Lecture #8
Selection

Algorithm

JBNU Spring 2021

Jinhong Jung

In Previous Lecture

Studied various sorting algorithms

- Basic sorting algorithms in $\Theta(n^2)$
  - Bubble, insertion, and selection sort
- Advanced sorting algorithms in $\Theta(n \log n)$
  - Merge, quick, and heap sort
- Special sorting algorithms in $\Theta(n)$
  - Counting and radix sort

In This Lecture

Selection algorithm

- Find i-th smallest number in an array
- Can we find the number in linear time for a worst case?

Outline

Selection problem

Linear selection algorithm on average

Linear selection algorithm in a worst case

Selection Problem (1)

Selection problem

- **Input:** unsorted array A of size n & parameter $1 \leq i \leq n$
- **Output:** i-th smallest number in the array

Example

- Input array $\mathrm{A} = \{7, 10, 4, 3, 20, 15\}$
  - Given $i = 3$, the answer is 7
  - Given $i = 4$, the answer is 10

Application

- Given n scores of students,
  - Find the maximum and minimum among the scores
  - Find the median of the scores

Selection Problem (2)

Naïve methods for the selection problem

- **M1)** Double-looped sequential search
  - For each step, find the minimum in the array and exclude it
  - Repeat the above i times
  - This requires $O(n^2)$ time complexity
- **M2)** Sort the array & return i-th value in the sorted array
  - This requires $O(n \log n)$ time complexity (e.g., merge & quick)
  - Partial sort can be possible using heap
- Note that we need to check all values in the array at least once $\Rightarrow \Omega(n)$
- **Q. Can we find i-th smallest element in $\Theta(n)$?**

Outline

Selection problem

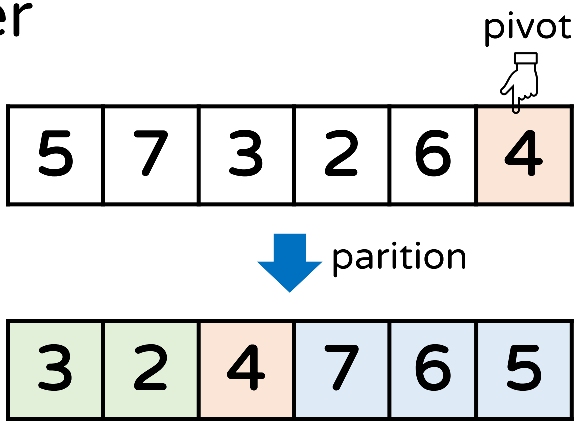
Linear selection algorithm on average

Linear selection algorithm in a worst case

Selection Algorithm (1)

Observation from quick sort's partitioning

- After the partition function, the pivot is correctly located in the sorted order



  - The pivot's location is 3 ⇒ it's the 3-rd smallest element
    - Case 1) If i < the pivot's index, repeat the selection in the left partition (i.e., do not need to check the right partition)
    - Case 2) If i = the pivots' index, return the pivot
    - Case 3) If i > the pivot's index, repeat the selection in the right partition

Selection Algorithm (2)

Idea of selection algorithm (a.k.a. quick-select)

- [**Divide**] split the input based on the pivot such that
  - Elements before pivot $\leq$ pivot $\leq$ elements after pivot
- [**Conquer**] find i-th smallest element in either of the left, the pivot, or the right partition

Example: find 2-nd smallest element ($i = 2$)

31	8	48	73	11	3	20	29	65	15

parition

8	11	3	15	31	48	20	29	65	73

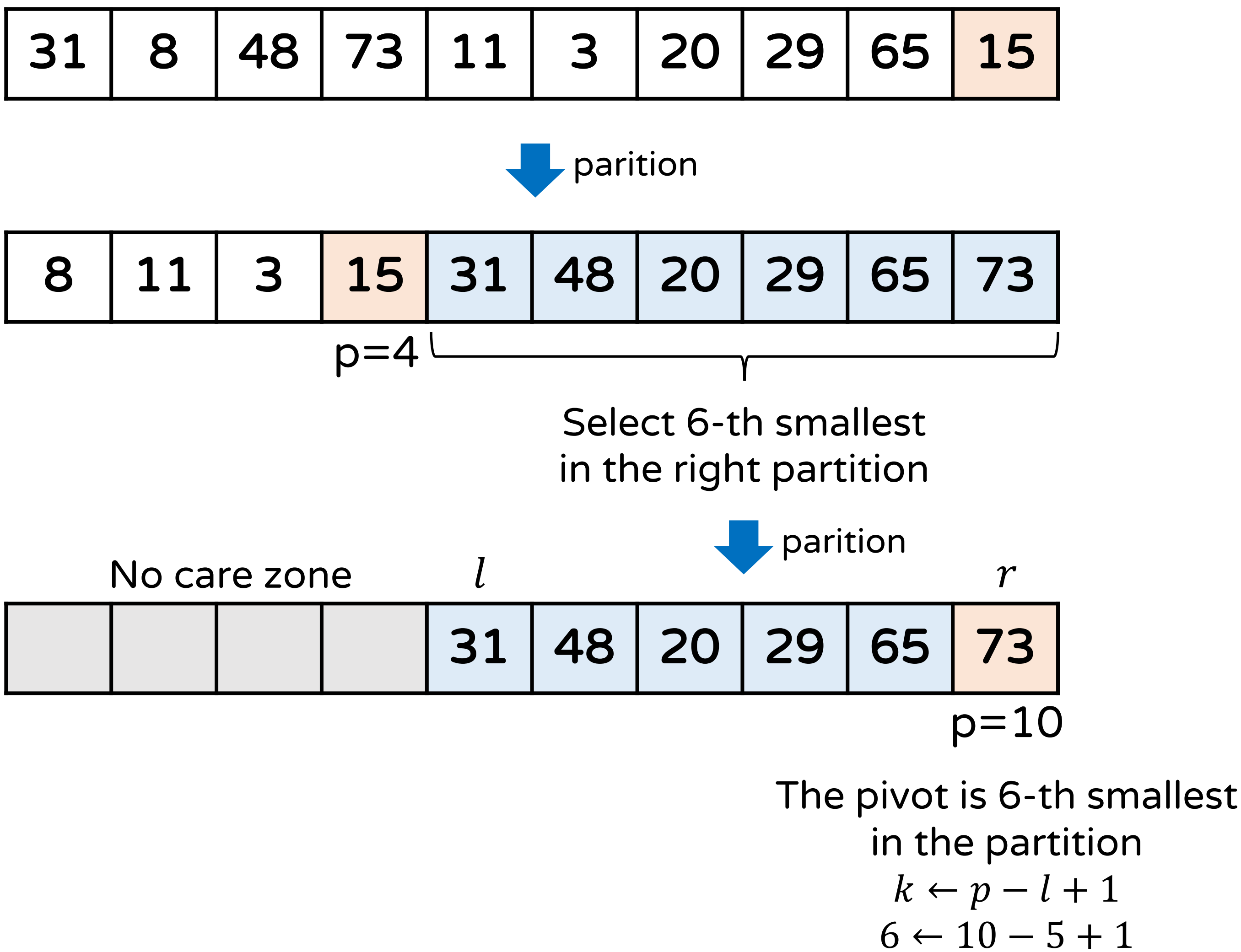
p=4

No care zone

Select 2nd smallest
in the left partition

Selection Algorithm (3)

Example: find 10-th smallest element ($i = 10$)

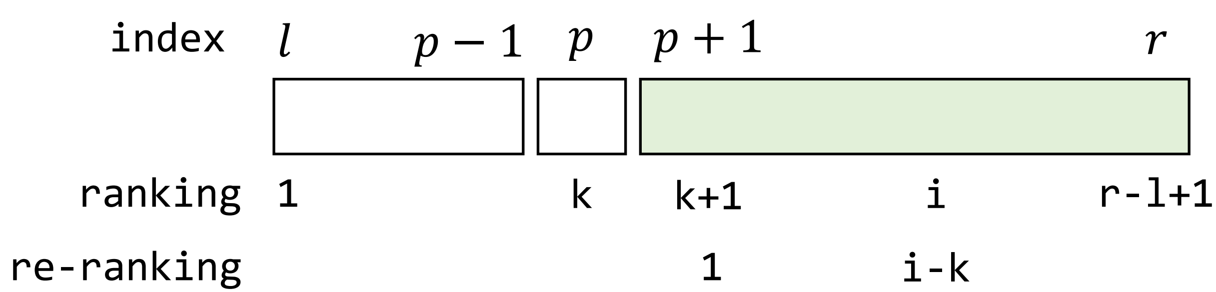


Selection Algorithm (4)

Pseudocode of selection

```
def select(A, l, r, i): # find i-th smallest in [l, r]
    if l == r:
        return A[l]  # for one element, i should be 1

    p ← partition(A, l, r)
    k ← p – l + 1    # the pivot is k-th smallest in [l, r]

    if i < k:         return select(A, l, p - 1, i)
    else if i == k: return A[p]
    else if i > k:  return select(A, p + 1, r, i - k)
```

When we go to the right partition, index and ranking are re-calculated



Selection Algorithm (5)

Time complexity of selection

```
def select(A, l, r, i):
    if l == r: return A[l]
    p ← partition(A, l, r)
    k ← p - l + 1

    if i < k:         return select(A, l, p - 1, i)
    else if i == k: return A[p]
    else if i > k:  return select(A, p + 1, r, i - k)
```

- For input size $n = r - l + 1$,
  - The size of left partition is $k - 1$ (i.e., $(p - 1) - l + 1 = p - l = k - 1$)
  - The size of right partition is $n - k$ (i.e., $[k - 1][1][n - k] = n$)

$$T(n) \leq \max[T(k-1), T(n-k)] + Cn$$

Selection Algorithm (6)

Average time complexity of selection

- Assume the pivot's ranking k is uniformly distributed and $T(n)$ monotonically increases
- Then, its expectation is represented as follows:

$$T(n) \leq \frac{1}{n}\left(\sum_{k=1}^{n} \max[T(k-1), T(n-k)]\right) + Cn$$

[See Appendix] $$\leq \frac{2}{n}\left(\sum_{k=\lfloor n/2 \rfloor}^{n-1} T(k)\right) + Cn$$

- Then, $T(n) = O(n) = \Omega(n) = \Theta(n)$ (refer to 142p)
  - Proved by strong induction; for $k \in [n/2, n)$, assume $T(k) \leq ck$

Selection Algorithm (7)

Worst-case time complexity of selection

- What if one of the partitions becomes empty for each step? Then, it is represented as follows:

$$T(n) = T(n-1) + Cn$$

- Thus, the time complexity is $\Theta(n^2)$
  - i.e., the inefficiency is from the perfect skewness
  - This is not good because we can do this in $n \log n$ time using sort

- Can we improve the complexity to $\Theta(n)$ even for a worst case?

Outline

Selection problem

Linear selection algorithm on average

Linear selection algorithm in a worst case

Observation On Partitioning

For each step, what if

- The input partition is divided by 1:9 ratio, and
- It goes to the right partition of 9 ratio

$$T(n) = T\left(\frac{9}{10}n\right) + \Theta(n)$$

  - By Master Theorem, it's $\Theta(n)$
  - This holds even when the input is divided by 1:99 ratio

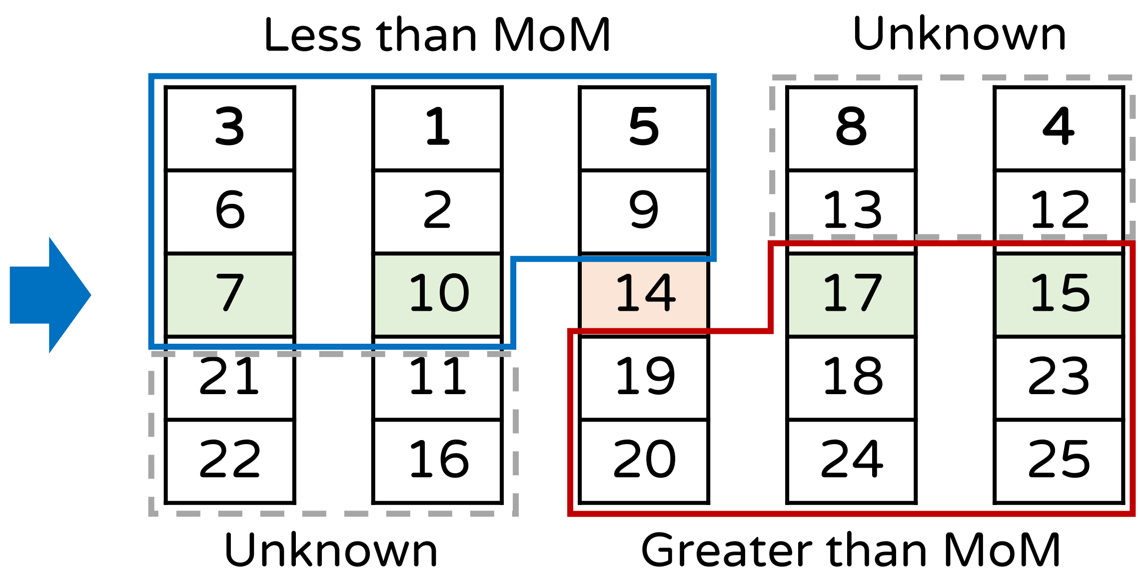
- This implies that it can be linear if
  - The input is divided by a ratio even though the split is skewed
  - The overhead for the split should be in $\Theta(n)$

Median Of Medians Algorithm (1)

Ideas of ‘Median of Medians’ (a.k.a. mom-select)

- Let’s divide the input into small groups and get the median of each group (the size of a group is 5)
- Use the median of the $\lceil n/5 \rceil$ medians as a pivot
  - Then, this guarantees that the pivot’s ranking is between top 30% and 70% ⇒ no perfect skewness for partitioning!!

Given randomly permuted 25 numbers, if we partition the numbers based on MoM as pivot



Median Of Medians Algorithm (2)

Pseudocode of mom-selection

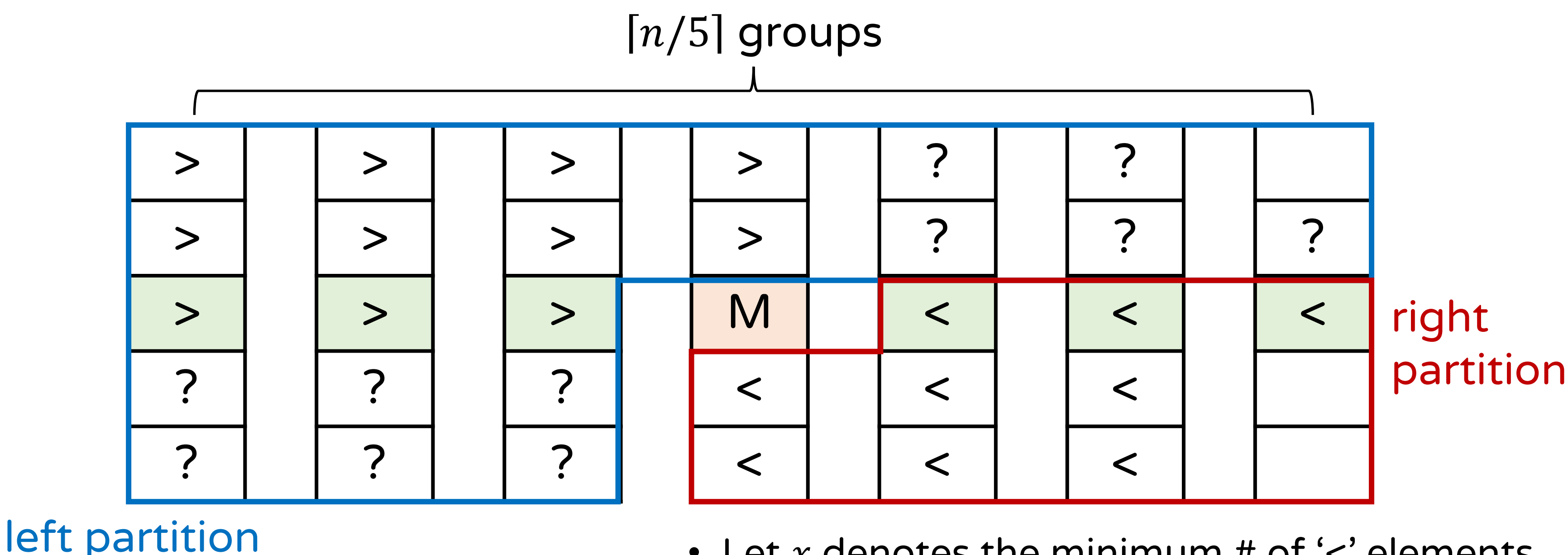
```
def mom-select(A, l, r, i):
    if r - l + 1 <= 5: # its size <= 5
        return select(A, l, r, i) # i-th smallest element of A in [l, r]

    1) divide A into ⌈n/5⌉ groups where the group's size is 5

    2) m_i ← get the median of each group for 1 ≤ i ≤ ⌈n/5⌉
              # e.g., select(A, l_i, r_i, 3) for size of 5

    3) M ← get the median of medians B = {m_1, ⋯, m_⌈n/5⌉}
              # mom-select(B, 1, ⌈n/5⌉, ⌈(1/2)⌈n/5⌉⌉)

    4) p ← get the pivot's index after partitioning A based on M
              # swap(A[r], A[idx(M)]) and do Lomuto partition

    5) return recursively call mom-select()
       on either of the left, the pivot, and the right partition
```

Median Of Medians Algorithm (3)

Skewness analysis

<: element greater than M
>: element less than M
?: unknown

- Given n items, the partition function of mom-selection divides the input for a worst case as follows
  - e.g., when all unknown elements go to the left partition



- Let x denotes the minimum # of ‘<’ elements
- After partitioning on M, at least x elements are in the right partition

|left partition| : |right partition| $= n - x - 1 : x$

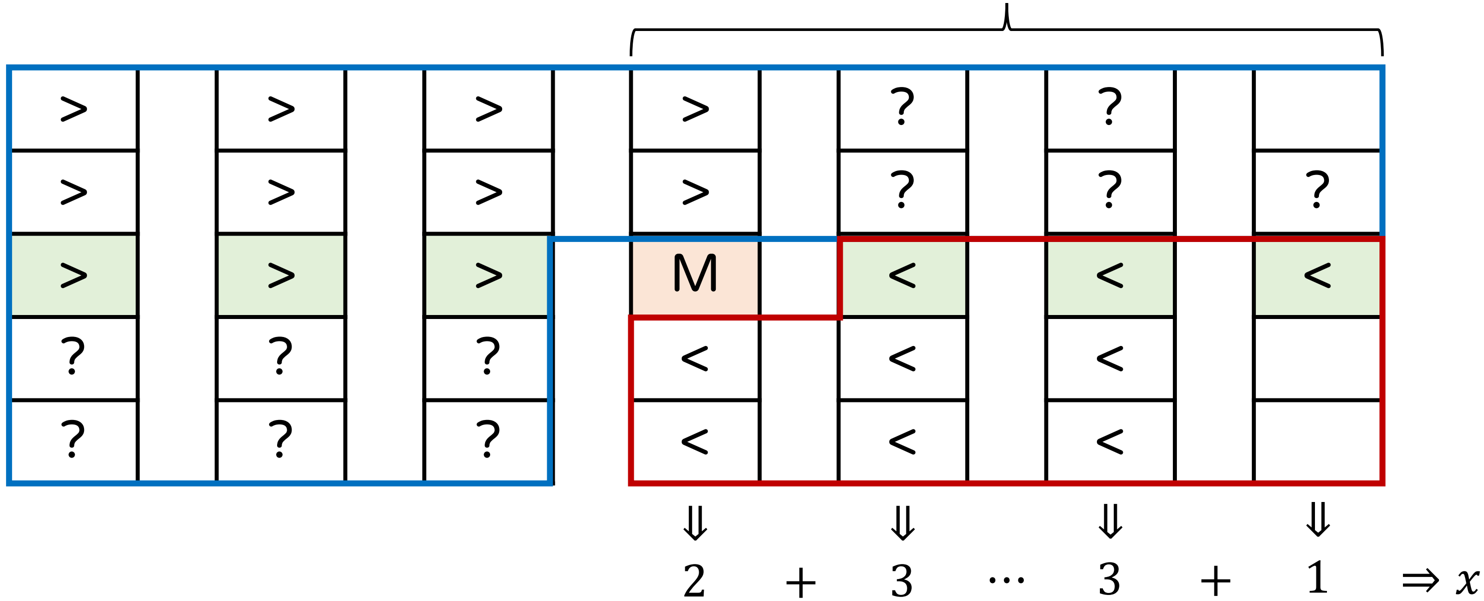
Median Of Medians Algorithm (4)

Skewness analysis

- In right half $\lceil n/5 \rceil$ groups
  - The first group contributes 2 elements to x
  - The last group contributes 1 element to x
  - Each of remaining group contributes 3 elements to x

# of rem. groups = $\left\lceil \frac{1}{2} \lceil n/5 \rceil \right\rceil - 2$

$\left\lceil \frac{1}{2} \lceil n/5 \rceil \right\rceil$ groups



Median Of Medians Algorithm (5)

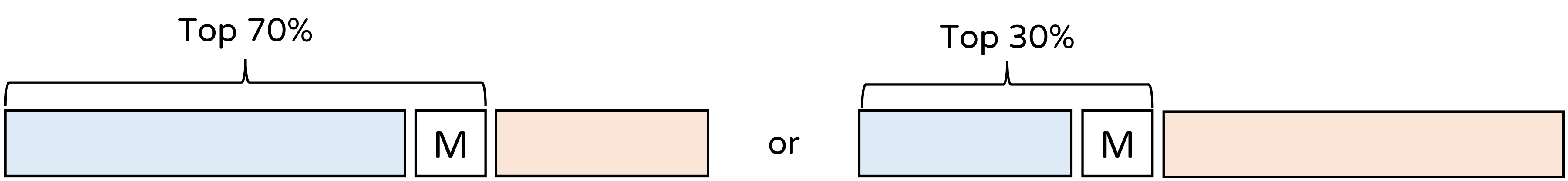
Skewness analysis

- In right half $\lceil n/5 \rceil$ groups
  - The first group contributes 2 elements to x
  - The last group contributes 1 element to x
  - Each of remaining group contributes 3 elements to x

$$x = 3\times\left(\left\lceil\frac{1}{2}\lceil n/5\rceil\right\rceil - 2\right) + 3 \geq \frac{3}{10}n - 3$$

- The right partition has at least $\frac{3}{10}n - 3$ elements!

|left partition| : |right partition| $= n - x - 1 : x = \frac{7}{10}n + 2 : \frac{3}{10}n - 3$



At least, the pivot's ranking is between top 30% and 70%

Median Of Medians Algorithm (6)

Worst-case time complexity of mom-select

```
def mom-select(A, l, r, i):
    if r - l + 1 <= 5: # its size <= 5
        return select(A, l, r, i)
```

base case in $\Theta(1)$

$T(n)$:

$\Theta(n) \Leftarrow$ 1) divide A into $\lceil n/5 \rceil$ groups where the group's size is 5

$\Theta(n) \Leftarrow$ 2) m_i $\leftarrow$ get the median of each group for $1 \leq i \leq \lceil n/5 \rceil$

e.g., select(A, l_i, r_i, 3) for size of 5

$T\left(\left\lceil \frac{n}{5} \right\rceil\right) \Leftarrow$ 3) M $\leftarrow$ get the median of medians $\mathrm{B} = \{m_1, \cdots, m_{\lceil n/5 \rceil}\}$

mom-select(B, 1, $\lceil n/5 \rceil$, $\left\lceil \frac{1}{2} \lceil n/5 \rceil \right\rceil$)

$\Theta(n) \Leftarrow$ 4) p $\leftarrow$ get the pivot's index after partitioning A based on M

swap(A[r], A[idx(M)]) and do Lomuto partition

$T\left(\frac{7}{10}n + 2\right) \Leftarrow$ 5) **return** recursively call mom-select()
on either of **the left**, the pivot, and the right partition
selected as a worst case

$$T(n) \leq T\left(\left\lceil \frac{n}{5} \right\rceil\right) + T\left(\frac{7}{10}n + 2\right) + \Theta(n)$$

Median Of Medians Algorithm (7)

Worst-case time complexity of mom-select

- Then, the time complexity is $\Theta(n)$
- **Proof)** Assume $T(i) \leq ci$ holds for $n_0 \leq i < k$

$$\begin{aligned} T(k) &\leq T\left(\left\lceil\frac{k}{5}\right\rceil\right) + T\left(\frac{7}{10}k + 2\right) + \Theta(k) \\ &\leq T\left(\frac{k}{5} + 1\right) + T\left(\frac{7}{10}k + 2\right) + \Theta(k) \\ &\leq c\left(\frac{k}{5} + 1\right) + c\left(\frac{7}{10}k + 2\right) + \Theta(k) \quad \leftarrow \text{using assumptions} \\ &\leq c\left(\frac{9}{10}k + 3\right) + \Theta(k) = ck - \frac{c}{10}k + 3c + \Theta(k) \leq ck \end{aligned}$$

  - By selecting c such that $-\frac{c}{10}k > 3c + \Theta(k)$, it holds!
  - Thus, $T(n) \leq cn = O(n)$ for any n by induction
    - Trivially, $T(n) = \Omega(n)$; thus, $T(n) = \Theta(n)$

Discussion

Asymptotically, mom-select guarantees $\Theta(n)$ time for a worst case!

- But it has a large coefficient inside $\Theta(n)$ actually
  - Incurred by selecting MoM from the input
- Practically, a hybrid strategy is used (called intro-select)
  - At initial, start with quick-select and switch to mom-select if it recurses too many times
- However, you should notice the idea behind mom-select
  - If we guarantee that the sub-problem size decreases over recursions with a linear overhead, then the final complexity dose not skyrocket

$$T(n) = T\left(\frac{9}{10}n\right) + \Theta(n)$$

What You Need To Know

Linear selection algorithm on average

- quick-select() using Lomuto partition to select i-th smallest element in an array
- Has $\Theta(n)$ on average, and $\Theta(n^2)$ for a worst case

Linear selection algorithm in a worst case

- A fixed skewness in partitioning with a linear overhead leads to $\Theta(n)$ time for a worst case
- For that, mom-select() uses the median of medians as a pivot, and partitions the input array by MoM
- Has $\Theta(n)$ for a worst case

In Next Lecture

Advanced data structure

- Self-balancing binary search tree
  - Red-black tree

- Must study or review **binary search tree** in advance!

Thank You

Appendix (1)

- If n is even,

$T(n)$ is a monotonically increasing function!
Thus, $T(1) \leq T(n-1)$

$$\sum_{k=1}^{n} \max[T(k-1), T(n-k)] = \max[T(1), T(n-1)] \Rightarrow T(n-1)$$

$$+ \max[T(2), T(n-2)] \Rightarrow T(n-2)$$

$$+ \cdots$$

$$+ \max\left[T\left(\frac{n}{2}-1\right), T\left(\frac{n}{2}\right)\right] \Rightarrow T\left(\frac{n}{2}\right)$$

$$+ \max\left[T\left(\frac{n}{2}\right), T\left(\frac{n}{2}-1\right)\right] \Rightarrow T\left(\frac{n}{2}\right)$$

$$+ \cdots$$

$$+ \max[T(n-2), T(2)] \Rightarrow T(n-2)$$

$$+ \max[T(n-1), T(1)] \Rightarrow T(n-1)$$

$$\leq 2 \sum_{k=\left\lfloor\frac{n}{2}\right\rfloor}^{n-1} T(k)$$

Appendix (2)

- If n is odd, you can prove it similarly to the case of even
  - Note $n = \left\lfloor \frac{n}{2} \right\rfloor + 1 + \left\lfloor \frac{n}{2} \right\rfloor = n = \left\lfloor \frac{n}{2} \right\rfloor + \left\lceil \frac{n}{2} \right\rceil$;
  - Then, you can obtain the following

$$\sum_{k=1}^{n} \max[T(k-1), T(n-k)]$$

$$= 2 \sum_{k=\left\lceil \frac{n}{2} \right\rceil}^{n-1} T(k) + T\left(\left\lfloor \frac{n}{2} \right\rfloor\right)$$

$$\leq 2 \sum_{k=\left\lceil \frac{n}{2} \right\rceil}^{n-1} T(k) + 2T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) = 2 \sum_{k=\left\lfloor \frac{n}{2} \right\rfloor}^{n-1} T(k)$$