Divide and Conquer: Quick Select

Madhavan Mukund

https://www.cmi.ac.in/~madhavan

Programming, Data Structures and Algorithms using Python

Week 8

Selection

- Find the k^{th} largest value in a sequence of length n

Selection

- Find the k^{th} largest value in a sequence of length n
- Sort in descending order and look at position k — $O(n \log n)$

Selection

- Find the k^{th} largest value in a sequence of length n
- Sort in descending order and look at position k — $O(n \log n)$
- Can we do better?
  - $k = 1$ — maximum, $O(n)$
  - $k = n$ — minimum, $O(n)$

Selection

- Find the k^{th} largest value in a sequence of length n
- Sort in descending order and look at position k — $O(n \log n)$
- Can we do better?
  - $k = 1$ — maximum, $O(n)$
  - $k = n$ — minimum, $O(n)$
- For any fixed k, k passes, $O(kn)$

Selection

- Find the k^{th} largest value in a sequence of length n
- Sort in descending order and look at position k — $O(n \log n)$
- Can we do better?
  - $k = 1$ — maximum, $O(n)$
  - $k = n$ — minimum, $O(n)$
- For any fixed k, k passes, $O(kn)$
- Median — $k = n/2$
  - If we can find median in $O(n)$, quicksort becomes $O(n \log n)$

Divide and conquer

- Recall partitioning for quicksort
  - Pivot partitions sequence as `lower` and `upper`

Divide and conquer

- Recall partitioning for quicksort
  - Pivot partitions sequence as `lower` and `upper`
- Let `m = len(lower)`. 3 cases:
  - `k <= m` — answer lies in `lower`
  - `k == m+1` — answer is `pivot`
  - `k > m+1` — answer lies in `upper`

Divide and conquer

- Recall partitioning for quicksort
  - Pivot partitions sequence as `lower` and `upper`
- Let `m = len(lower)`. 3 cases:
  - `k <= m` — answer lies in `lower`
  - `k == m+1` — answer is `pivot`
  - `k > m+1` — answer lies in `upper`
- Recursive strategy
  - Case 1: `select(lower,k)`
  - Case 2: `return(pivot)`
  - Case 3: `select(upper,k-(m+1))`

Divide and conquer

- Recall partitioning for quicksort
  - Pivot partitions sequence as `lower` and `upper`
- Let `m = len(lower)`. 3 cases:
  - `k <= m` — answer lies in `lower`
  - `k == m+1` — answer is `pivot`
  - `k > m+1` — answer lies in `upper`
- Recursive strategy
  - Case 1: `select(lower,k)`
  - Case 2: `return(pivot)`
  - Case 3: `select(upper,k-(m+1))`

```
def quickselect(L,l,r,k): # k-th largest in L[l:r]
  if (k < 1) or (k > r-l):
    return(None)

  (pivot,lower,upper) = (L[l],l+1,l+1)
  for i in range(l+1,r):
    if L[i] > pivot:  # Extend upper segment
      upper = upper + 1
    else: # Exchange L[i] with start of upper segment
      (L[i], L[lower]) = (L[lower], L[i])
      (lower,upper) = (lower+1,upper+1)
  (L[l],L[lower-1]) = (L[lower-1],L[l]) # Move pivot
  lower = lower - 1

  # Recursive calls
  lowerlen = lower - l
  if k <= lowerlen:
    return(quickselect(L,l,lower,k))
  elif k == (lowerlen + 1):
    return(L[lower])
  else:
    return(quickselect(L,lower+1,r,k-(lowerlen+1)))
```

Analysis

- Recurrence is similar to quicksort

```
def quickselect(L,l,r,k): # k-th largest in L[l:r]
  if (k < 1) or (k > r-l):
    return(None)

  (pivot,lower,upper) = (L[l],l+1,l+1)
  for i in range(l+1,r):
    if L[i] > pivot:  # Extend upper segment
      upper = upper + 1
    else: # Exchange L[i] with start of upper segment
      (L[i], L[lower]) = (L[lower], L[i])
      (lower,upper) = (lower+1,upper+1)
  (L[l],L[lower-1]) = (L[lower-1],L[l]) # Move pivot
  lower = lower - 1

  # Recursive calls
  lowerlen = lower - l
  if k <= lowerlen:
    return(quickselect(L,l,lower,k))
  elif k == (lowerlen + 1):
    return(L[lower])
  else:
    return(quickselect(L,lower+1,r,k-(lowerlen+1)))
```

Analysis

- Recurrence is similar to quicksort
- $T(1) = 1$
  $T(n) = \max(T(m), T(n-(m+1))) + n$,
  where $m = len(lower)$

```
def quickselect(L,l,r,k): # k-th largest in L[l:r]
  if (k < 1) or (k > r-l):
    return(None)

  (pivot,lower,upper) = (L[l],l+1,l+1)
  for i in range(l+1,r):
    if L[i] > pivot:  # Extend upper segment
      upper = upper + 1
    else: # Exchange L[i] with start of upper segment
      (L[i], L[lower]) = (L[lower], L[i])
      (lower,upper) = (lower+1,upper+1)
  (L[l],L[lower-1]) = (L[lower-1],L[l]) # Move pivot
  lower = lower - 1

  # Recursive calls
  lowerlen = lower - l
  if k <= lowerlen:
    return(quickselect(L,l,lower,k))
  elif k == (lowerlen + 1):
    return(L[lower])
  else:
    return(quickselect(L,lower+1,r,k-(lowerlen+1)))
```

Analysis

- Recurrence is similar to quicksort
- $T(1) = 1$
  $T(n) = \max(T(m), T(n-(m+1))) + n$,
  where $m = len(lower)$
- Worst case: m is always 0 or $n-1$
  - $T(n) = T(n-1) + n$
  - $T(n)$ is $O(n^2)$

```python
def quickselect(L,l,r,k): # k-th largest in L[l:r]
  if (k < 1) or (k > r-l):
    return(None)

  (pivot,lower,upper) = (L[l],l+1,l+1)
  for i in range(l+1,r):
    if L[i] > pivot:  # Extend upper segment
      upper = upper + 1
    else: # Exchange L[i] with start of upper segment
      (L[i], L[lower]) = (L[lower], L[i])
      (lower,upper) = (lower+1,upper+1)
  (L[l],L[lower-1]) = (L[lower-1],L[l]) # Move pivot
  lower = lower - 1

  # Recursive calls
  lowerlen = lower - l
  if k <= lowerlen:
    return(quickselect(L,l,lower,k))
  elif k == (lowerlen + 1):
    return(L[lower])
  else:
    return(quickselect(L,lower+1,r,k-(lowerlen+1)))
```

Analysis

- Recurrence is similar to quicksort
- $T(1) = 1$
  $T(n) = \max(T(m), T(n-(m+1))) + n$,
  where $m = len(lower)$
- Worst case: m is always 0 or $n-1$
  - $T(n) = T(n-1) + n$
  - $T(n)$ is $O(n^2)$
- Recall: if pivot is within a fixed fraction, quicksort is $O(n \log n)$
  - E.g., pivot in middle third of values
  - $T(n) = T(n/3) + T(2n/3) + n$

```
def quickselect(L,l,r,k): # k-th largest in L[l:r]
  if (k < 1) or (k > r-l):
    return(None)

  (pivot,lower,upper) = (L[l],l+1,l+1)
  for i in range(l+1,r):
    if L[i] > pivot:  # Extend upper segment
      upper = upper + 1
    else: # Exchange L[i] with start of upper segment
      (L[i], L[lower]) = (L[lower], L[i])
      (lower,upper) = (lower+1,upper+1)
  (L[l],L[lower-1]) = (L[lower-1],L[l]) # Move pivot
  lower = lower - 1

  # Recursive calls
  lowerlen = lower - l
  if k <= lowerlen:
    return(quickselect(L,l,lower,k))
  elif k == (lowerlen + 1):
    return(L[lower])
  else:
    return(quickselect(L,lower+1,r,k-(lowerlen+1)))
```

Analysis

- Recurrence is similar to quicksort
- $T(1) = 1$
  $T(n) = \max(T(m), T(n-(m+1))) + n$,
  where $m = len(lower)$
- Worst case: m is always 0 or $n-1$
  - $T(n) = T(n-1) + n$
  - $T(n)$ is $O(n^2)$
- Recall: if pivot is within a fixed fraction, quicksort is $O(n \log n)$
  - E.g., pivot in middle third of values
  - $T(n) = T(n/3) + T(2n/3) + n$
- Can we find a good pivot quickly?

```
def quickselect(L,l,r,k): # k-th largest in L[l:r]
  if (k < 1) or (k > r-l):
    return(None)

  (pivot,lower,upper) = (L[l],l+1,l+1)
  for i in range(l+1,r):
    if L[i] > pivot:  # Extend upper segment
      upper = upper + 1
    else: # Exchange L[i] with start of upper segment
      (L[i], L[lower]) = (L[lower], L[i])
      (lower,upper) = (lower+1,upper+1)
  (L[l],L[lower-1]) = (L[lower-1],L[l]) # Move pivot
  lower = lower - 1

  # Recursive calls
  lowerlen = lower - l
  if k <= lowerlen:
    return(quickselect(L,l,lower,k))
  elif k == (lowerlen + 1):
    return(L[lower])
  else:
    return(quickselect(L,lower+1,r,k-(lowerlen+1)))
```

Median of medians

- Divide L into blocks of 5

Median of medians

- Divide L into blocks of 5
- Find the median of each block (brute force)

Median of medians

- Divide L into blocks of 5
- Find the median of each block (brute force)
- Let M be the list of block medians

Median of medians

- Divide L into blocks of 5
- Find the median of each block (brute force)
- Let M be the list of block medians
- Recursively apply the process to M

Median of medians

- Divide L into blocks of 5
- Find the median of each block (brute force)
- Let M be the list of block medians
- Recursively apply the process to M

```
def MoM(L): # Median of medians

  if len(L) <= 5:
    L.sort()
    return(L[len(L)//2])

  # Construct list of block medians
  M = []

  for i in range(0,len(L),5):
    X = L[i:i+5]
    X.sort()
    M.append(X[len(X)//2])

  return(MoM(M))
```

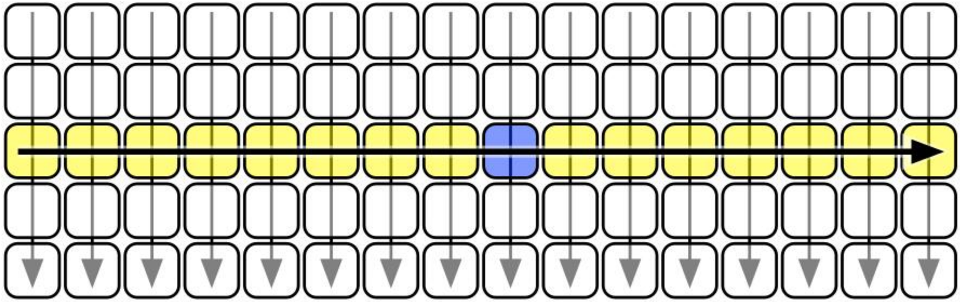
Median of medians

- Divide L into blocks of 5
- Find the median of each block (brute force)
- Let M be the list of block medians
- Recursively apply the process to M
- What can we guarantee about `MoM(L)`?

```
def MoM(L): # Median of medians

  if len(L) <= 5:
    L.sort()
    return(L[len(L)//2])

  # Construct list of block medians
  M = []

  for i in range(0,len(L),5):
    X = L[i:i+5]
    X.sort()
    M.append(X[len(X)//2])

  return(MoM(M))
```

Median of medians

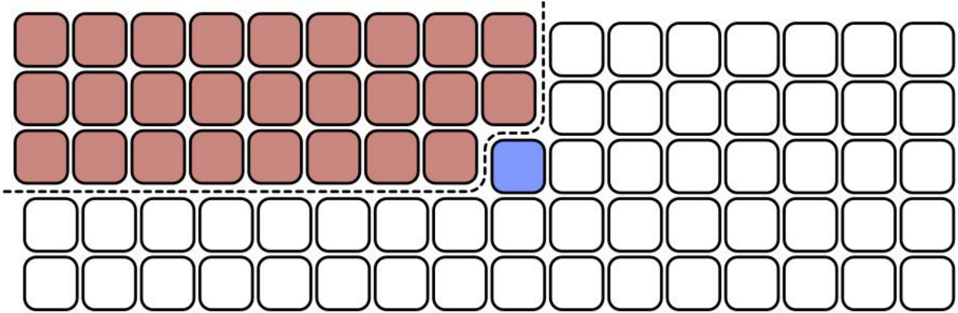
- We can visualize the blocks as follows



- Each block of 5 is arranged in ascending order, top to bottom
- Block medians are the middle row

Median of medians

- We can visualize the blocks as follows
- Each block of 5 is arranged in ascending order, top to bottom
- Block medians are the middle row
- The median of block medians lies between $3len(L)/10$ and $7len(L)/10$



Analysis

- Use median of block medians to locate pivot for `quickselect`

Analysis

- Use median of block medians to locate pivot for `quickselect`

```
def fastselect(L,l,r,k): # k-th largest in L[l:r]
  if (k < 1) or (k > r-l):
    return(None)

  # Find MoM pivot and move to L[l]
  pivot = MoM(L[l:r])
  pivotpos = min([i for i in range(l,r) if L[i] == pivo
  (L[l],L[pivotpos]) = (L[pivotpos],L[l])

  # Partition as before
  (pivot,lower,upper) = (L[l],l+1,l+1)
  for i in range(l+1,r):
    ...

  # Recursive calls
  lowerlen = lower - l
  if k <= lowerlen:
    return(fastselect(L,l,lower,k))
  elif k == (lowerlen + 1):
    return(L[lower])
  else:
    return(fastselect(L,lower+1,r,k-(lowerlen+1)))
```

Analysis

- Use median of block medians to locate pivot for `quickselect`
- `MoM` is $O(n)$

$$T(1) = 1$$
$$T(n) = T(n/5) + n,$$

```
def fastselect(L,l,r,k): # k-th largest in L[l:r]
  if (k < 1) or (k > r-l):
    return(None)

  # Find MoM pivot and move to L[l]
  pivot = MoM(L[l:r])
  pivotpos = min([i for i in range(l,r) if L[i] == pivo
  (L[l],L[pivotpos]) = (L[pivotpos],L[l])

  # Partition as before
  (pivot,lower,upper) = (L[l],l+1,l+1)
  for i in range(l+1,r):
    ...

  # Recursive calls
  lowerlen = lower - l
  if k <= lowerlen:
    return(fastselect(L,l,lower,k))
  elif k == (lowerlen + 1):
    return(L[lower])
  else:
    return(fastselect(L,lower+1,r,k-(lowerlen+1)))
```

Analysis

- Use median of block medians to locate pivot for `quickselect`

- `MoM` is $O(n)$

$$T(1) = 1$$
$$T(n) = T(n/5) + n,$$

- Recurrence for `fastselect` is now

$$T(1) = 1$$
$$T(n) = \max(T(3m/10), T(7m/10) + n,$$
where $m = len(lower)$

```
def fastselect(L,l,r,k): # k-th largest in L[l:r]
  if (k < 1) or (k > r-l):
    return(None)

  # Find MoM pivot and move to L[l]
  pivot = MoM(L[l:r])
  pivotpos = min([i for i in range(l,r) if L[i] == pivo
  (L[l],L[pivotpos]) = (L[pivotpos],L[l])

  # Partition as before
  (pivot,lower,upper) = (L[l],l+1,l+1)
  for i in range(l+1,r):
    ...

  # Recursive calls
  lowerlen = lower - l
  if k <= lowerlen:
    return(fastselect(L,l,lower,k))
  elif k == (lowerlen + 1):
    return(L[lower])
  else:
    return(fastselect(L,lower+1,r,k-(lowerlen+1)))
```

Analysis

- Use median of block medians to locate pivot for `quickselect`
- `MoM` is $O(n)$

  $$T(1) = 1$$
  $$T(n) = T(n/5) + n,$$

- Recurrence for `fastselect` is now

  $$T(1) = 1$$
  $$T(n) = \max(T(3m/10), T(7m/10) + n,$$
  where $m = len(lower)$

- $T(n)$ is $O(n)$

```
def fastselect(L,l,r,k): # k-th largest in L[l:r]
  if (k < 1) or (k > r-l):
    return(None)

  # Find MoM pivot and move to L[l]
  pivot = MoM(L[l:r])
  pivotpos = min([i for i in range(l,r) if L[i] == pivo
  (L[l],L[pivotpos]) = (L[pivotpos],L[l])

  # Partition as before
  (pivot,lower,upper) = (L[l],l+1,l+1)
  for i in range(l+1,r):
    ...

  # Recursive calls
  lowerlen = lower - l
  if k <= lowerlen:
    return(fastselect(L,l,lower,k))
  elif k == (lowerlen + 1):
    return(L[lower])
  else:
    return(fastselect(L,lower+1,r,k-(lowerlen+1)))
```

Analysis

- Use median of block medians to locate pivot for `quickselect`
- `MoM` is $O(n)$

  $$T(1) = 1$$
  $$T(n) = T(n/5) + n,$$

- Recurrence for `fastselect` is now

  $$T(1) = 1$$
  $$T(n) = \max(T(3m/10), T(7m/10) + n,$$
  where $m = len(lower)$

- $T(n)$ is $O(n)$
- Can also use `MoM` to make quicksort $O(n \log n)$

```python
def fastselect(L,l,r,k): # k-th largest in L[l:r]
  if (k < 1) or (k > r-l):
    return(None)

  # Find MoM pivot and move to L[l]
  pivot = MoM(L[l:r])
  pivotpos = min([i for i in range(l,r) if L[i] == pivo
  (L[l],L[pivotpos]) = (L[pivotpos],L[l])

  # Partition as before
  (pivot,lower,upper) = (L[l],l+1,l+1)
  for i in range(l+1,r):
    ...

  # Recursive calls
  lowerlen = lower - l
  if k <= lowerlen:
    return(fastselect(L,l,lower,k))
  elif k == (lowerlen + 1):
    return(L[lower])
  else:
    return(fastselect(L,lower+1,r,k-(lowerlen+1)))
```

Summary

- Median of block medians helps find a good pivot in $O(n)$

Summary

- Median of block medians helps find a good pivot in $O(n)$
- Selection becomes $O(n)$, quicksort becomes $O(n \log n)$

Summary

- Median of block medians helps find a good pivot in $O(n)$
- Selection becomes $O(n)$, quicksort becomes $O(n \log n)$
- Notice that `fastselect` with `k = len(L)/2` finds median in time $O(n)$

Summary

- Median of block medians helps find a good pivot in $O(n)$
- Selection becomes $O(n)$, quicksort becomes $O(n \log n)$
- Notice that `fastselect` with `k = len(L)/2` finds median in time $O(n)$

Historical note

Summary

- Median of block medians helps find a good pivot in $O(n)$
- Selection becomes $O(n)$, quicksort becomes $O(n \log n)$
- Notice that `fastselect` with `k = len(L)/2` finds median in time $O(n)$

Historical note

- C.A.R. Hoare described `quickselect` in the same paper that introduced `quicksort`, 1962

Summary

- Median of block medians helps find a good pivot in $O(n)$
- Selection becomes $O(n)$, quicksort becomes $O(n \log n)$
- Notice that `fastselect` with `k = len(L)/2` finds median in time $O(n)$

Historical note

- C.A.R. Hoare described `quickselect` in the same paper that introduced `quicksort`, 1962
- The median of medians algorithm is due to Manuel Blum, Robert Floyd, Vaughn Pratt, Ron Rivest and Robert Tarjan, 1973

Acknowledgment
Illustrations from *Algorithms* by Jeff Erickson, `https://jeffe.cs.illinois.edu/teaching/algorithms/`