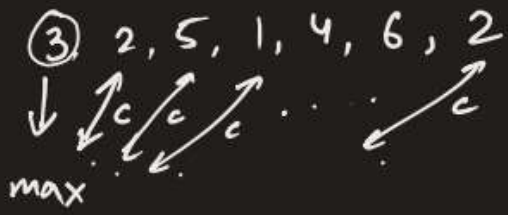
Arrays Lecture 4

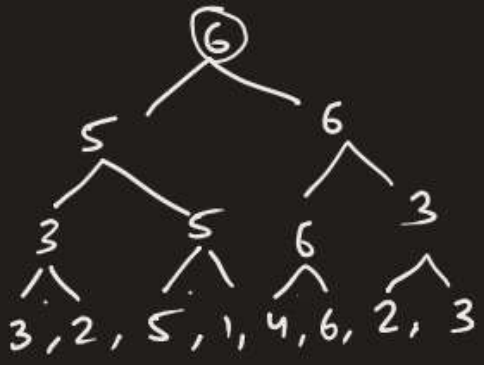
Wednesday, 17 July 2024 6:05 AM

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



comparisions = $n-1$

Linear Traversal



7 comparisions

Tournament method

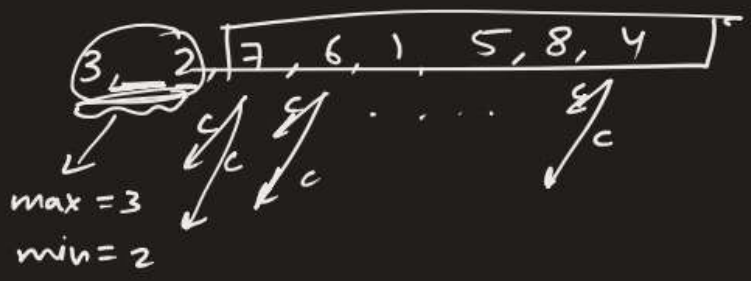
Sorting

$n \log n$ comparisions

```
max ← arr[0]
for (i: 1→n)                    ——→ n-1 comparisions
    if (max < arr[i]) {
        max = arr[i]
    }
```

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

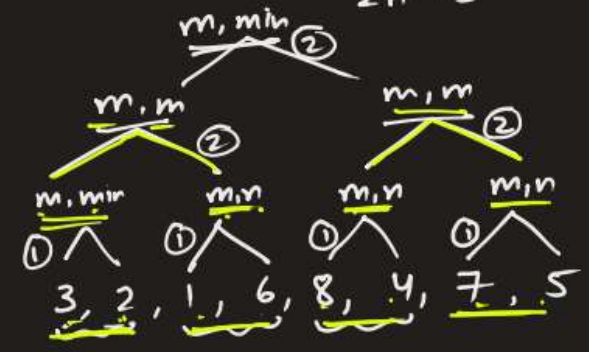
Linear Search



comparisions = $2(n-2)+1$

$2n-3$

Tournament Method



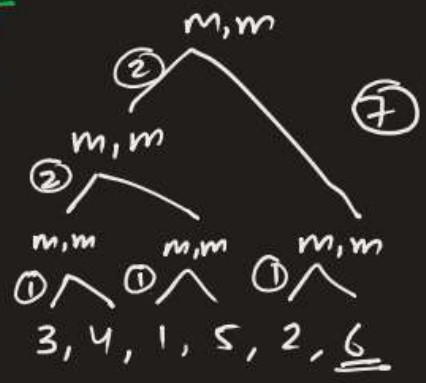
comparisions = 10

$$T(n) = \begin{cases} 2T(n/2) + 2 \\ 1, & n = 2 \\ 0, & n = 1 \end{cases} \Rightarrow \boxed{T(n) = \frac{3n}{2} - 2}$$

n ≡ pow of 2

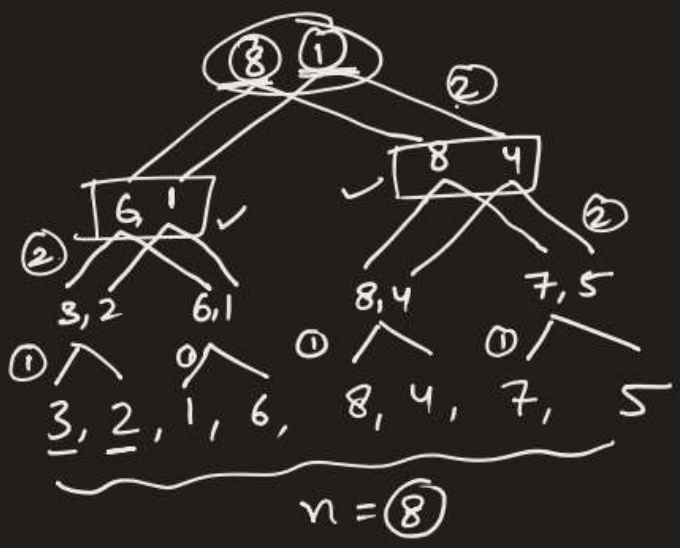
$$T(n) = \begin{cases} T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + 2 \\ 1, & n = 2 \\ 0, & n = 1 \end{cases}$$

n ≡ Not a pow of 2



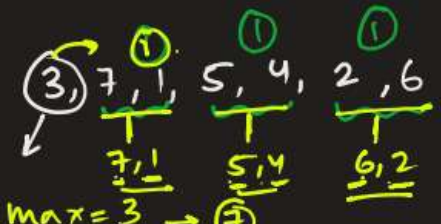
$\frac{3n}{2} - 2$

$3 \times 3 - 2 = 7$



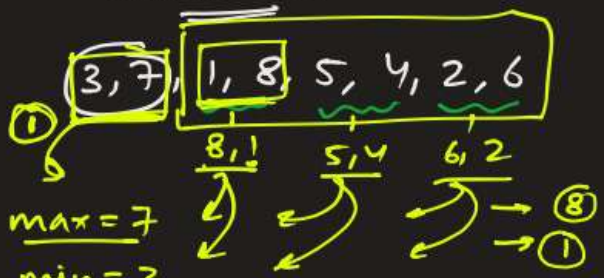
Comparision in Pairs ★ ✓

odd case



$$\frac{(n-1)}{2} \times 3$$ ✓

even case



$$\#\text{comp} = 1 + \frac{(n-2)}{2} \times 3$$

$$= \frac{3n}{2} - 2$$ ✓

—x—

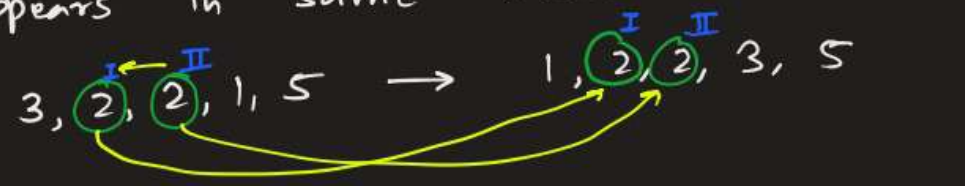
Sorting

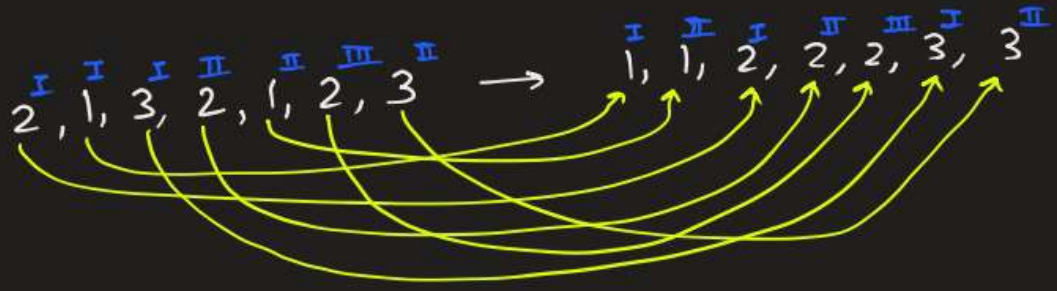
Terminology

> 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.



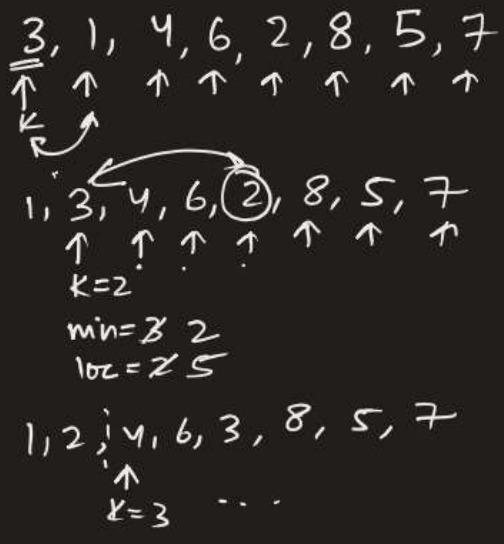


> Unstable Sorting :- when two same data appear in different relative ordering after sorting.

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: 1 → n-1) {
        min = A[k]
        loc ← k
        for (j: k+1 → n) {
            if (min > A[j]) {
                min = A[j]
                loc ← j
            }
        }
        swap( A[k], A[loc])
    }
}
```

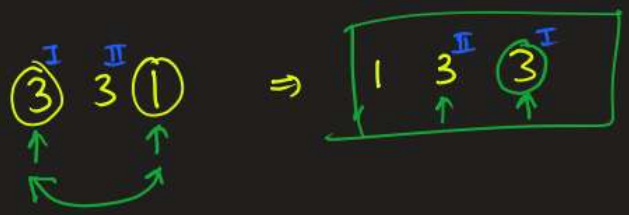


Time Complexity :- Best Case :- $O(n^2)$
Worst Case :- $O(n^2)$

Extra Space :- $O(1)$

In place / out-of-place :- Inplace

Stable / Unstable :- Unstable



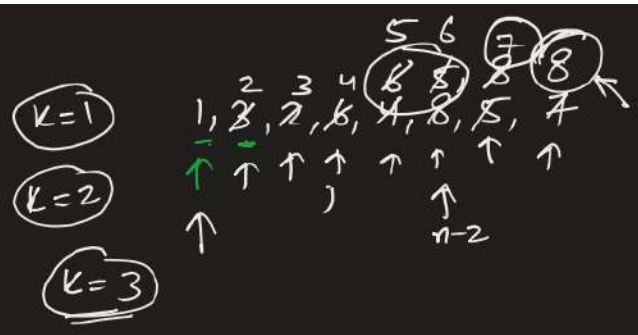
Bubble Sort

↳ 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= Best = $O(n^2)$

Worst = $O(n^2)$



j: 1 → n-1 k=1

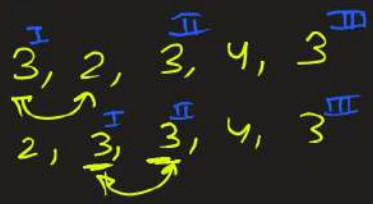
j: 1 → n-2 k=2

j: 1 → n-3 k=3

```
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



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

Stable / Unstable :- Stable