

## Arrays Lecture 4

Wednesday, 17 July 2024

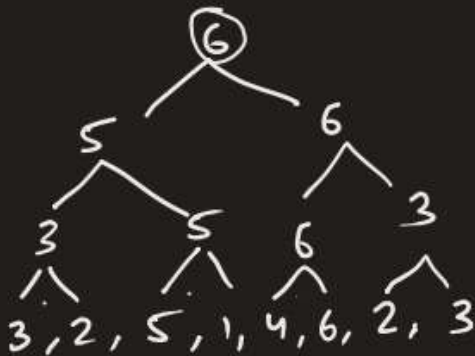
6:05 AM

Find the maximum element in an array with minimum number of comparisons.



$$\# \text{ comparisons} = \boxed{n-1}$$

Linear Traversal



7 comparisons

Tournament method

Sorting

$n \log n$  comparisons

$\text{max} \leftarrow \text{arr}[0]$

for ( $i: 1 \rightarrow n$ )

if ( $\text{max} < \text{arr}[i]$ ) {

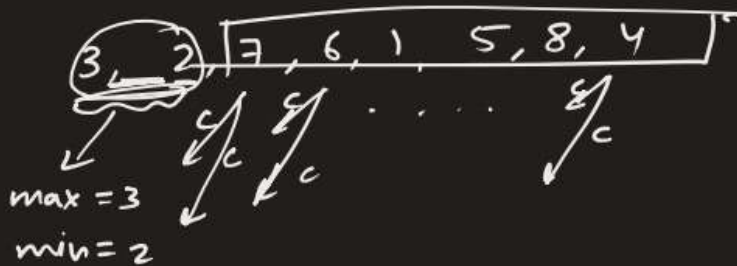
$\text{max} = \text{arr}[i]$

}

$n-1$  comparisons

Find the maximum & minimum element of the array  
in min. number of comparisons

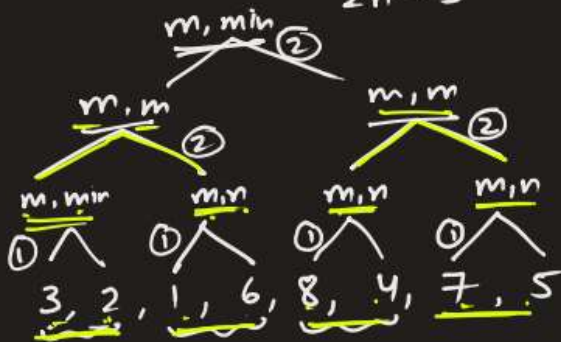
Linear  
Search



$$\# \text{ comparisons} = 2(n-2) + 1$$

$$2n - 3$$

Tournament  
Method

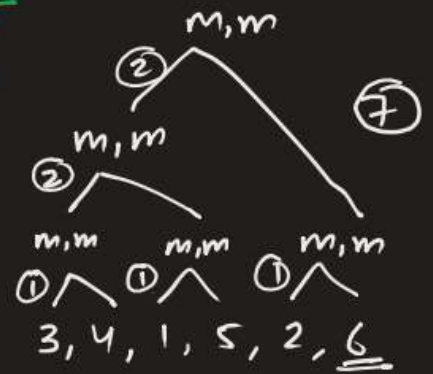


$$\# \text{ comparisons} = 10$$

$$T(n) = \begin{cases} 2T(n/2) + 2 & n \equiv \text{pow of } 2 \\ \underline{1}, & \underline{n=2} \\ 0, & \underline{n=1} \end{cases}$$

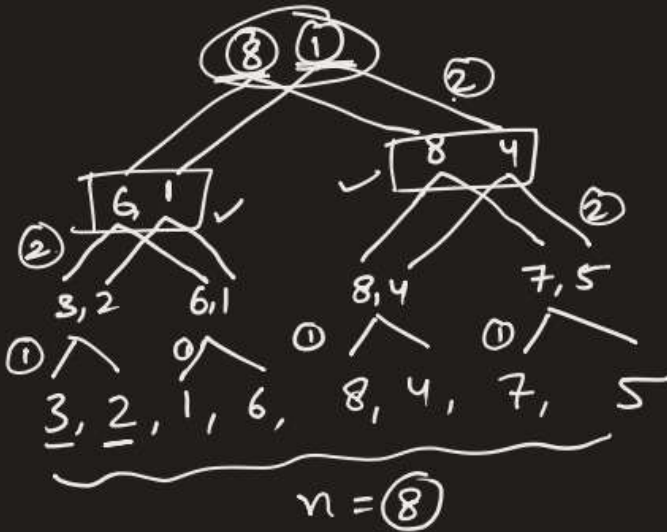
$$\Rightarrow T(n) = \frac{3n}{2} - 2$$

$$T(n) = \begin{cases} T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + 2 & n \equiv \text{Not a pow of } 2 \\ 1, & n=2 \\ 0, & n=1 \end{cases}$$



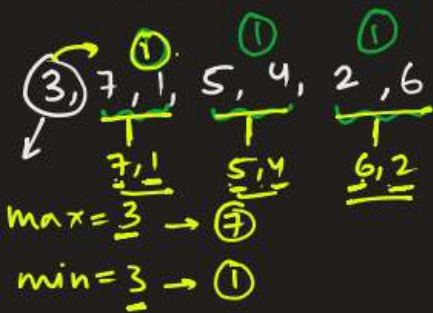
$$\frac{3n}{2} - 2$$

$$3 \times 3 - 2 = \underline{7}$$



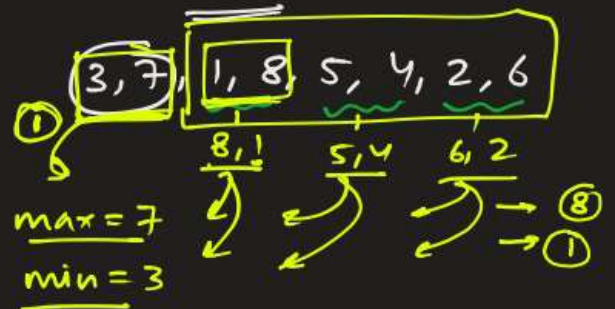
## Comparison in Pairs ★ ✓

odd case



$$\boxed{\frac{(n-1)}{2} \times 3} \checkmark$$

even case

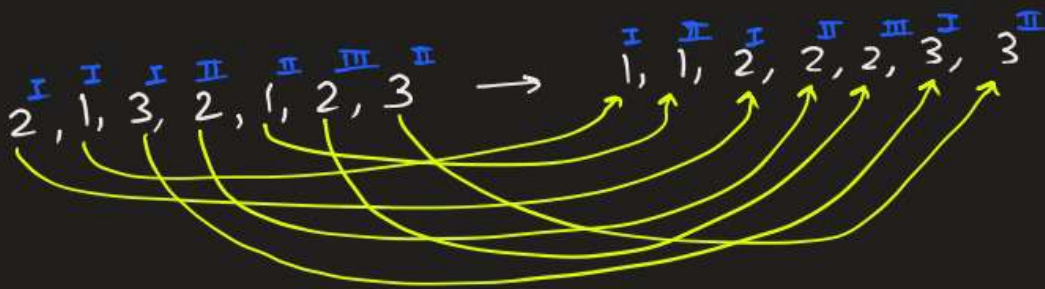


$$\begin{aligned} \# \text{ Comp} &= 1 + \frac{(n-2)}{2} \times 3 \\ &= \boxed{\frac{3n}{2} - 2} \checkmark \end{aligned}$$

# Sorting

## Terminology

- > **In-place / Internal Sorting :-** It uses constant extra Space. It sorts the array by only modifying the orders of the elements within the list.
- > **Out-of-place / External sorting :-** It uses extra space other than the array itself.
- > **Stable Sorting :-** When two same data appear in some order, then in sorted array also, the same data appears in same order.



- > **Unstable Sorting :-** When two same data appear in different relative ordering after sorting.



## Selection Sort

Simple and efficient sorting algorithm which repeatedly selects the smallest (or largest) element from the unsorted portion of the list and moves it to the sorted portion of the list.

Selection-Sort ( $A, n$ ) {

  for ( $k: \underline{1 \rightarrow n-1}$ ) {

$\text{min} = A[k]$

$\text{loc} \leftarrow k$

    for ( $j: \underline{k+1 \rightarrow n}$ ) {

      if ( $\text{min} > A[j]$ ) {

$\text{min} = A[j]$

$\text{loc} \leftarrow j$

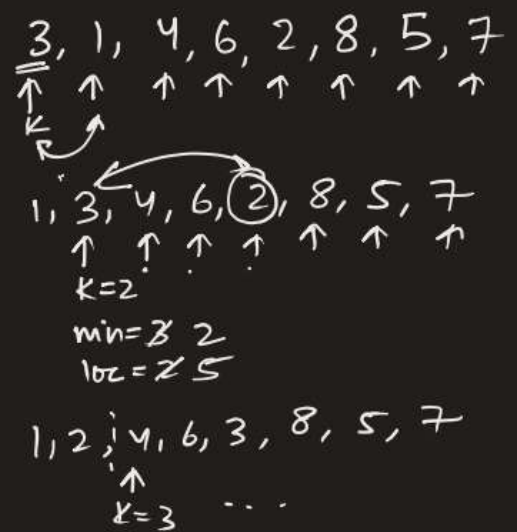
      }

    }

    swap( $A[k], A[\text{loc}]$ )

  }

}

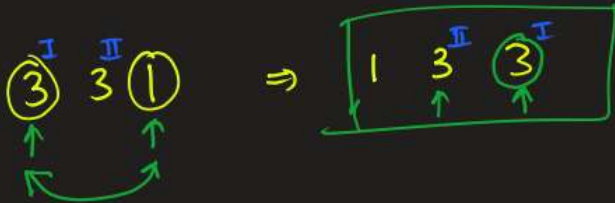


Time Complexity :-  $\boxed{\begin{array}{l} \text{Best Case :- } O(n^2) \\ \text{Worst Case :- } O(n^2) \end{array}}$

In place / out-of-place :-  $\boxed{\text{Inplace}}$

Extra Space :-  $\boxed{O(1)}$

Stable / unstable :-  $\boxed{\text{Unstable}}$



## Bubble Sort

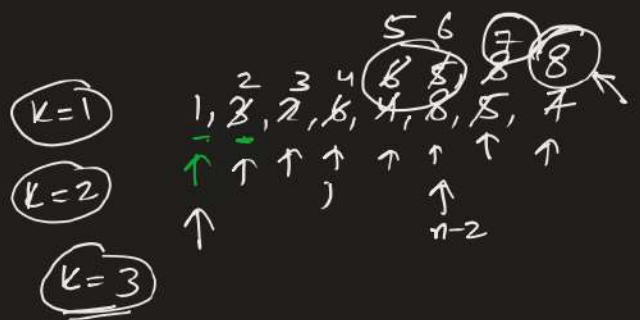
↳ This is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they were in wrong order.

```

BubbleSort(A, n) {
  → for(k: 1 → n-1) {
    → for(j: 1 → n-k) {
      if (A[j] > A[j+1])
        swap(A[j], A[j+1])
    }
  }
}

```

$T = \text{Best} = O(n^2)$   
 $\text{Worst} = O(n^2)$



$j: 1 \rightarrow n-1 \quad k=1$   
 $j: 1 \rightarrow n-2 \quad k=2$   
 $j: 1 \rightarrow n-3 \quad k=3$   
 $\vdots$



BubbleSort(A, n) {

→ for(k: 1 → n-1) {

flag = 0

→ for(j: 1 → n-k) {

if (A[j] > A[j+1]) {

swap(A[j], A[j+1])

flag = 1 }

}

if (flag == 0) break;

}

}

1, 2, 3, 4  
↑ ↑ ↑ k=1

Time :-

Best:  $O(n)$

Worst :-  $O(n^2)$

Average

<sup>I</sup>3, 2, <sup>II</sup>3, 4, <sup>III</sup>3  
↖ ↗  
2, <sup>I</sup>3, <sup>II</sup>3, 4, <sup>III</sup>3  
↖ ↗

In-place / out-of-place :- Inplace

Stable / unstable :- Stable