Order Statistics

Selection Problem

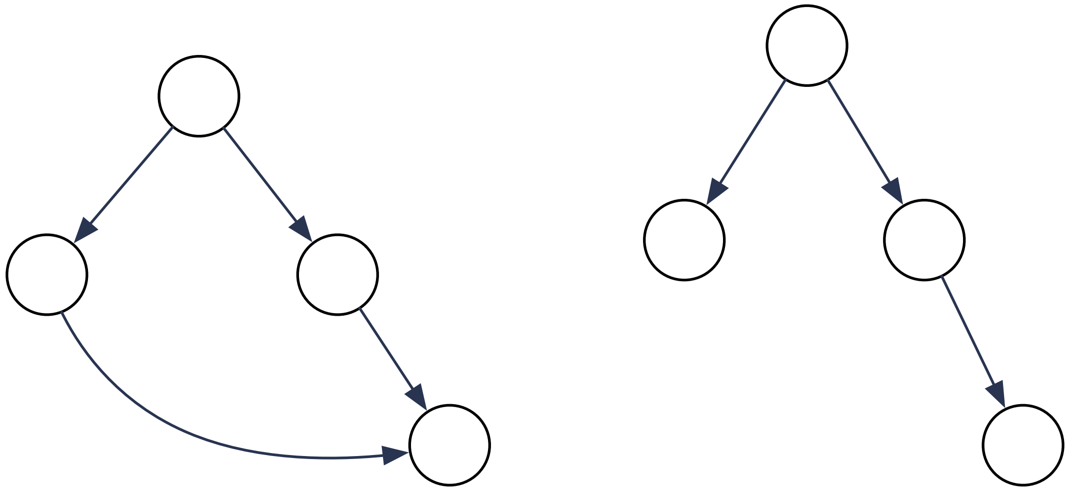
- Given *n* elements
    - Find the i^{th} smallest element

Minimum

- Determine the minimum of *n* elements
  - At least *n*-1 comparisons are needed

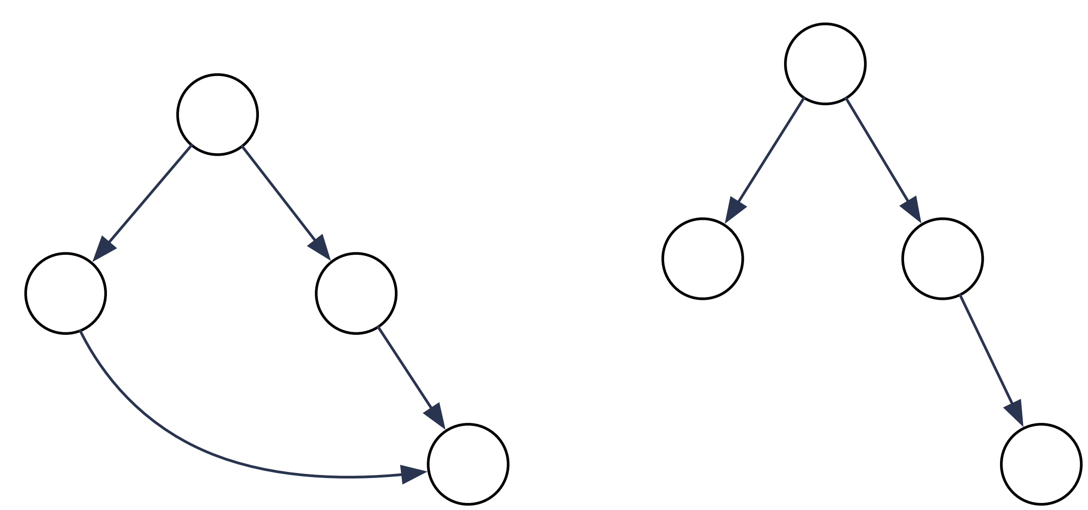
Minimum

- Determine the minimum of *n* elements
  - At least *n*-1 comparisons are needed
  - Proof: Arrange the results of the comparisons as a tournament tree with nodes being elements



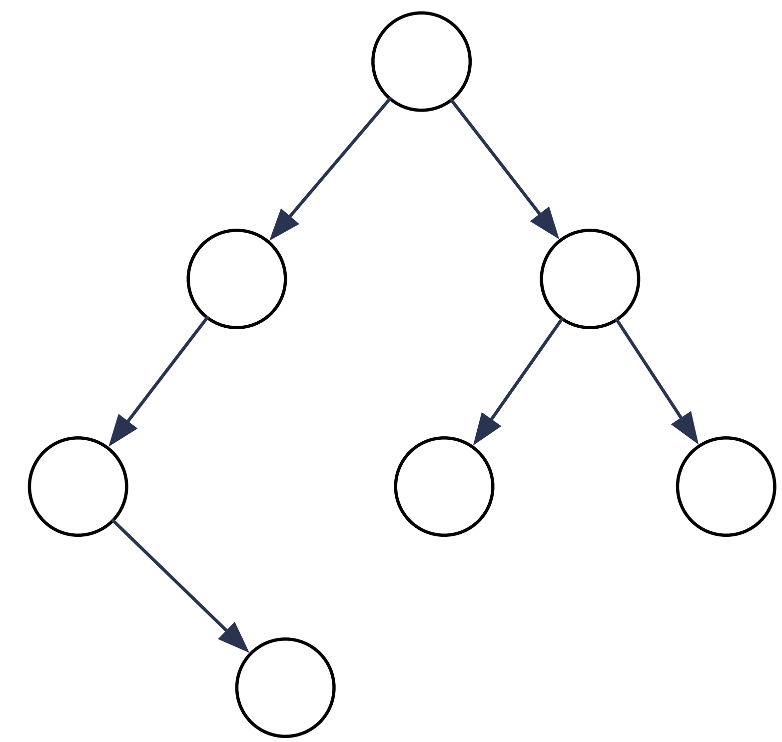
Minimum

- At least *n*-1 comparisons are needed
- Proof: Arrange the results of the comparisons as a tournament graph with nodes being elements
- Tournament graph needs to have a single connected component in order to have a winner



Minimum

- At least *n*-1 comparisons are needed
- Proof: Arrange the results of the comparisons as a tournament graph with nodes being elements
- Tournament graph needs to have a single connected component in order to have a winner
- Component with *x* elements has to have at least *x*-1 edges
  - Proof by induction



Minimum

- This algorithm has *n*-1 comparisons

```
def min(array):
  current_best = array[0]
  for i in range(1, len(array)):
    if array[i] < current_best:
      current_best = array[i]
  return current_best
```

Simultaneous Minimum and Maximum

- Determine minimum and maximum of n elements independently:
    - $2(n-1)$ comparisons

Simultaneous Minimum and Maximum

- Better method?
  - Divide elements into four sets:
    - A: could be either
    - B: could be minimum but not maximum
    - C: could be maximum but not minimum
    - D: could be neither
- In the beginning, every element in A
- At the end, one in B, one in C, everybody else in D

Simultaneous Minimum and Maximum

- Case 1: $x \in A, y \in A$

A: poss. both
B: poss. min
C: poss. max
D: neither

$$x < y \implies x \in B, y \in C$$

$$x > y \implies x \in C, y \in B$$

`Two moves`

Simultaneous Minimum and Maximum

- Case 2: $x \in A, y \in B$

A: poss. both
B: poss. min
C: poss. max
D: neither

$$x < y \implies x \in B, y \in D$$

Two moves, but can always rearrange

$$x > y \implies x \in C, y \in B$$

One move

Simultaneous Minimum and Maximum

- Case 3: $x \in A, y \in C$

$$x < y \implies x \in B, y \in C$$

A: poss. both
B: poss. min
C: poss. max
D: neither

$$x > y \implies x \in C, y \in D$$

Two moves if we are lucky, but we wi

Simultaneous Minimum and Maximum

- Case 4: $x \in A, y \in D$

$$x < y \implies x \in B, y \in D$$

A: poss. both
B: poss. min
C: poss. max
D: neither

$$x > y \implies x \in C, y \in D$$

One move

Simultaneous Minimum and Maximum

- Case 5: $x \in B, y \in B$

A: poss. both
B: poss. min
C: poss. max
D: neither

$$x < y \implies x \in B, y \in D$$

$$x > y \implies x \in D, y \in B$$

One move

Simultaneous Minimum and Maximum

- Case 6: $x \in B, y \in C$

A: poss. both
B: poss. min
C: poss. max
D: neither

$$x < y \implies x \in B, y \in C$$

```
Two moves, at best, but I can cook t
happen
```

$$x > y \implies x \in D, y \in D$$

Simultaneous Minimum and Maximum

- Case 7: $x \in B, y \in D$

A: poss. both
B: poss. min
C: poss. max
D: neither

$$x < y \implies x \in B, y \in D$$

$$x > y \implies x \in D, y \in D$$

Simultaneous Minimum and Maximum

- Case 9: $x \in C, y \in D$

A: poss. both
B: poss. min
C: poss. max
D: neither

$$x < y \implies x \in D, y \in D$$

$$x > y \implies x \in C, y \in D$$

Simultaneous Minimum and Maximum

- Case 10: $x \in D, y \in D$

A: poss. both
B: poss. min
C: poss. max
D: neither

$$x < y \implies x \in D, y \in D$$

$$x > y \implies x \in D, y \in D$$

Simultaneous Minimum and Maximum

- Start out with *n* elements in *A*
  - Best moves involve two elements in *A*
    - Can be done up to *n*/2 times
  - Then need to move *n*-2 elements from *B* and *C* to *D*
    - Can always reshuffle the elements that it needs another *n*-2 comparisons
  - Total is $\lfloor \frac{n}{2} \rfloor + n - 2$ comparisons

Algorithm

- Proof shows how it should be done
  - Make $n/2$ comparisons of virgin elements
  - Then determine the minimum among the losers and the maximum among the winners

Implementation

```
def min_max(lista):
   if len(lista)%2:
      min=lista[0]
      max=lista[0]
      for i in range(1, len(lista)//2):
         if lista[2*i] < lista[2*i+1]:
             loser, winner = lista[2*i], lista[2*i+1]
         else:
            loser, winner = lista[2*i+1], lista[2*i]
         if loser<min:  min = loser
         if winner>max: max = winner
      return min, max
```

$$\frac{n-1}{2} \cdot 3 = \lfloor \frac{n}{2} \rfloor * 3$$

Implementation

```
else:
     if lista[0]<lista[1]:
         min, max = lista[0], lista[1]
     else:
         max, min = lista[1], lista[0]
     for i in range(1,len(lista)//2):
         if lista[2*i]<lista[2*i+1]:
            looser, winner = lista[2*i], lista[2*i+1]
         else:
            looser, winner = lista[2*i+1], lista[2*i]
         if min>looser: min=looser
         if max>winner: max=winner
     return min, max
```

$$1 + (\frac{n-2}{2}) \cdot 3$$

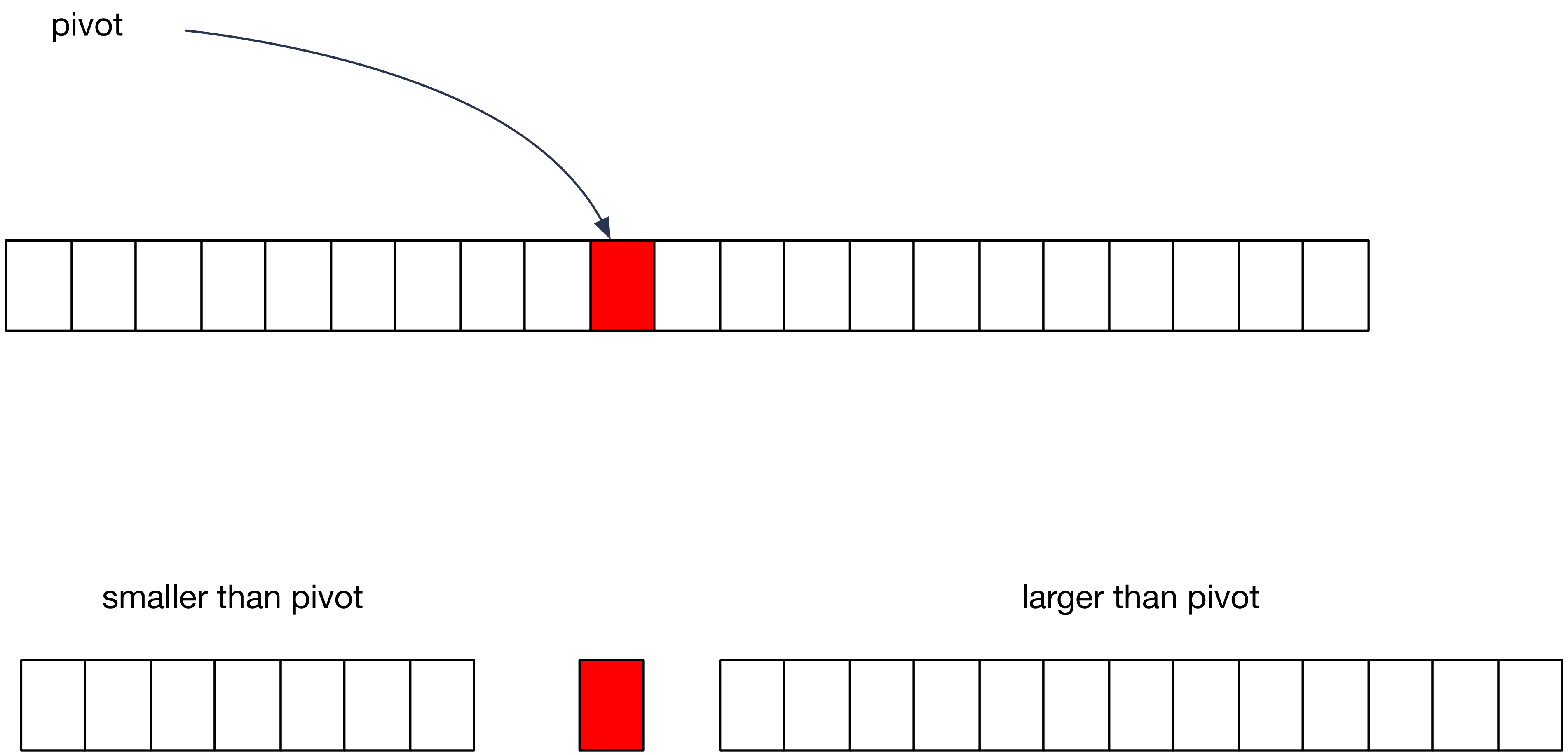
Evaluation

$$\begin{cases} \lfloor \frac{n}{2} \rfloor * 3 & n \text{ odd} \\ 1 + (\frac{n-2}{2}) \cdot 3 & n \text{ even} \end{cases} \quad \leq \quad \lfloor \frac{n}{2} \rfloor * 3$$

Finding the Median

- Return the *i*th largest element in the array
  - Quicksort like algorithm:
    - Select a random pivot
    - Divide the array in two sub-arrays
      - One with elements larger
      - One with elements smaller than pivot

Finding the Median



$n - 1$ comparisons

Finding the Median

- Now process one of the two sub-arrays in order to find the *i*-th largest element
  - On average, the sub-array is of size $\lfloor \frac{n}{2} \rfloor$

Finding the Median

- Now process one of the two sub-arrays in order to find the *i*-th largest element
  - On average, the sub-array is of size $\lfloor \frac{n}{2} \rfloor$
    - Recursion formula is "intuitively"

$$C(n) = n - 1 + C(\lfloor \frac{n}{2} \rfloor$$

Finding the Median

- "Solution"

$$C(n) \leq n + C(n/2) \leq n + \frac{n}{2} + C(n/4) \leq n(1 + \frac{1}{2} + \frac{1}{4} + \ldots) \leq 2n$$

Finding the Median

- Next time:
    - How to make this argument exact