ( left >= right ) return;
    mid = (left + right) / 2 ;
    msort ( A , left , mid ) ;
    msort ( A, mid+1, right ) ;

    merge ( A , left, mid, right) ;

}
```

Merge function

```
void merge( A[], left, mid , right) {
```



$$T(n) = 2T(n/2) + c \cdot n$$