CSE3081 Design and Analysis of Algorithms

Dept. of Computer Engineering,
Sogang University

Chapter 9. Medians and Order Statistics

Order Statistics

- The ith order statistic of a set of n elements is the ith smallest element.
  - First order statistic → minimum
  - nth order statistic $(i = n)$ → maximum

- Median
  - The "halfway point" of the set.
  - When n is odd, the median is unique, occurring at $i = (n + 1)/2$.
  - When n is even, the lower median occurs at $i = n/2$ and the upper median occurs at $i = \frac{n}{2} + 1$.
  - For simplicity, we will use the phrase "median" to refer to the lower median.

Selection Problem

- The problem of selecting the ith order statistic from a set of n distinct numbers.

- **Input:** A set A of n distinct numbers and an integer i, with $1 \leq i \leq n$.
- **Output:** The element $x \in A$ that is larger than exactly $i - 1$ other elements of A.

- We can solve the selection problem in $O(n \log n)$ time simply by
  - sorting the numbers using heapsort or merge sort
  - and then outputting the ith element in the sorted array.

- Can we do better than $O(n \log n)$?

9.1 Minimum and Maximum

Finding Minimum or Maximum

- How many comparison are necessary to determine the minimum of n elements?

- The upper bound is $n - 1$, as shown in the algorithm.

```
MINIMUM(A, n)
1  min = A[1]
2  for i = 2 to n
3      if min > A[i]
4          min = A[i]
5  return min
```

- The lower bound is also $n - 1$, because we cannot determine whether an element is smaller than all other elements without comparing the element with each of the other elements.

- Finding the maximum is the same: we need $n - 1$ comparisons.

Finding Minimum or Maximum Simultaneously

- If we need to find the minimum and the maximum, we would need $2n - 2$ comparisons to separately find the two.
- Although it is asymptotically optimal, we can actually improve the constant and find both the minimum and the maximum using at most $3\lfloor n/2 \rfloor$ comparisons.

- Strategy
  - Maintain the minimum and the maximum elements seen thus far.
  - Initially, if n is odd, set both the minimum and the maximum to the value of the first element. If n is even, compare the first two elements to determine the minimum and the maximum.
  - For each pair of elements,
    - Compare the elements each other.
    - Compare the smaller with the minimum.
    - Compare the larger with the maximum.
- How many comparisons do we need using this strategy?

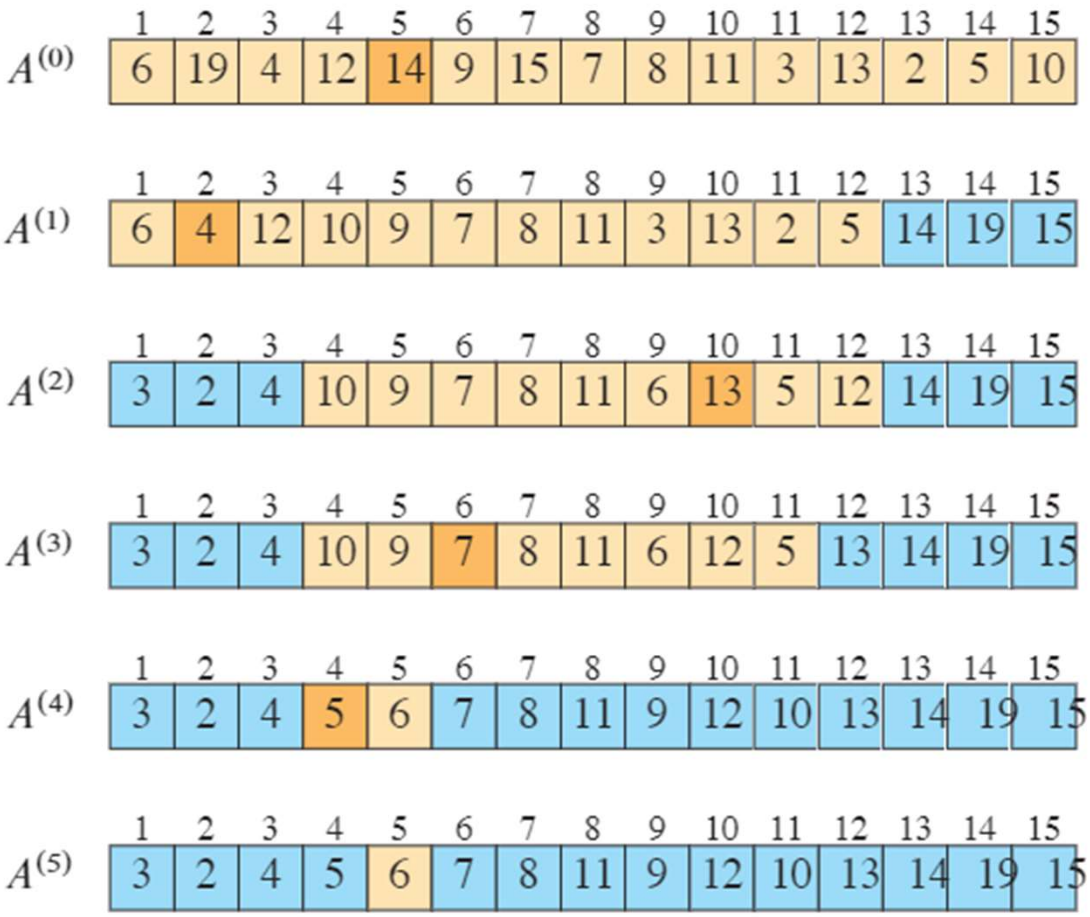
9.2 Selection in Expected Linear Time

Selection Problem

- Finding the ith order statistic for any value i is more difficult than finding the minimum or the maximum.
- However, we can still do it in $\Theta(n)$ time.
- First, we look at the expected running time of the randomized algorithm, RANDOMIZED-SELECT.
  - The algorithm use RANDOMIZED-PARTITION from the quicksort algorithm.

```
RANDOMIZED-SELECT(A, p, r, i)
1  if p == r
2       return A[p]        // 1 ≤ i ≤ r − p + 1 when p == r means that i = 1
3  q = RANDOMIZED-PARTITION(A, p, r)
4  k = q − p + 1
5  if i == k
6       return A[q]        // the pivot value is the answer
7  elseif i < k
8       return RANDOMIZED-SELECT(A, p, q − 1, i)
9  else return RANDOMIZED-SELECT(A, q + 1, r, i − k)
```

RANDOMIZED-SELECT: example



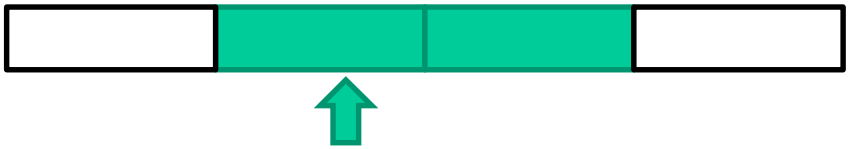
	p	r	i	partitioning	helpful?
$A^{(0)}$	1	15	5		
				1	no
$A^{(1)}$	1	12	5		
				2	yes
$A^{(2)}$	4	12	2		
				3	no
$A^{(3)}$	4	11	2		
				4	yes
$A^{(4)}$	4	5	2		
				5	yes
$A^{(5)}$	5	5	1		

RANDOMIZED-SELECT: Worst-case Running Time

- RANDOMIZED-SELECT calls RANDOMIZED-PARTITION, which randomly selects a pivot and does the partition.
- In the worst-case, the largest element is always selected as pivot.
- Then, the running time is $T(n) = T(n-1) + \Theta(n) = \Theta(n^2)$.

RANDOMIZED-SELECT: Expected Running Time

- Suppose that each time the algorithm randomly selects a pivot element, the pivot lies somewhere between the second and third quartiles ("middle half")



- If the ith smallest element is less than the pivot, then all the elements greater than the pivot are ignored in all future recursive calls.
  - These ignored elements include at least the uppermost quartile.

- If the ith smallest element is greater than the pivot, then all the elements less than the pivot – at least the first quartile – are ignored in all future recursive calls.

- Either way, at most 3/4 of the remaining elements are left "in play".

RANDOMIZED-SELECT: Expected Running Time

- If the pivot always lies somewhere between the second and third quartiles, the recurrence for the running time would be $T(n) = T\left(\frac{3n}{4}\right) + \Theta(n)$.
- In this case, the master method says $T(n) = \Theta(n)$.

- The pivot does not always fall into the middle half. It falls into the middle half with probability $1/2$.

- We can say the partitioning is helpful when the pivot is in the middle half.

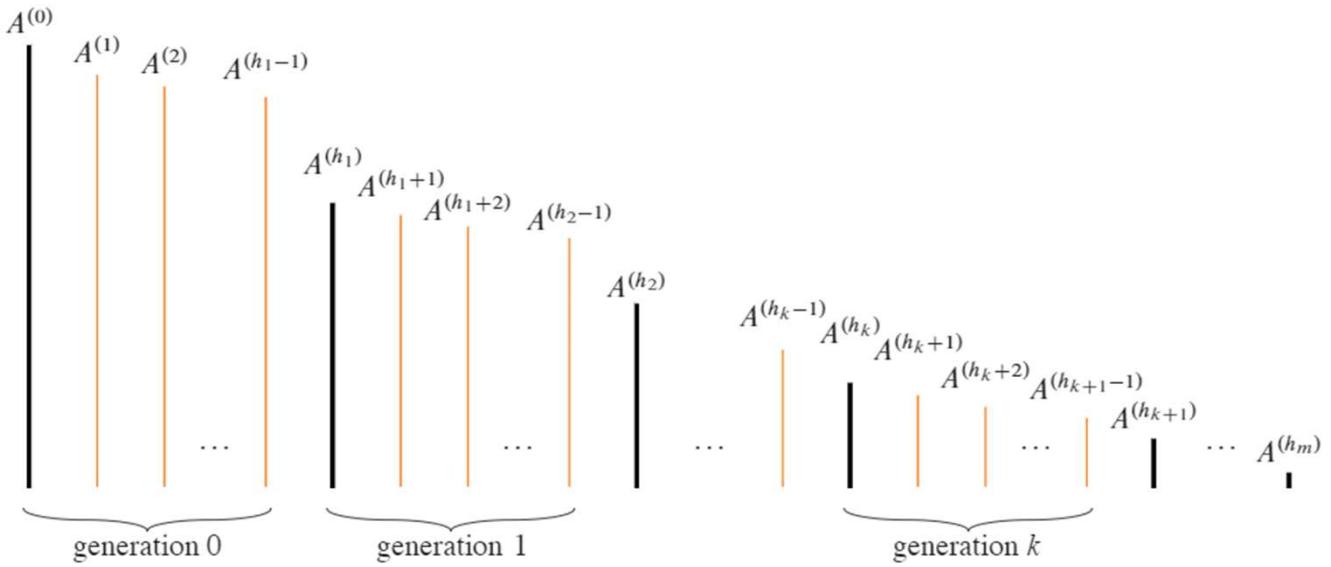
RANDOMIZED-SELECT: Proof

- Theorem 9.2
  - The procedure RANDOMIZED-SELECT on an input array of n distinct elements has an expected running time of $\Theta(n)$.

- Proof
  - Since not every partitioning is helpful, let us give each partitioning an index starting at 0 and denote by $\langle h_0, h_1, h_2, \ldots, h_m \rangle$ the sequence of partitionings that are helpful.
  - Although the number m of helpful partitionings is a random variable, we can bound it, since after at most $\lceil \log_{4/3} n \rceil$ helpful partitionings, only one element remains in play.
  - Let's denote $n_k = |A^{(h_k)}|$, where $n_0 = |A^{(0)}|$ is the original problem size.
    - We consider the dummy 0th partitioning as helpful, so that $h_0 = 0$.
  - $n_k = |A^{(h_k)}| \le \left(\frac{3}{4}\right) |A^{(h_k-1)}| = \left(\frac{3}{4}\right) n_{k-1}$ for $k = 0, 1, 2, \ldots, m$.
  - Iterating $n_k \le \left(\frac{3}{4}\right) n_{k-1}$, we have $n_k \le \left(\frac{3}{4}\right)^k n_0$ for $k = 0, 1, 2, \ldots, m$.

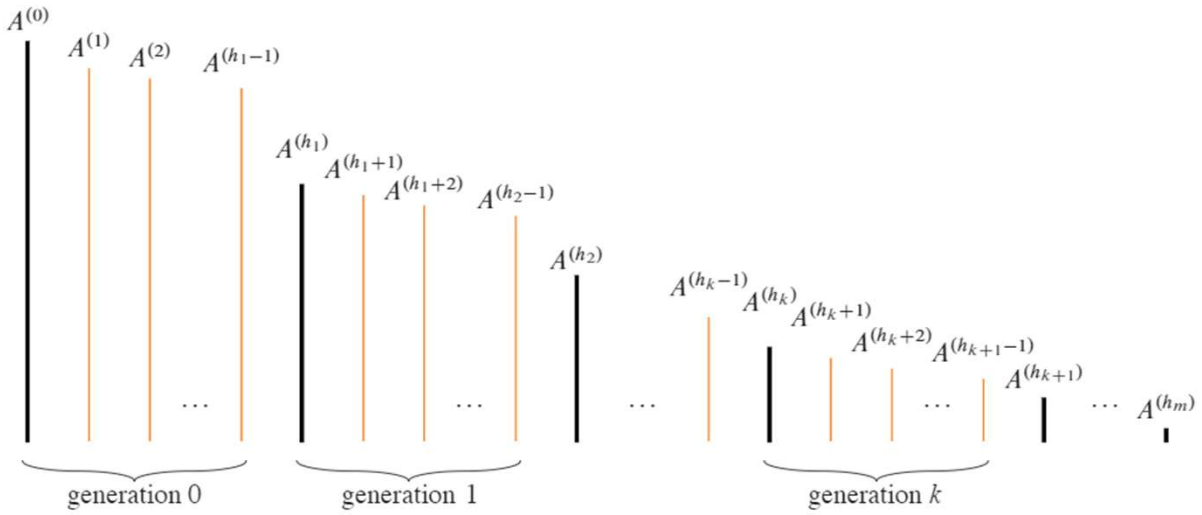
RANDOMIZED-SELECT: Proof

- Proof (cont.)
  - we break up the sequence of sets $A^{(j)}$ into m generations consisting of consecutively partitioned sets, starting with the result $A^{(h_k)}$ of a helpful partitioning and ending with the last set $A^{(h_{k+1}-1)}$ before the next helpful partitioning.
  - For each set of elements $A^{(j)}$ in the kth generation, we have $\left|A^{(j)}\right| \leq \left|A^{(h_k)}\right| = n_k \leq \left(\frac{3}{4}\right)^k n_0$.



RANDOMIZED-SELECT: Proof

- Proof (cont.)
  - We define the random variable $X_k = h_{k+1} - h_k$ for $k = 0, 1, 2, \ldots, m-1$.
  - X_k is the number of sets in the kth generation.
  - The probability that a partitioning is helpful is at least 1/2.
  - The probability is actually even higher, since a partitioning is helpful even if the pivot does not fall into the middle half but the ith smallest element happens to lie in the smaller side of the partitioning.
  - Using the lower bound, we can say $E[X_k] \leq 2$ for $k = 0, 1, 2, \ldots, m-1$.



RANDOMIZED-SELECT: Proof

- Proof (cont.)
  - Let's drive an upper bound on how many comparisons are made altogether during partitioning, which dominates the running time.
  - The jth partitioning takes the set $A^{(j-1)}$ of elements in play, and it compares the randomly chosen pivot with all other $\left|A^{(j-1)}\right| - 1$ elements.
  - In other words, The jth partitioning makes fewer than $\left|A^{(j-1)}\right|$ comparisons.
  - The sets in the kth generations have $\left|A^{(h_k)}\right|, \left|A^{(h_k+1)}\right|, \ldots, \left|A^{(h_k+X_k-1)}\right|$.
  - Thus, the total number of comparisons during partitioning is less than,

$$\begin{aligned}\sum_{k=0}^{m-1}\sum_{j=h_k}^{h_k+X_k-1}|A^{(j)}| &\le \sum_{k=0}^{m-1}\sum_{j=h_k}^{h_k+X_k-1}|A^{(h_k)}| \\ &= \sum_{k=0}^{m-1} X_k\,|A^{(h_k)}| \\ &\le \sum_{k=0}^{m-1} X_k \left(\frac{3}{4}\right)^k n_0\,.\end{aligned}$$

RANDOMIZED-SELECT: Proof

- Proof (cont.)
  - Since $E[X_k] \leq 2$, we have the expected total number of comparisons during partitioning as

$$
\begin{aligned}
\mathrm{E}\left[\sum_{k=0}^{m-1} X_k \left(\frac{3}{4}\right)^k n_0\right] &= \sum_{k=0}^{m-1} \mathrm{E}\left[X_k \left(\frac{3}{4}\right)^k n_0\right] \quad \text{(by linearity of expectation)} \\
&= n_0 \sum_{k=0}^{m-1} \left(\frac{3}{4}\right)^k \mathrm{E}[X_k] \\
&\leq 2n_0 \sum_{k=0}^{m-1} \left(\frac{3}{4}\right)^k \\
&< 2n_0 \sum_{k=0}^{\infty} \left(\frac{3}{4}\right)^k \\
&= 8n_0
\end{aligned}
$$

  - Since n_0 is the size of the original array A, we can conclude that the expected running time is $O(n)$.
  - Since all n elements are examined in the first call to RANDOMIZED-PARTITION, we know that the running time is $\Omega(n)$.
  - Thus, the expected running time is $\Theta(n)$.

9.3 Selection in Worst-case Linear Time

Selection in Linear Time in the Worst Case

- RANDOMIZED-SELECT achieves $\Theta(n)$ expected running time. However, in the worst case, the running time is $\Theta(n^2)$.

- In this section, we look at algorithm SELECT which achieves $\Theta(n)$ in the worst case.
  - In the practical sense, using RANDOMIZED-SELECT may be better.

- SELECT guarantees a good split by choosing a provably good pivot when partitioning the array.
  - The algorithm finds the pivot recursively.

- There are two invocations of SELECT: one to find a good pivot, and a second to recursively find the desired order statistic.

PARTITION-AROUND

- Similar to PARTITION, but modified to take the element to partition around as an additional input parameter.
  - $q =$ PARTITION-AROUND(A, p, r, x) → partition around x (Use x as pivot).
- Like PARTITION, PARTITION-AROUND algorithm returns index of the pivot.

SELECT: pseudocode

```
SELECT(A, p, r, i)
 1  while (r − p + 1) mod 5 ≠ 0
 2      for j = p + 1 to r                          // put the minimum into A[p]
 3          if A[p] > A[j]
 4              exchange A[p] with A[j]
 5      // If we want the minimum of A[p : r], we're done.
 6      if i == 1
 7          return A[p]
 8      // Otherwise, we want the (i − 1)st element of A[p + 1 : r].
 9      p = p + 1
10      i = i − 1
11  g = (r − p + 1)/5                               // number of 5-element groups
12  for j = p to p + g − 1                          // sort each group
13      sort ⟨A[j], A[j + g], A[j + 2g], A[j + 3g], A[j + 4g]⟩ in place
14  // All group medians now lie in the middle fifth of A[p : r].
15  // Find the pivot x recursively as the median of the group medians.
16  x = SELECT(A, p + 2g, p + 3g − 1, ⌈g/2⌉)
17  q = PARTITION-AROUND(A, p, r, x)                // partition around the pivot
18  // The rest is just like lines 3–9 of RANDOMIZED-SELECT.
19  k = q − p + 1
20  if i == k
21      return A[q]                                 // the pivot value is the answer
22  elseif i < k
23      return SELECT(A, p, q − 1, i)
24  else return SELECT(A, q + 1, r, i − k)
```

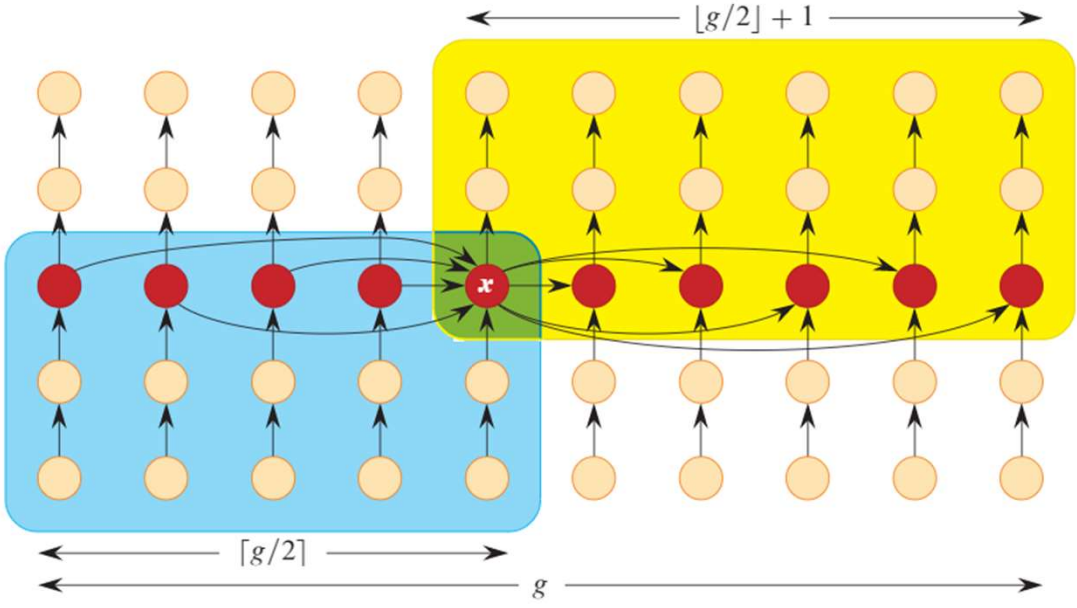
SELECT: explanation

- Lines 1-10 reduces the $r - p + 1$ elements in the subarray until it is divisible by 5.
- The while loop executes 0 to 4 times, each time rearranging the elements of $A[p:r]$ so that $A[p]$ contains the minimum element.
- If $i = 1$, which means we actually want the minimum element, then the procedure returns it in line 7.
- Otherwise, SELECT eliminates the minimum from the subarray $A[p:r]$ and iterates to find the $(i-1)st$ element in $A[p+1:r]$ (lines 9-10).
- After line 10, if the procedure is not returned, $A[p:r]$ is evenly divisible by 5.

```
1   while (r − p + 1) mod 5 ≠ 0
2       for j = p + 1 to r                    // put the minimum into A[p]
3           if A[p] > A[j]
4               exchange A[p] with A[j]
5       // If we want the minimum of A[p : r], we're done.
6       if i == 1
7           return A[p]
8       // Otherwise, we want the(i − 1)st element of A[p + 1 : r].
9       p = p + 1
10      i = i − 1
```

SELECT: explanation

- In the next part, we divide the elements into groups of 5 elements each.
  - number of groups: $g = (r - p + 1)/5$.
  - The first group: $\langle A[p], A[p+g], A[p+2g], A[p+3g], A[p+4g] \rangle$
  - The second group: $\langle A[p+1], A[p+g+1], A[p+2g+1], A[p+3g+1], A[p+4g+1] \rangle$
  - The last group: $\langle A[p+g-1], A[p+2g-1], A[p+3g-1], A[p+4g-1], A[r] \rangle$
    - $r = p + 5g - 1$

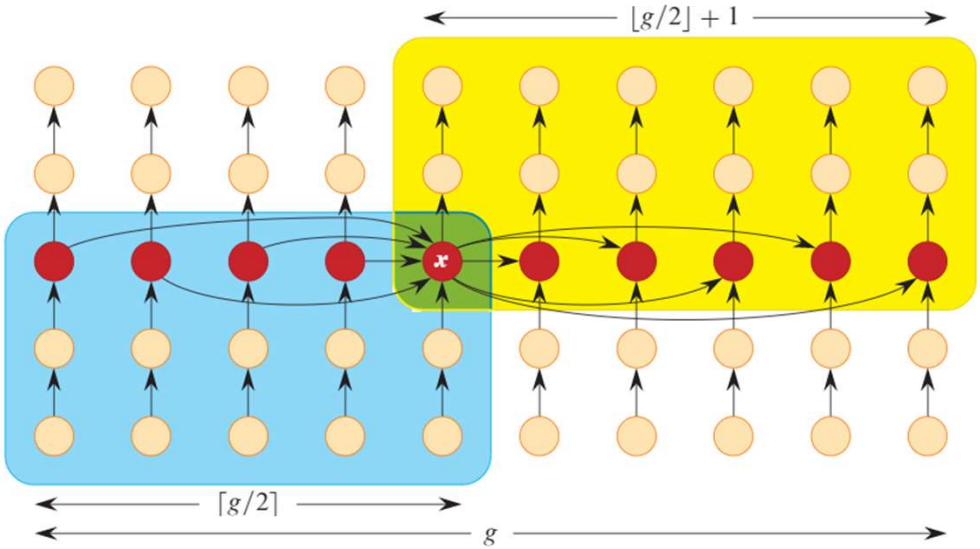


SELECT: explanation

- In lines 11-13, we sort each group using insertion sort.
  - After sorting, for $j = p, p+1, \ldots, p+g-1$, we have $A[j] \le A[j+g] \le A[j+2g] \le A[j+3g] \le A[j+4g]$.

```
11  g = (r − p + 1)/5                          // number of 5-element groups
12  for j = p to p + g − 1                     // sort each group
13      sort ⟨A[j], A[j + g], A[j + 2g], A[j + 3g], A[j + 4g]⟩ in place
```

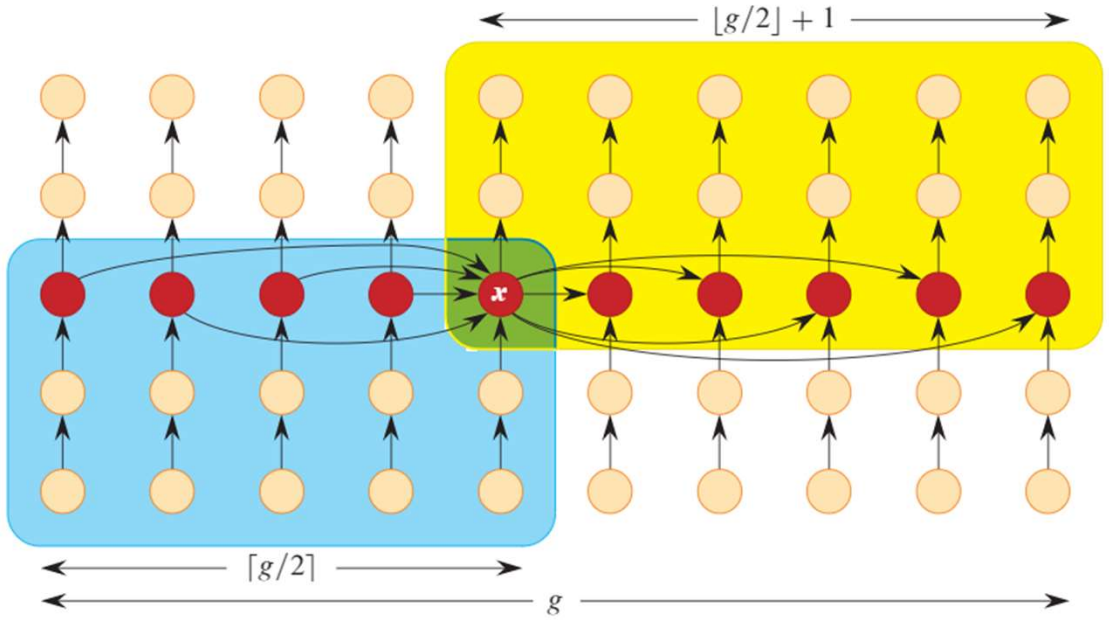
  - The median of each 5-element group is $A[j+2g]$, and thus all the 5-element medians, shown in red, lie in the range $A[p+2g : p+3g-1]$.



SELECT: explanation

- Line 16 determines the pivot x by recursively calling SELECT to find the median (specifically, the $\lceil g/2 \rceil$th smallest) of the g group medians.
- The "median of medians" is used as a pivot to partition the elements using PARTITION-AROUND (line 17).

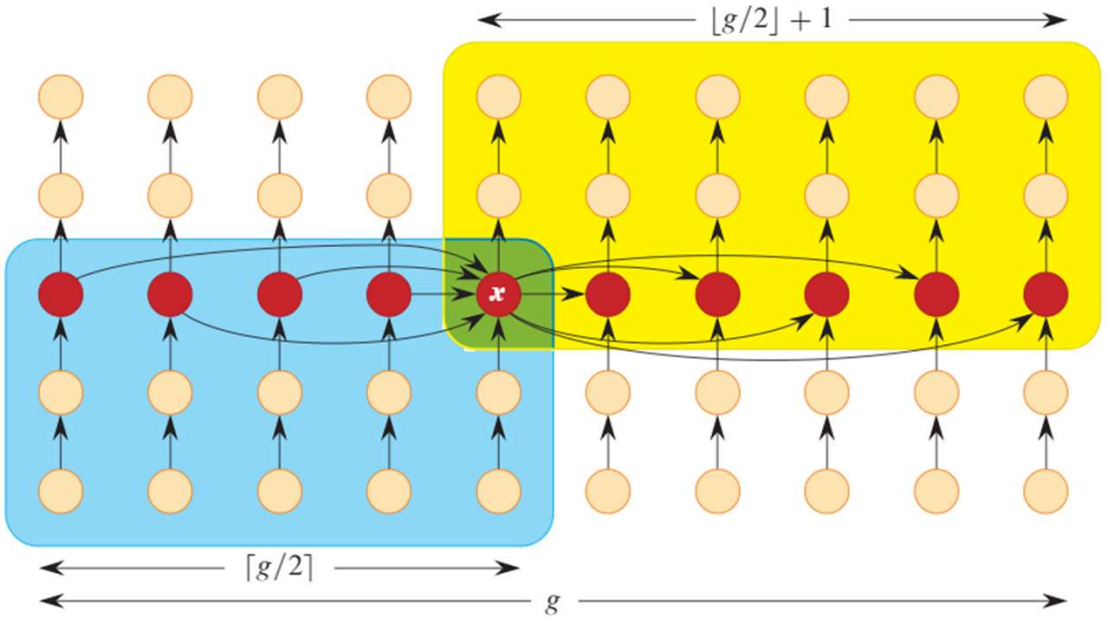
```
16  x = SELECT(A, p + 2g, p + 3g − 1, ⌈g/2⌉)
17  q = PARTITION-AROUND(A, p, r, x)   // partition around the pivot
```



SELECT: explanation

- The rest is similar to RANDOMIZED-SELECT

```
19  k = q − p + 1
20  if i == k
21        return A[q]                          // the pivot value is the answer
22  elseif i < k
23        return SELECT(A, p, q − 1, i)
24  else return SELECT(A, q + 1, r, i − k)
```



SELECT: running time analysis

- Theorem 9.3: The running time of SELECT on an input of n elements is $\Theta(n)$.

- We first determine an upper bound on the time spent outside the recursive calls in lines 16, 23, and 24.

- The while loop in lines 1-10 executes 0 to 4 times, which is $O(1)$ times.
- Since the dominant time within the loop is the computation of the minimum in lines 2-4, which takes $\Theta(n)$.
- Thus, the lines 1-10 execute in $O(n)$ time.
- The sorting of 5-element groups in lines 11-13 takes $\Theta(n)$ time
  - Each 5-element group takes $\Theta(1)$ time to sort.
  - There are g elements to sort, where $n/5 - 1 < g \leq n/5$.
- The running time to partition in line 17 is $\Theta(n)$.
- Total time spent outside of the recursive calls: $O(n) + \Theta(n) + \Theta(n) + \Theta(1) = \Theta(n)$.

SELECT: running time analysis

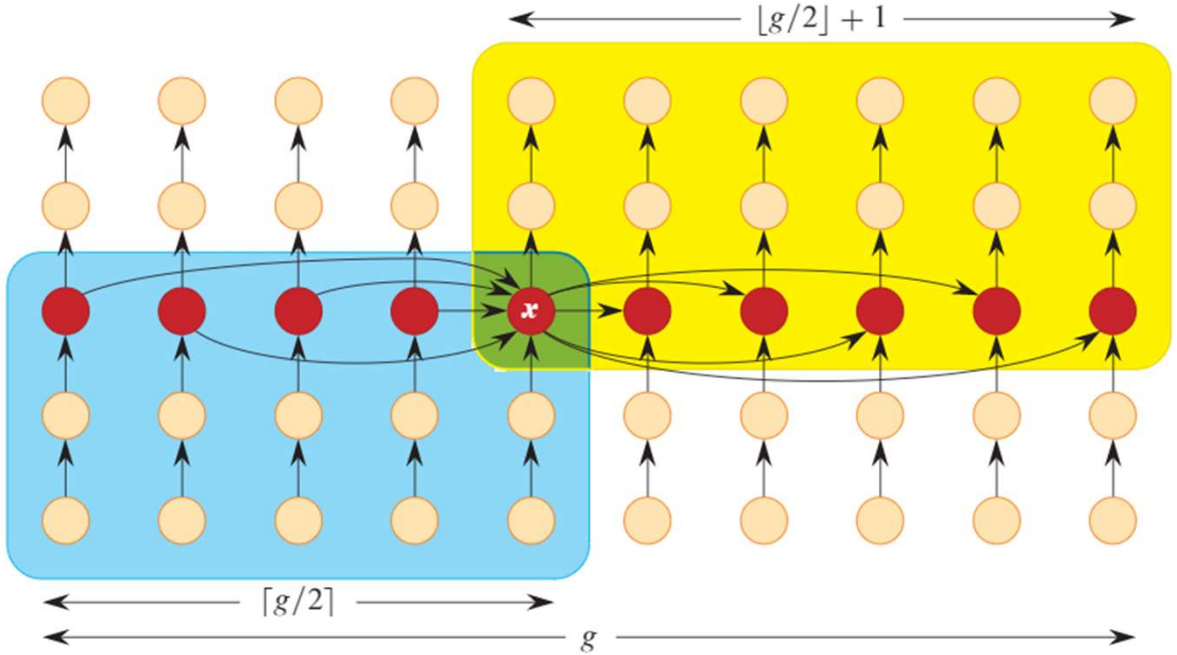
- Now we determine the running time for the recursive calls.

- The recursive call to find the pivot in line 16 takes $T(g) \leq T(n/5)$ time, since $g \leq n/5$, and $T(n)$ monotonically increases.

- Of the two recursive calls in lines 23 and 24, at most one is executed. Either way, the number of elements in the recursive call turns out to be at most $7n/10$, and hence the worst-case cost for lines 23 and 24 is at most $T(7n/10)$.

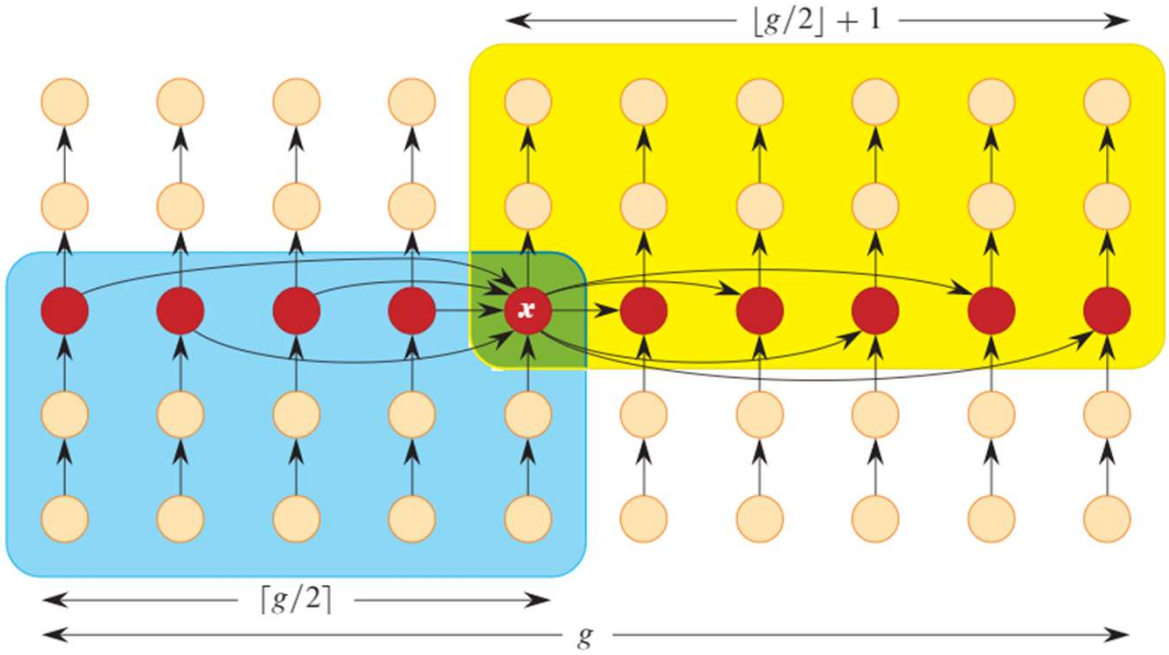
SELECT: running time analysis

- Why $7n/10$? Let us look at the figure.
- There are $n/5$ groups of 5 elements, with each group shown as a column sorted from bottom to top.
- The columns are ordered from left to right with groups to the left of x's group having group median less than x, and groups to the right of x's group having group median greater than x.



SELECT: running time analysis

- The yellow region contains elements that are greater than or equal to x.
- The blue region contains elements that are less than or equal to x.
- These two regions each contain at least $3g/2$ elements.
- Thus, partitioning elements using pivot x will remove $3g/2$ elements.
- The low side or the high side of the partition can contain at most $5g - \frac{3g}{2} = \frac{7g}{2} \leq 7n/10$ elements.



SELECT: running time analysis

- The recurrence for the worst-case running time of SELECT
- $T(n) \leq T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + \Theta(n)$.

- By substitution method, we can show that $T(n) = O(n)$. Assuming $n \geq 5$,

$$\begin{aligned} T(n) &\leq c(n/5) + c(7n/10) + \Theta(n) \\ &\leq 9cn/10 + \Theta(n) \\ &= cn - cn/10 + \Theta(n) \\ &\leq cn \end{aligned}$$

  - If c is chosen large enough that c/10 dominates the upper-bound constant hidden by $\Theta(n)$.
- When $n \leq 4$, we can pick c large enough that $T(n) \leq cn$ for all $n \leq 4$.
- Thus, the running time of SELECT is $O(n)$.
- Since line 13 takes $\Theta(n)$ time, the total time is $\Theta(n)$.

Selection in Linear Time: Summary

- The selection algorithms in this chapter are not subject to $\Omega(n \log n)$ lower bound, because they manage to solve the selection problem without sorting the elements.
- Thus, solving the selection problem by sorting and indexing is asymptotically inefficient in the comparison model.

End of Class

Questions?

Instructor office: AS-1013

Email: jso1@sogang.ac.kr