CMPSC 465: LECTURE VI

QuickSort
Selection

Ke Chen

September 10, 2025

QuickSort

Input: 8,1,9,2,8,4,6,5

Idea: divide and conquer

		<	pivot	>
1	Split input by a pivot	1,2,4	5	8,9,6,8
2	Sort each part	1,2,4	5	6,8,8,9
3	TADA!	1,2,4,5,6,8,8,9		

QuickSort(A, st, ed)

```
if st < ed then
    p = Partition(A, st, ed)
    QuickSort(A, st, p − 1)
    QuickSort(A, p + 1, ed)
```

How to do Partition in place, namely, with $O(1)$ extra space?

Partition around pivot

Idea: check elements one by one against the pivot and maintain a $<$ pivot region and a $\geq$ pivot region

$$8 \quad 1 \quad 9 \quad 2 \quad 8 \quad 4 \quad 6 \quad 5$$

Partition(A, st, ed)

```
pivot = A[ed]
bd = st − 1
for cur = st to ed − 1 do
    if A[cur] < pivot then
        bd = bd + 1
        swap A[bd] with A[cur]
swap A[bd + 1] with A[ed]
return bd+1
```

Partition around pivot

Idea: check elements one by one against the pivot and maintain a $<$ pivot region and a $\geq$ pivot region

last element in the $<$ pivot region

8	1	9	2	8	4	6	5
st							ed

Partition(A, st, ed)

- $pivot = A[ed]$
- $bd = st - 1$
- **for** $cur = st$ **to** $ed - 1$ **do**
  - **if** $A[cur] < pivot$ **then**
    - $bd = bd + 1$
    - swap $A[bd]$ with $A[cur]$
- swap $A[bd + 1]$ with $A[ed]$
- **return** *bd+1*

Partition around pivot

Idea: check elements one by one against the pivot and maintain a $<$ pivot region and a $\geq$ pivot region

$\downarrow$

8	1	9	2	8	4	6	5
st							ed

Partition(A, st, ed)

```
pivot = A[ed]
bd = st − 1
for cur = st to ed − 1 do
    if A[cur] < pivot then
        bd = bd + 1
        swap A[bd] with A[cur]
swap A[bd + 1] with A[ed]
return bd+1
```

Partition around pivot

Idea: check elements one by one against the pivot and maintain a $<$ pivot region and a $\geq$ pivot region

↓							
8	1	9	2	8	4	6	5
st							ed

Partition(A, st, ed)

- $pivot = A[ed]$
- $bd = st - 1$
- **for** $cur = st$ **to** $ed - 1$ **do**
  - **if** $A[cur] < pivot$ **then**
    - $bd = bd + 1$
    - swap $A[bd]$ with $A[cur]$
- swap $A[bd + 1]$ with $A[ed]$
- **return** *bd+1*

Partition around pivot

Idea: check elements one by one against the pivot and maintain a $<$ pivot region and a $\geq$ pivot region

$\downarrow$

8 1 9 2 8 4 6 5

st ed

Partition(A, st, ed)

$pivot = A[ed]$
$bd = st - 1$
for $cur = st$ **to** $ed - 1$ **do**
  if $A[cur] < pivot$ **then**
    $bd = bd + 1$
    swap $A[bd]$ with $A[cur]$
swap $A[bd+1]$ with $A[ed]$
return *bd+1*

Partition around pivot

Idea: check elements one by one against the pivot and maintain a $<$ pivot region and a $\geq$ pivot region

$$\begin{array}{cccccccc} \downarrow & & & & & & & \\ 1 & 8 & 9 & 2 & 8 & 4 & 6 & 5 \\ st & & & & & & & ed \end{array}$$

Partition(A, st, ed)

- $pivot = A[ed]$
- $bd = st - 1$
- **for** $cur = st$ **to** $ed - 1$ **do**
  - **if** $A[cur] < pivot$ **then**
    - $bd = bd + 1$
    - swap $A[bd]$ with $A[cur]$
- swap $A[bd + 1]$ with $A[ed]$
- **return** *bd+1*

Partition around pivot

Idea: check elements one by one against the pivot and maintain a $<$ pivot region and a $\geq$ pivot region

$\downarrow$							
1	8	9	2	8	4	6	5
st							ed

Partition(A, st, ed)

$pivot = A[ed]$
$bd = st - 1$
for $cur = st$ **to** $ed - 1$ **do**
    if $A[cur] < pivot$ **then**
        $bd = bd + 1$
        swap $A[bd]$ with $A[cur]$
swap $A[bd + 1]$ with $A[ed]$
return *bd+1*

Partition around pivot

Idea: check elements one by one against the pivot and maintain a $<$ pivot region and a $\geq$ pivot region

$$\downarrow$$

1	8	9	2	8	4	6	5
st							ed

Partition(A, st, ed)

- $pivot = A[ed]$
- $bd = st - 1$
- **for** $cur = st$ **to** $ed - 1$ **do**
  - **if** $A[cur] < pivot$ **then**
    - $bd = bd + 1$
    - swap $A[bd]$ with $A[cur]$
- swap $A[bd + 1]$ with $A[ed]$
- **return** *bd+1*

Partition around pivot

Idea: check elements one by one against the pivot and maintain a $<$ pivot region and a $\geq$ pivot region

1	8	9	2	8	4	6	5
st							ed

Partition(A, st, ed)

- $pivot = A[ed]$
- $bd = st - 1$
- **for** $cur = st$ **to** $ed - 1$ **do**
  - **if** $A[cur] < pivot$ **then**
    - $bd = bd + 1$
    - swap $A[bd]$ with $A[cur]$
- swap $A[bd + 1]$ with $A[ed]$
- **return** *bd+1*

Partition around pivot

Idea: check elements one by one against the pivot and maintain a $<$ pivot region and a $\geq$ pivot region

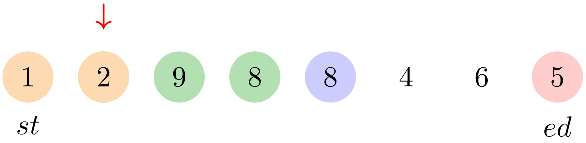
$$\begin{array}{cccccccc} & \downarrow & & & & & & \\ 1 & 2 & 9 & 8 & 8 & 4 & 6 & 5 \\ st & & & & & & & ed \end{array}$$

Partition(A, st, ed)

- $pivot = A[ed]$
- $bd = st - 1$
- **for** $cur = st$ **to** $ed - 1$ **do**
  - **if** $A[cur] < pivot$ **then**
    - $bd = bd + 1$
    - swap $A[bd]$ with $A[cur]$
- swap $A[bd + 1]$ with $A[ed]$
- **return** *bd+1*

Partition around pivot

Idea: check elements one by one against the pivot and maintain a $<$ pivot region and a $\geq$ pivot region



Partition(A, st, ed)

- $pivot = A[ed]$
- $bd = st - 1$
- **for** $cur = st$ **to** $ed - 1$ **do**
  - **if** $A[cur] < pivot$ **then**
    - $bd = bd + 1$
    - swap $A[bd]$ with $A[cur]$
- swap $A[bd + 1]$ with $A[ed]$
- **return** *bd+1*

Partition around pivot

Idea: check elements one by one against the pivot and maintain a $<$ pivot region and a $\geq$ pivot region

$$\downarrow$$

$$\begin{array}{cccccccc} 1 & 2 & 9 & 8 & 8 & 4 & 6 & 5 \\ st & & & & & & & ed \end{array}$$

Partition(A, st, ed)

- $pivot = A[ed]$
- $bd = st - 1$
- **for** $cur = st$ **to** $ed - 1$ **do**
  - **if** $A[cur] < pivot$ **then**
    - $bd = bd + 1$
    - swap $A[bd]$ with $A[cur]$
- swap $A[bd + 1]$ with $A[ed]$
- **return** *bd+1*

Partition around pivot

Idea: check elements one by one against the pivot and maintain a $<$ pivot region and a $\geq$ pivot region

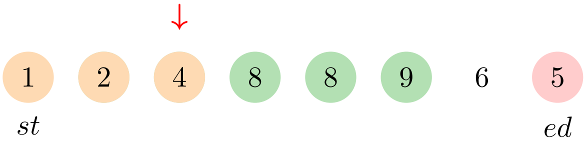
1	2	9	8	8	4	6	5
st							ed

Partition(A, st, ed)

$pivot = A[ed]$
$bd = st - 1$
for $cur = st$ **to** $ed - 1$ **do**
  if $A[cur] < pivot$ **then**
    $bd = bd + 1$
    swap $A[bd]$ with $A[cur]$
swap $A[bd + 1]$ with $A[ed]$
return bd+1

Partition around pivot

Idea: check elements one by one against the pivot and maintain a $<$ pivot region and a $\geq$ pivot region

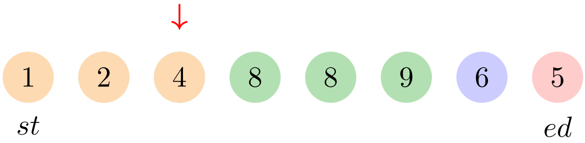


Partition(A, st, ed)

- $pivot = A[ed]$
- $bd = st - 1$
- **for** $cur = st$ **to** $ed - 1$ **do**
  - **if** $A[cur] < pivot$ **then**
    - $bd = bd + 1$
    - swap $A[bd]$ with $A[cur]$
- swap $A[bd + 1]$ with $A[ed]$
- **return** *bd+1*

Partition around pivot

Idea: check elements one by one against the pivot and maintain a $<$ pivot region and a $\geq$ pivot region

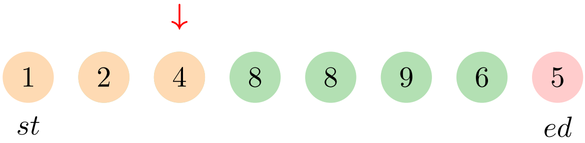


Partition(A, st, ed)

$pivot = A[ed]$
$bd = st - 1$
for $cur = st$ **to** $ed - 1$ **do**
    if $A[cur] < pivot$ **then**
        $bd = bd + 1$
        swap $A[bd]$ with $A[cur]$
swap $A[bd + 1]$ with $A[ed]$
return *bd+1*

Partition around pivot

Idea: check elements one by one against the pivot and maintain a $<$ pivot region and a $\geq$ pivot region

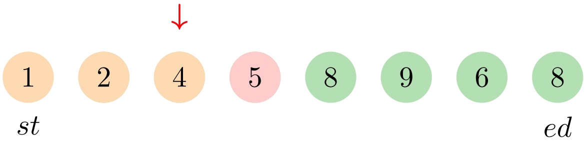


Partition(A, st, ed)

$pivot = A[ed]$
$bd = st - 1$
for $cur = st$ **to** $ed - 1$ **do**
  if $A[cur] < pivot$ **then**
    $bd = bd + 1$
    swap $A[bd]$ with $A[cur]$
swap $A[bd+1]$ with $A[ed]$
return *bd+1*

Partition around pivot

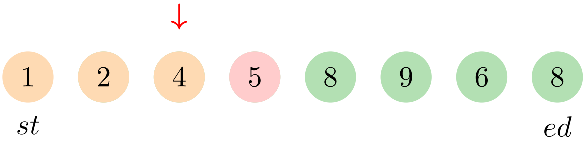
Idea: check elements one by one against the pivot and maintain a $<$ pivot region and a $\geq$ pivot region




```
Partition(A, st, ed)
    pivot = A[ed]
    bd = st − 1
    for cur = st to ed − 1 do
        if A[cur] < pivot then
            bd = bd + 1
            swap A[bd] with A[cur]
    swap A[bd + 1] with A[ed]
    return bd+1
```

Partition around pivot

Idea: check elements one by one against the pivot and maintain a $<$ pivot region and a $\geq$ pivot region



Partition(A, st, ed)

- $pivot = A[ed]$
- $bd = st - 1$
- **for** $cur = st$ **to** $ed - 1$ **do**
  - **if** $A[cur] < pivot$ **then**
    - $bd = bd + 1$
    - swap $A[bd]$ with $A[cur]$
- swap $A[bd + 1]$ with $A[ed]$
- **return** *bd+1*

Time complexity? $\Theta(n)$

Space complexity? $\Theta(1)$

QuickSort

```
QuickSort(A, st, ed)
  if st < ed then
    p = Partition(A, st, ed)
    QuickSort(A, st, p − 1)
    QuickSort(A, p + 1, ed)
```

Time complexity?

- Worst-case is when each partition results in sizes (0, 1, n-1).
  $T(n) = \Theta(n) + T(n-1)$

QuickSort

QuickSort(A, st, ed)

if $st < ed$ **then**

  $p =$ Partition(A, st, ed)

  QuickSort(A, st, $p - 1$)

  QuickSort(A, $p + 1$, ed)

Time complexity?

- Worst-case is when each partition results in sizes (0, 1, n-1).
  $T(n) = \Theta(n) + T(n-1)$ $\rightsquigarrow$ $T(n) = O(n^2)$.

QuickSort

QuickSort(A, st, ed)

if $st < ed$ **then**
    $p =$ Partition(A, st, ed)
    QuickSort(A, st, $p - 1$)
    QuickSort(A, $p + 1$, ed)

Time complexity?

- Worst-case is when each partition results in sizes (0, 1, n-1).
  $T(n) = \Theta(n) + T(n-1)$ $\rightsquigarrow$ $T(n) = O(n^2)$.
- Can you find an input that achieves this worst-case performance?

QuickSort

QuickSort(A, st, ed)

if $st < ed$ **then**
    $p =$ Partition(A, st, ed)
    QuickSort(A, st, $p - 1$)
    QuickSort(A, $p + 1$, ed)

Time complexity?

- Worst-case is when each partition results in sizes (0, 1, n-1).
  $T(n) = \Theta(n) + T(n-1)$ $\rightsquigarrow$ $T(n) = O(n^2)$.
- Can you find an input that achieves this worst-case performance?
- Why is it called QuickSort then?

QuickSort

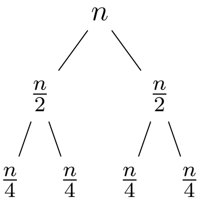
Why do we use QuickSort if it is not asymptotically optimal?

- Its average-case time complexity is $O(n \log n)$: more precisely, the expected running time on a random permutation of n numbers is $O(n \log n)$.

QuickSort

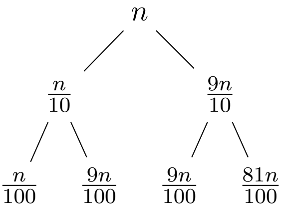
Why do we use QuickSort if it is not asymptotically optimal?

- Its average-case time complexity is $O(n \log n)$: more precisely, the expected running time on a random permutation of n numbers is $O(n \log n)$.
- Intuitively, even "quite unbalanced" splits, such as 10% – 90% at each level, still yield a $O(n \log n)$ running time.



Perfect split, depth $= \log n$

$T(n) = \Theta(n) + 2T(n/2)$



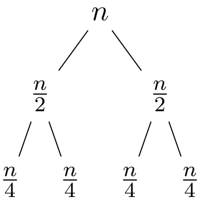
Good split, depth $= \log_{\frac{10}{9}} n \approx 7 \log n$

$T(n) = \Theta(n) + T(n/10) + T(9n/10)$

QuickSort

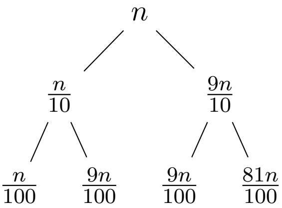
Why do we use QuickSort if it is not asymptotically optimal?

- Its average-case time complexity is $O(n \log n)$: more precisely, the expected running time on a random permutation of n numbers is $O(n \log n)$.
- Intuitively, even "quite unbalanced" splits, such as $10\% - 90\%$ at each level, still yield a $O(n \log n)$ running time.



Perfect split, depth $= \log n$

$T(n) = \Theta(n) + 2T(n/2)$



Good split, depth $= \log_{\frac{10}{9}} n \approx 7 \log n$

$T(n) = \Theta(n) + T(n/10) + T(9n/10)$

- Partition is efficient: small hidden constants and runs in place.

Selection problem

- Recall the selection problem: Given a list of n numbers and a target k, select the k-th smallest number.

Selection problem

- Recall the selection problem: Given a list of n numbers and a target k, select the k-th smallest number.
- We have argued a $\Omega(n)$ lower bound on this task.

Selection problem

- Recall the selection problem: Given a list of n numbers and a target k, select the k-th smallest number.
- We have argued a $\Omega(n)$ lower bound on this task.
- Is there an algorithm that can achieve a matching running time $O(n)$?

Randomized selection

Input: 8, 1, 9, 2, 8, 4, 6, 5 $\quad k = 5$

Idea: divide-and-conquer

1. Split input by a random pivot $\quad$ 1,2,4 | 5 | 8, 9, 6 , 8 $\quad k = 5$

Randomized selection

Input: 8, 1, 9, 2, 8, 4, 6, 5 $\quad k = 5$

Idea: divide-and-conquer

1. Split input by a random pivot $\qquad$ 1,2,4 | 5 | 8, 9, 6 , 8 $\quad k = 5$
2. Decide which side to recurse on $\qquad$ 1,2,4 | 5 | 8, 9, 6 , 8 $\quad k = 1$

Randomized selection

Input: 8, 1, 9, 2, 8, 4, 6, 5 $\quad k = 5$

Idea: divide-and-conquer

1. Split input by a random pivot $\quad$ 1,2,4 | 5 | 8, 9, 6 , 8 $\quad k = 5$
2. Decide which side to recurse on $\quad$ 1,2,4 | 5 | 8, 9, 6 , 8 $\quad k = 1$
3. Done! $\quad$ 1,2,4 | 5 | 8, 9, 6 , 8 $\quad k = 1$

Randomized selection

Input: 8, 1, 9, 2, 8, 4, 6, 5 $\quad k = 5$

Idea: divide-and-conquer

1	Split input by a **random pivot**	1,2,4 | 5 | 8, 9, 6 , 8	$k = 5$
2	Decide which side to recurse on	1,2,4 | 5 | 8, 9, 6 , 8	$k = 1$
3	Done!	1,2,4 | 5 | 8, 9, 6 , 8	$k = 1$

RandomizedSelect(A, st, ed, k)

if $st == ed$ **then return** $A[st]$
`// p is the index of the random pivot`
$p =$ Partition(A, st, ed)
$rank = p - st + 1$ `// rank of the pivot`
if $k == rank$ **then return** $A[p]$
else if $k < rank$ **then**
  return RandomizedSelect(A, st, $p - 1$, k)
else return RandomizedSelect(A, $p + 1$, ed, $k - rank$)

Randomized selection

Time complexity?

- Worst-case: $T(n) = \Theta(n) + T(\text{?})$

Randomized selection

Time complexity?

- Worst-case: $T(n) = \Theta(n) + T($n-1$)$

Randomized selection

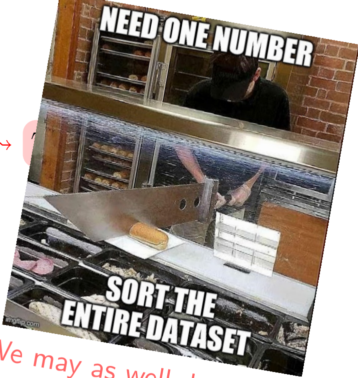
Time complexity?

- Worst-case: $T(n) = \Theta(n) + T(\text{n-1}) \rightsquigarrow T(n) = O(n^2)$

Randomized selection

Time complexity?

- Worst-case: $T(n) = \Theta(n) + T(\text{n-1}) \rightsquigarrow$



We may as well do sorting!

Randomized selection

Time complexity?

- Worst-case: $T(n) = \Theta(n) + T(\text{n-1}) \rightsquigarrow T(n) = O(n^2)$
- Can you find an input that force the worst-case performance?

Randomized selection

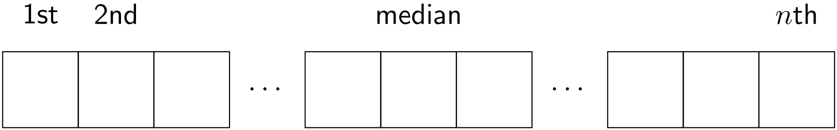
Time complexity?

- Worst-case: $T(n) = \Theta(n) + T(\text{n-1}) \rightsquigarrow T(n) = O(n^2)$
- Can you find an input that force the worst-case performance?
- Average-case: what's the expected input size for the recursive call?

Randomized selection

Time complexity?

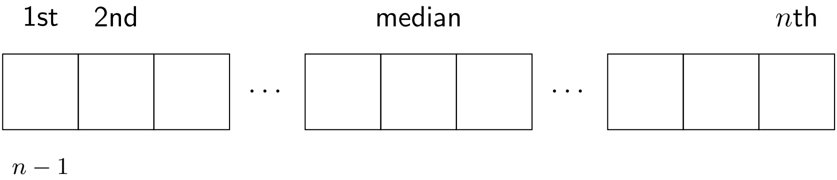
- Worst-case: $T(n) = \Theta(n) + T(\text{n-1}) \rightsquigarrow T(n) = O(n^2)$
- Can you find an input that force the worst-case performance?
- Average-case: what's the expected input size for the recursive call?



Randomized selection

Time complexity?

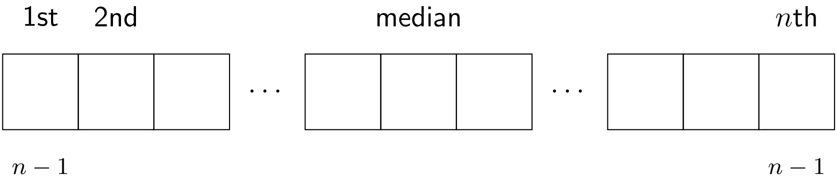
- Worst-case: $T(n) = \Theta(n) + T(\text{n-1}) \rightsquigarrow T(n) = O(n^2)$
- Can you find an input that force the worst-case performance?
- Average-case: what's the expected input size for the recursive call?



Randomized selection

Time complexity?

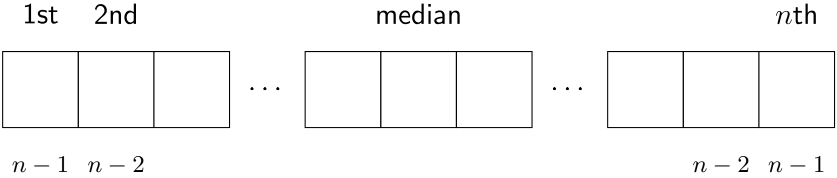
- Worst-case: $T(n) = \Theta(n) + T(\text{n-1}) \rightsquigarrow T(n) = O(n^2)$
- Can you find an input that force the worst-case performance?
- Average-case: what's the expected input size for the recursive call?



Randomized selection

Time complexity?

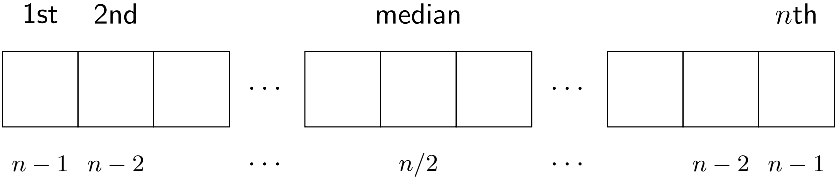
- Worst-case: $T(n) = \Theta(n) + T($n-1$) \rightsquigarrow T(n) = O(n^2)$
- Can you find an input that force the worst-case performance?
- Average-case: what's the expected input size for the recursive call?



Randomized selection

Time complexity?

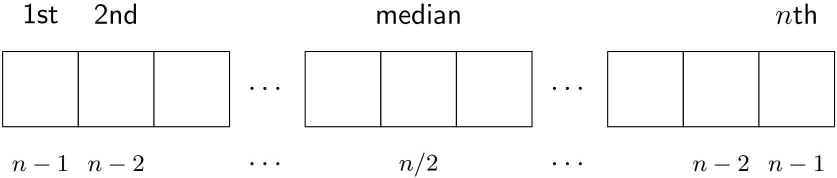
- Worst-case: $T(n) = \Theta(n) + T(\text{n-1}) \rightsquigarrow T(n) = O(n^2)$
- Can you find an input that force the worst-case performance?
- Average-case: what's the expected input size for the recursive call?



Randomized selection

Time complexity?

- Worst-case: $T(n) = \Theta(n) + T($n-1$)$ $\rightsquigarrow$ $T(n) = O(n^2)$
- Can you find an input that force the worst-case performance?
- Average-case: what's the expected input size for the recursive call?



Each number has an equal chance of $1/n$ to be the pivot, so the expected input size is:
$((n-1)+(n-2)+\cdots+n/2+\cdots+(n-2)+(n-1))/n \approx$ 3n/4.

Randomized selection

Time complexity?

- Worst-case: $T(n) = \Theta(n) + T(n-1) \rightsquigarrow T(n) = O(n^2)$.
- Average-case: $T(n) = \Theta(n) + T(3n/4) \rightsquigarrow$

Randomized selection

Time complexity?

- Worst-case: $T(n) = \Theta(n) + T(n-1) \rightsquigarrow T(n) = O(n^2)$.
- Average-case: $T(n) = \Theta(n) + T(3n/4) \rightsquigarrow T(n) = O(n)$.

Randomized selection

Time complexity?

- Worst-case: $T(n) = \Theta(n) + T(n-1) \rightsquigarrow T(n) = O(n^2)$.
- Average-case: $T(n) = \Theta(n) + T(3n/4) \rightsquigarrow T(n) = O(n)$.

Can we do better?

Randomized selection

Time complexity?

- Worst-case: $T(n) = \Theta(n) + T(n-1) \rightsquigarrow T(n) = O(n^2)$.
- Average-case: $T(n) = \Theta(n) + T(3n/4) \rightsquigarrow T(n) = O(n)$.

Can we do better?

Can we do selection in worst-case linear time?

Median of medians as pivot

Select(A, k):

1. Divide the input A into groups of 5.
2. Find the median of each group.
3. Recursively find the median m of all these $n/5$ medians.
4. Partition A with pivot m.
5. Do recursive call as in RandomizedSelect.

Median of medians as pivot

Select(A, k):

1. Divide the input A into groups of 5.
2. Find the median of each group.
3. Recursively find the median m of all these $n/5$ medians.
4. Partition A with pivot m.
5. Do recursive call as in RandomizedSelect.

Time complexity?

- Worst-case: $T(n) =$

Median of medians as pivot

Select(A, k):

1. Divide the input A into groups of 5.
2. Find the median of each group. $\leftarrow$
3. Recursively find the median m of all these $n/5$ medians.
4. Partition A with pivot m.
5. Do recursive call as in RandomizedSelect.

Time complexity?

- Worst-case: $T(n) = c \cdot n/5 +$

Median of medians as pivot

Select(A, k):

1. Divide the input A into groups of 5.
2. Find the median of each group.
3. Recursively find the median m of all these $n/5$ medians. $\leftarrow$
4. Partition A with pivot m.
5. Do recursive call as in RandomizedSelect.

Time complexity?

- Worst-case: $T(n) = c \cdot n/5 + T(n/5) +$

Median of medians as pivot

Select(A, k):

1. Divide the input A into groups of 5.
2. Find the median of each group.
3. Recursively find the median m of all these $n/5$ medians.
4. Partition A with pivot m. $\leftarrow$
5. Do recursive call as in RandomizedSelect.

Time complexity?

- Worst-case: $T(n) = c \cdot n/5 + T(n/5) + \Theta(n) +$

Median of medians as pivot

Select(A, k):

1. Divide the input A into groups of 5.
2. Find the median of each group.
3. Recursively find the median m of all these $n/5$ medians.
4. Partition A with pivot m.
5. Do recursive call as in RandomizedSelect. $\leftarrow$

Time complexity?

- Worst-case: $T(n) = c \cdot n/5 + T(n/5) + \Theta(n) + T(\;?\;)$

Median of medians as pivot

Select(A, k):

1. Divide the input A into groups of 5.
2. Find the median of each group.
3. Recursively find the median m of all these $n/5$ medians.
4. Partition A with pivot m.
5. Do recursive call as in RandomizedSelect.

Time complexity?

- Worst-case: $T(n) = c \cdot n/5 + T(n/5) + \Theta(n) + T(\text{?})$

$$= T(n/5) + T(\text{?}) + \Theta(n).$$

How good is MoM?

Select(A, k):

1. Divide the input A into groups of 5.
2. Find the median of each group.
3. Recursively find the median m of all these $n/5$ medians.
4. Partition A with pivot m.
5. Do recursive call as in RandomizedSelect.

$n = 30$

19	7	15	42	5	88	91	4	29	21
13	67	54	18	20	73	8	36	49	2
9	25	31	44	12	6	80	14	22	3

How good is MoM?

Select(A, k):

1. Divide the input A into groups of 5.
2. Find the median of each group.
3. Recursively find the median m of all these $n/5$ medians.
4. Partition A with pivot m.
5. Do recursive call as in RandomizedSelect.

$n = 30$

19	7	15	42	5	88	91	4	29	21
13	67	54	18	20	73	8	36	49	2
9	25	31	44	12	6	80	14	22	3

How good is MoM?

Select(A, k):

1. Divide the input A into groups of 5.
2. Find the median of each group.
3. Recursively find the median m of all these $n/5$ medians.
4. Partition A with pivot m.
5. Do recursive call as in RandomizedSelect.

$n = 30$

19	7	15	42	5	88	91	4	29	21
13	67	54	18	20	73	8	36	49	2
9	25	31	44	12	6	80	14	22	3

How good is MoM?

- m $\leq (n/5)/2 = n/10$ medians; also $\geq n/10$ medians.

$n = 30$, $m = 20$

19	7	15	42	5	88	91	4	29	21
13	67	54	18	20	73	8	36	49	2
9	25	31	44	12	6	80	14	22	3

How good is MoM?

- m $\leq (n/5)/2 = n/10$ medians; also $\geq n/10$ medians.
- For each median $\leq m$, there are at least 2 more in its group $\leq m$.

$n = 30$, $m = 20$

19	7	15	42	5	88	91	4	29	21
13	67	54	18	20	73	8	36	49	2
9	25	31	44	12	6	80	14	22	3

How good is MoM?

- m $\leq (n/5)/2 = n/10$ medians; also $\geq n/10$ medians.
- For each median $\leq m$, there are at least 2 more in its group $\leq m$.
- For each median $\geq m$, there are at least 2 more in its group $\geq m$.

$n = 30$, $m = 20$

19	7	15	42	5	88	91	4	29	21
13	67	54	18	20	73	8	36	49	2
9	25	31	44	12	6	80	14	22	3

How good is MoM?

- m $\leq (n/5)/2 = n/10$ medians; also $\geq n/10$ medians.
- For each median $\leq m$, there are at least 2 more in its group $\leq m$.
- For each median $\geq m$, there are at least 2 more in its group $\geq m$.

$n = 30$, $m = 20$

19	7	15	42	5	88	91	4	29	21
13	67	54	18	20	73	8	36	49	2
9	25	31	44	12	6	80	14	22	3

How good is MoM?

- m $\leq (n/5)/2 = n/10$ medians; also $\geq n/10$ medians.
- For each median $\leq m$, there are at least 2 more in its group $\leq m$.
- For each median $\geq m$, there are at least 2 more in its group $\geq m$.
- In total, we can guarantee $3n/10$ numbers $\leq m$; also $3n/10$ numbers $\geq m$.

$n = 30$, $m = 20$

19	7	15	42	5	88	91	4	29	21
13	67	54	18	20	73	8	36	49	2
9	25	31	44	12	6	80	14	22	3

How good is MoM?

- m $\le (n/5)/2 = n/10$ medians; also $\ge n/10$ medians.
- For each median $\le m$, there are at least 2 more in its group $\le m$.
- For each median $\ge m$, there are at least 2 more in its group $\ge m$.
- In total, we can guarantee $3n/10$ numbers $\le m$; also $3n/10$ numbers $\ge m$.

$n = 30$, $m = 20$, 15 numbers $\le m$, 15 numbers $\ge m$

19	7	15	42	5	88	91	4	29	21
13	67	54	18	20	73	8	36	49	2
9	25	31	44	12	6	80	14	22	3

Median of medians as pivot

Select(A, k):

1. Divide the input A into groups of 5.
2. Find the median of each group.
3. Recursively find the median m of all these $n/5$ medians.
4. Partition A with pivot m.
5. Do recursive call as in RandomizedSelect.

Time complexity?

- Worst-case: $T(n) = T(n/5) + T(\,?\,) + \Theta(n)$

Median of medians as pivot

Select(A, k):

1. Divide the input A into groups of 5.
2. Find the median of each group.
3. Recursively find the median m of all these $n/5$ medians.
4. Partition A with pivot m.
5. Do recursive call as in RandomizedSelect.

Time complexity?

- Worst-case: $T(n) = T(n/5) + T($ n-3n/10 $) + \Theta(n)$

Median of medians as pivot

Select(A, k):

1. Divide the input A into groups of 5.
2. Find the median of each group.
3. Recursively find the median m of all these $n/5$ medians.
4. Partition A with pivot m.
5. Do recursive call as in RandomizedSelect.

Time complexity?

- Worst-case: $$T(n) = T(n/5) + T(\text{n-3n/10}) + \Theta(n)$$ $$= T(n/5) + T(\text{7n/10}) + \Theta(n).$$

Median of medians as pivot

Select(A, k):

1. Divide the input A into groups of 5.
2. Find the median of each group.
3. Recursively find the median m of all these $n/5$ medians.
4. Partition A with pivot m.
5. Do recursive call as in RandomizedSelect.

Time complexity?

- Worst-case: $T(n) = T(n/5) + T(\text{n-3n/10}) + \Theta(n)$
  $$= T(n/5) + T(\text{7n/10}) + \Theta(n).$$
- (Exercise) Prove that $T(n) = O(n)$.
- (Exercise) What happens if we group by 3, or 7?