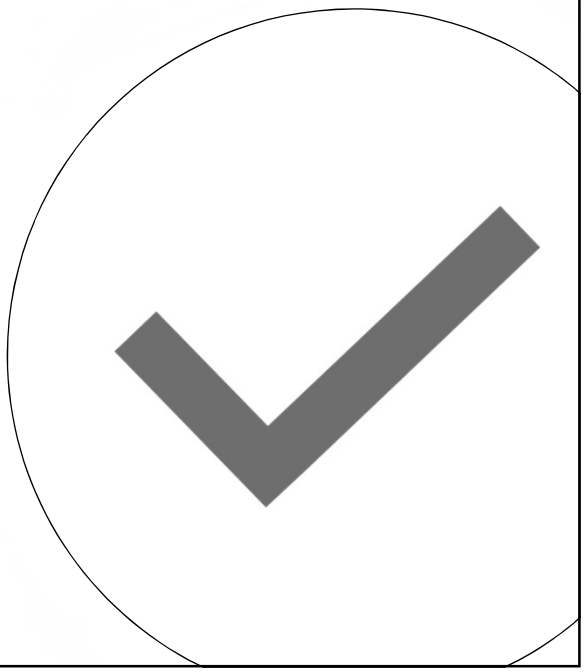
CSCI 3100

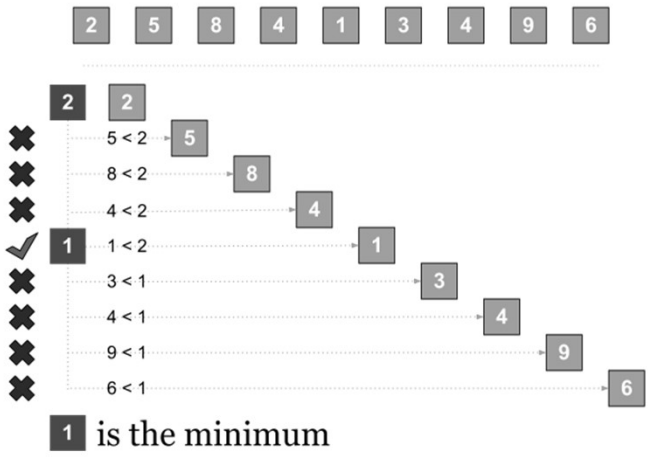
Comparison Based Algorithms



Overview

- **Analyzing** array search problems using comparisons:
  - Find min
  - Find min and max
  - Find second largest element
  - **How many comparisons does it take?**
  - **Is this the best we can do?**
- Comparison based sorting
  - Insertion sort
  - Heap sort

Find Minimum Value in an Array



Find minimum value in an array

MINIMUM(A)

1 *min* = A(1)

2 **for** i = 2 **to** A.length

3 **if** (*min* > A[i])

4 *min* = A[i]

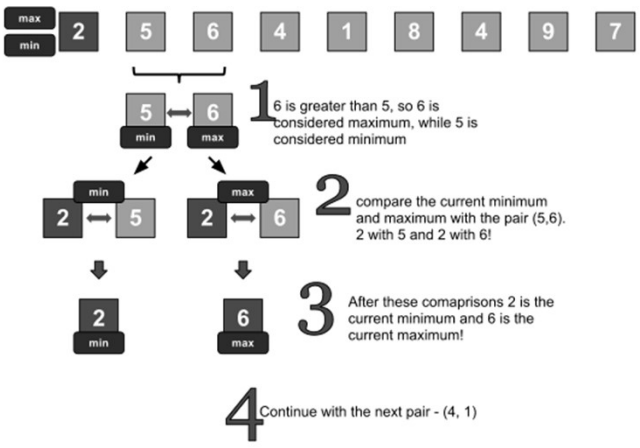
5 **return** min

- Q: How many comparisons does it take?
  - A.length-1
- Is this the best we can do?
  - Yes

Find Min and Max

- Simple solution: find min, then find max
  - Q: how many comparisons does this take?
    - (n-1)+(n-1) = 2n-2
    - If we remove the min before looking for max, then (n-1)+(n-1-1)=2n-3

Find Min and Max (faster solution)

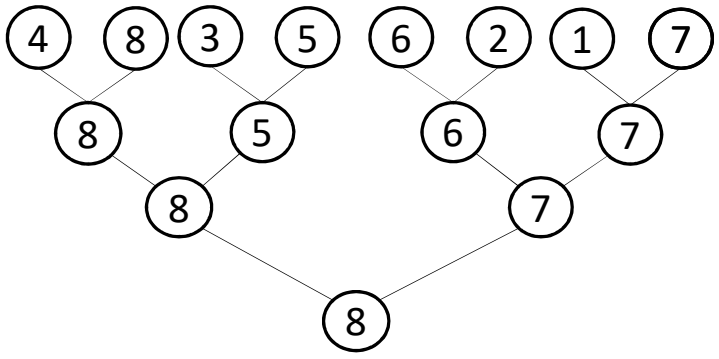


Find Min and Max (faster solution algorithm)

```
MIN_MAX(A)
1    min = A[1], max = A[1]
2    for j = 2 to A.length (j+=2)
3        if (A[j] < A[j+1])
4            temp_min = A[j]
5            temp_max = A[j+1]
6        else
7            temp_min = A[j+1]
8            temp_max = A[j]
9        if (temp_min < min)
10           min = temp_min
11       if (temp_max > max)
12           max = temp_max
13   return [min, max]
```

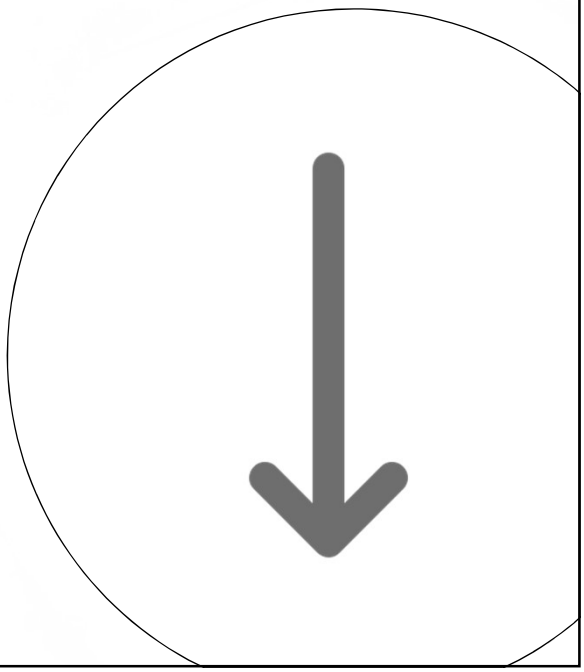
- Q: How many comparisons does it take?
  - Ceil(3n/2)
- Q: Will this work for all inputs?
  - No, will not work for even length array

Find the seconds largest element



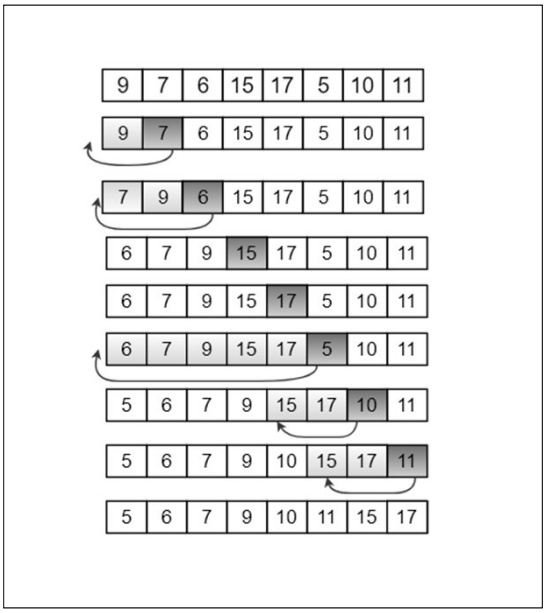
- How many comparisons does it take?
  - n + lg(n) - 2
- Is this the best we can do using element comparison?

Comparison based sorting



Insertion Sort

```
INSERTION-SORT(A)
1  for j = 2 to A.length
2      key = A[j]
3      // Insert A[j] into the sorted
           sequence A[1 .. j − 1].
4      i = j − 1
5      while i > 0 and A[i] > key
6          A[i + 1] = A[i]
7          i = i − 1
8      A[i + 1] = key
```



Analyze insertion sort

- How many comparisons?
- Can the number of comparisons be reduced?
- How many shifts?
  - If input A is sorted
  - If input A is sorted in reverse order
- Analysis of insertion sort: pages 24 - 28

```
INSERTION-SORT(A)
1  for j = 2 to A.length
2      key = A[j]
3      // Insert A[j] into the sorted
           sequence A[1 .. j − 1].
4      i = j − 1
5      while i > 0 and A[i] > key
6          A[i + 1] = A[i]
7          i = i − 1
8      A[i + 1] = key
```

INSERTION-SORT(A)		*cost*	*times*
1	**for** $j = 2$ **to** $A.length$	c_1	n
2	$key = A[j]$	c_2	$n-1$
3	// Insert $A[j]$ into the sorted sequence $A[1..j-1]$.	0	$n-1$
4	$i = j-1$	c_4	$n-1$
5	**while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6	$A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n}(t_j-1)$
7	$i = i-1$	c_7	$\sum_{j=2}^{n}(t_j-1)$
8	$A[i+1] = key$	c_8	$n-1$

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\sum_{j=2}^{n} t_j + c_6\sum_{j=2}^{n}(t_j-1) + c_7\sum_{j=2}^{n}(t_j-1) + c_8(n-1)\,.$$

$$\sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1$$

$$\sum_{j=2}^{n}(j-1) = \frac{n(n-1)}{2}$$

Best, Worst, and Average Cases

- What is the best case running time of this algorithm?
- What is the worst case running time of this algorithm?
- What is the average case running time of this algorithm?