# Medians & Selection

CS16: Introduction to Data Structures & Algorithms

Spring 2019

# Outline

- Medians
- Selection
- Randomized Selection

# Medians

- The median of a collection of numbers
  - is the middle element
  - half of the numbers are smaller and half are larger
- Used to summarize the collection
- The mean or average can also be used…
  - …but averages are sensitive to outliers
- What are the mean & median of
  - `[9,5,4,6,5,7,10000,6,4,8]`
  - mean `1005.4` & median `6`
- Finding the median is easy: sort the list and pick the middle element
  - `O(n log n)`…can we do better?

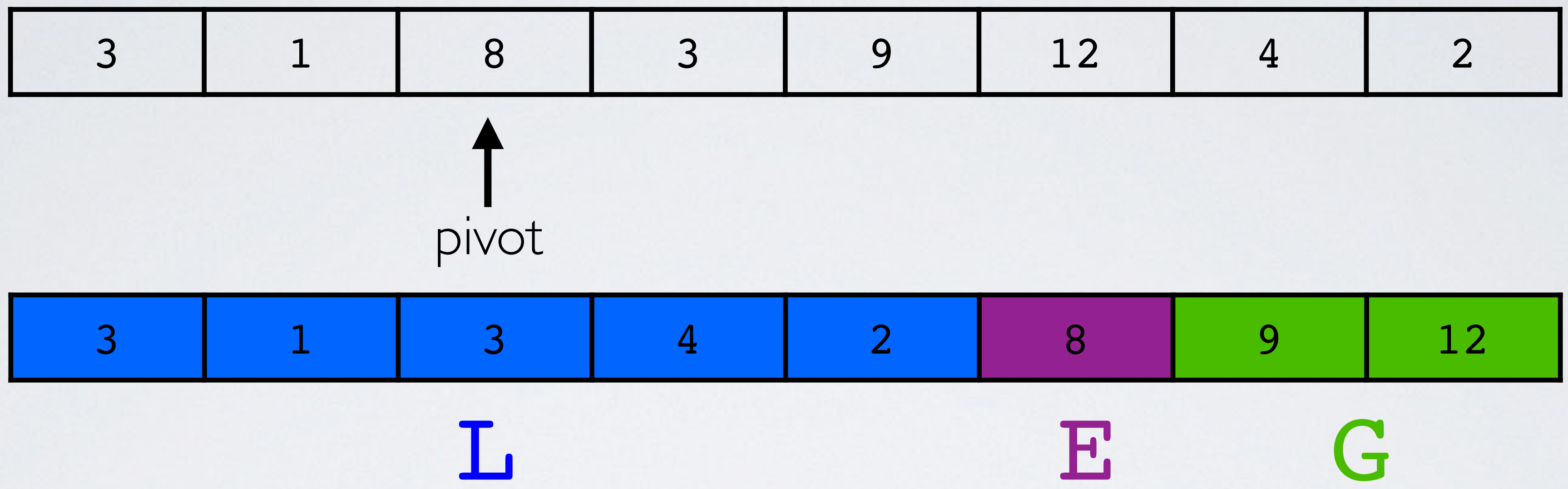
# Selection

- Let's consider a more general problem than median
- The Selection problem
  - given a list **`L`** and an integer **`k`**
  - output the **`k`**th smallest element in the list
- The Median problem can be solved using
  - Selection with **`k = n/2`**

# Quickselect (Hoare's Selection)

- Divide and conquer algorithm
  - divide: pick random element `p` (called pivot) and partition set into
    - **L:** elements less than `p`
    - **E:** elements equal to `p`
    - **G:** elements larger than `p`
  - make recursive call:
    - if `k≤|L|`: call `quickselect(L,k)`
    - if `|L|<k≤|L|+|E|`: return `p`
    - if `k>|L|+|E|`: call `quickselect(G, k−(|L|+|E|))`
  - conquer: return

# Quickselect (Hoare's Selection)




- Suppose `k=4`. Where is the `4`th smallest element?
  - the `4`th smallest element has to be in `L`
  - make recursive call on `L`…but with `k=?`
- Suppose `k=7`. Where is the `7`th smallest element?
  - the `7`th smallest element has to be in `G`
  - make recursive call on `G`…but with `k=?`
- Suppose `k=6`. Where is the `6`th smallest element?
  - the `6`th smallest element has to be in `E`
  - base case

# Quickselect (Hoare's Selection)


- make recursive call:
  - if `k≤|L|`: call `quickselect(L,k)`
  - if `|L|<k≤|L|+|E|`: return `p`
  - if `k>|L|+|E|`: call `quickselect(G, k–(|L|+|E|))`

# Quickselect Pseudo-code

```
quickselect(list, k):
  if list has 1 element return it
  pivot = list[rand(0, list.size)]
  L = []     E = []     G = []
  for x in list:
    if x < pivot: L.append(x)
    if x == pivot: E.append(x)
    if x > pivot: G.append(x)
  if k <= L.size:
    return quickselect(L, k)
  else if k <= (L.size + E.size)
    return pivot
  else
    return quickselect(G, k – (L.size + E.size))
```

# Quickselect

```
quickselect(list, k):
   if list has 1 element return it
   pivot = list[rand(0, list.size)]
   L = []      E = []      G = []
   for x in list:
      if x < pivot: L.append(x)
      if x == pivot: E.append(x)
      if x > pivot: G.append(x)
   if k <= L.size:
      return quickselect(L, k)
   else if k <= (L.size + E.size)
      return pivot
   else
      return quickselect(G, k – (L.size + E.size))
```

**Activity #1+2**

3 min

# Quickselect

```
quickselect(list, k):
   if list has 1 element return it
   pivot = list[rand(0, list.size)]
   L = []     E = []     G = []
   for x in list:
      if x < pivot: L.append(x)
      if x == pivot: E.append(x)
      if x > pivot: G.append(x)
   if k <= L.size:
      return quickselect(L, k)
   else if k <= (L.size + E.size)
      return pivot
   else
      return quickselect(G, k – (L.size + E.size))
```

**Activity #1+2**

3 min

# Quickselect

```
quickselect(list, k):
   if list has 1 element return it
   pivot = list[rand(0, list.size)]
   L = []     E = []     G = []
   for x in list:
      if x < pivot: L.append(x)
      if x == pivot: E.append(x)
      if x > pivot: G.append(x)
   if k <= L.size:
      return quickselect(L, k)
   else if k <= (L.size + E.size)
      return pivot
   else
      return quickselect(G, k – (L.size + E.size))
```

• **Activity #1+2**

2 min

# Quickselect

```
quickselect(list, k):
   if list has 1 element return it
   pivot = list[rand(0, list.size)]
   L = []     E = []     G = []
   for x in list:
      if x < pivot: L.append(x)
      if x == pivot: E.append(x)
      if x > pivot: G.append(x)
   if k <= L.size:
      return quickselect(L, k)
   else if k <= (L.size + E.size)
      return pivot
   else
      return quickselect(G, k – (L.size + E.size))
```

**Activity #1+2**

1 min

# Quickselect

```
quickselect(list, k):
   if list has 1 element return it
   pivot = list[rand(0, list.size)]
   L = []     E = []     G = []
   for x in list:
      if x < pivot: L.append(x)
      if x == pivot: E.append(x)
      if x > pivot: G.append(x)
   if k <= L.size:
      return quickselect(L, k)
   else if k <= (L.size + E.size)
      return pivot
   else
      return quickselect(G, k – (L.size + E.size))
```

- **Activity #1+2**

*0 min*

# Quickselect Analysis

- How fast is Quickselect?
  - kind of like Quicksort except we make only `1` recursive call
- The worst-case is we keep picking min/max element as pivot
  - which leads to worst-case $O(n^2)$ run time
- What about expected run time? (remember Quickselect is randomized)
- We'll assume all elements are distinct
  - if list has more than one copy of pivot,
  - it would shrink the sub-lists and improve runtime

# Quickselect Analysis

- Each pivot has equal probability of being chosen
- Each pivot splits sequence into two
  - one of size `i` and one of size `n-1-i`
  - we recur on only `1` set
- Recurrence relation now has form

$$\mathbb{E}[T(n)] = (n-1) + \frac{1}{n}\sum_{i=1}^{n-1} T(i)$$

- which is `O(n)`

Don't need to know the proof of this.

# Summary

- Quickselect runs in expected `O(n)` time
- Also, if we can solve Selection we can solve Median
  - **`Median(L) = Select(L, n/2)`**
- So we can solve Median in expected `O(n)` time
- What if instead of choosing a random pivot in Quicksort, we used the median?
  - In Quicksort, we could use Quickselect to find the median
  - we would set **`pivot = Quickselect(L, n/2)`**
  - this would avoid the worst-case behavior of Quicksort (i.e., always choosing min/max element)
  - but Quickselect is worst-case $O(n^2)$ so Quicksort would be worst-case $\Omega(n^2)$
  - which is worse than Merge Sort

# Readings

- Dasgupta et al.
  - Section 2.4: analysis of median finding algorithms
- Wocjan's analysis of Selection w/ random pivot
  - http://www.eecs.ucf.edu/courses/cot5405/fall2010/chapter1_2/QuickSelAvgCase.pdf