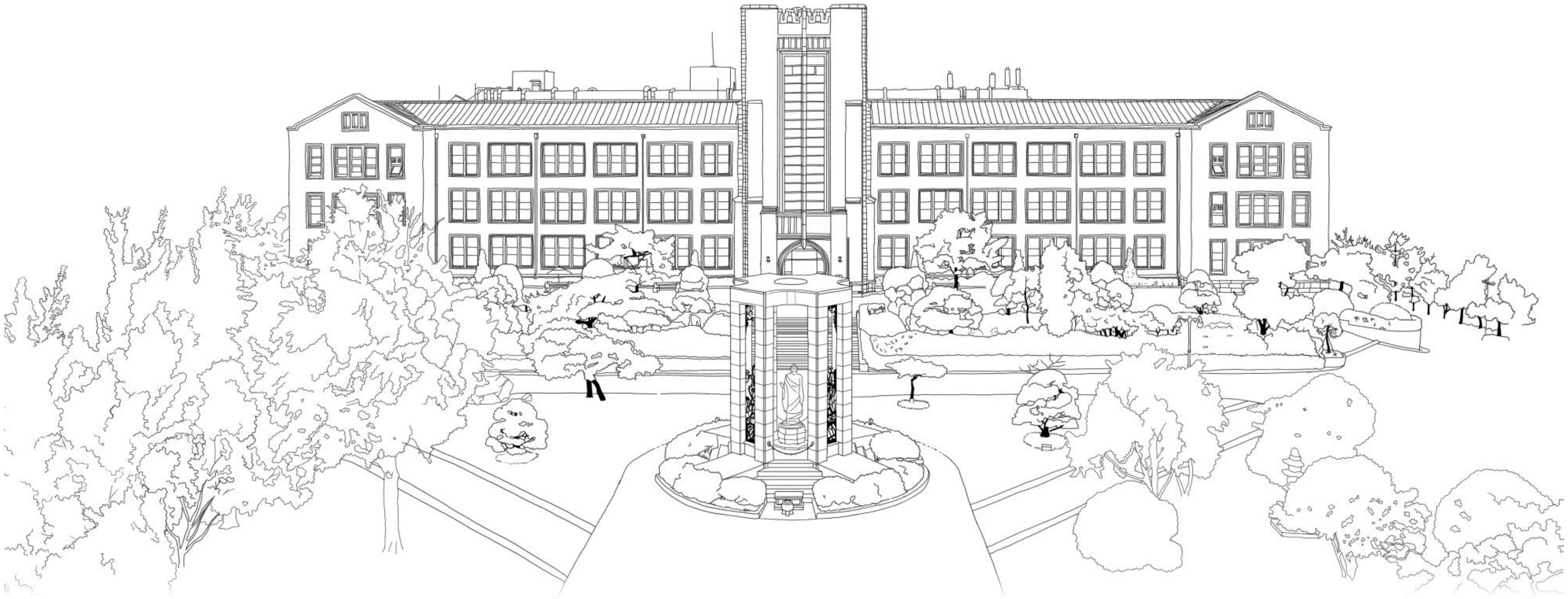


Introduction to Algorithms

DongYeon Kim
Department of Multimedia Engineering
Dongguk University

Contents

❖ Chapter.2 Sorting and Order Statistics

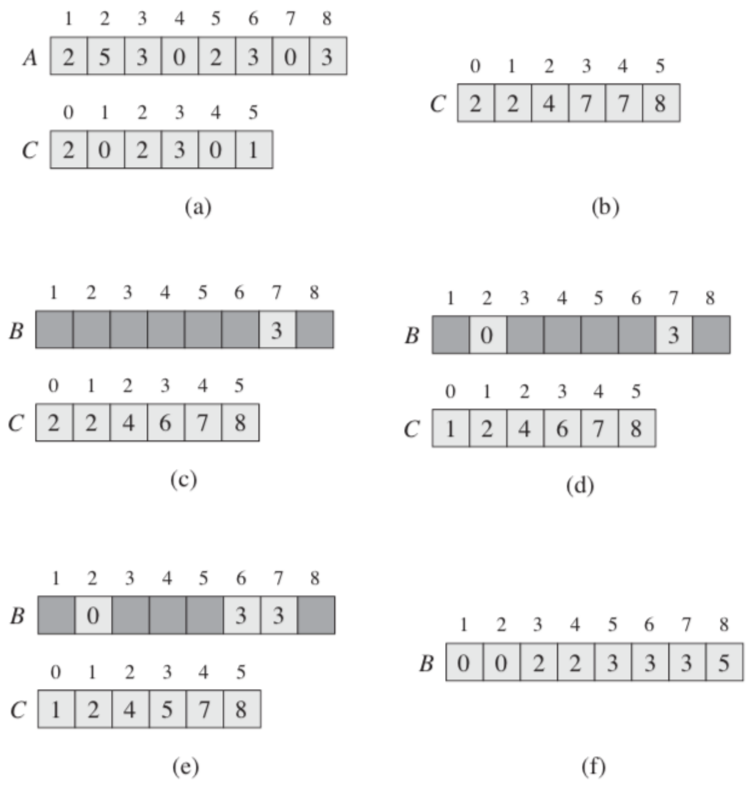
❖ Counting sort

- Introduction
  - ➢ Sort the number according to keys that are small positive integer
  - ➢ Assume that each of the input elements is an integer in the range 0 to k
    - ✓ The range of input elements are small
  - ➢ Running time of Counting sort: $\theta(n)$
    - ✓ Linear time algorithm

- Definition
  - ➢ For input element x, the number of elements less than x are computed
  - ➢ Use this information to put x directly in its position in the output array
    - ✓ ex) if 17 elements are less than x, then x belongs in output position 18
  - ✓ Modify this method to handle the situation in which several elements have the same value

❖ Counting sort

- Definition
  - ➢ A : input array B : sorted output array C : temporary working storage
  - ➢ Each element of A is a nonnegative integer no larger than $k = 5$



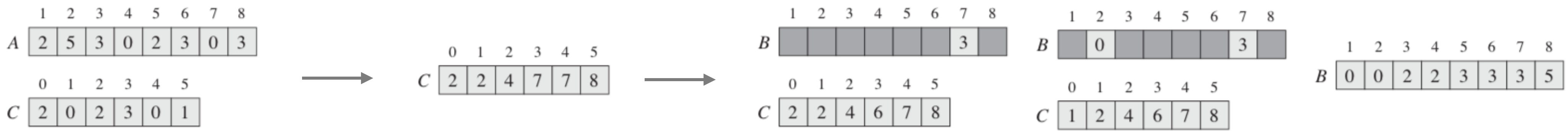
- ➢ (a) : $C[i]$ holds the number of input elements equal to i for each integer $i = 0, 1, \dots, k$
- ➢ (b) : accumulate the array C in step (a)
- ➢ (b) : the array C represents how many input elements are less than or equal to i
- ➢ $(c \sim e)$: the array C represents the proper position index for each value $(1 \sim k)$
- ➢ $(c \sim e)$: place input elements of A to output array B according to the index value of array C
- ➢ (f) : the sorted output array

❖ Counting sort

- Pseudocode
  - ➢ A : input array B : sorted output array C : temporary working storage
  - ➢ Each element of A is a nonnegative integer no larger than $k = 5$

```
COUNTING-SORT(A, B, k)
1   let C[0..k] be a new array
2   for i = 0 to k
3       C[i] = 0
4   for j = 1 to A.length
5       C[A[j]] = C[A[j]] + 1
6   // C[i] now contains the number of elements equal to i.
7   for i = 1 to k
8       C[i] = C[i] + C[i − 1]
9   // C[i] now contains the number of elements less than or equal to i.
10  for j = A.length downto 1
11      B[C[A[j]]] = A[j]
12      C[A[j]] = C[A[j]] − 1
```

- ➢ $line$ 2~3 : initialize the array C to all zeros
- ➢ $line$ 4~5 : count the number of each number (1~k) from input array A
- ➢ $line$ 7~8 : accumulate the array C
- ➢ $line$ 10~12 : place each element $A[j]$ into its correct sorted position in the output B



$line$ 4~5 $line$ 7~8 $line$ 10~12

Sorting in Linear Time

❖ Counting sort

- Running Time
  - ➢ For input size n, the running time of counting sort is $\theta(n)$

```
COUNTING-SORT(A, B, k)
1   let C[0..k] be a new array
2   for i = 0 to k
3       C[i] = 0
4   for j = 1 to A.length
5       C[A[j]] = C[A[j]] + 1
6   // C[i] now contains the number of elements equal to i.
7   for i = 1 to k
8       C[i] = C[i] + C[i - 1]
9   // C[i] now contains the number of elements less than or equal to i.
10  for j = A.length downto 1
11      B[C[A[j]]] = A[j]
12      C[A[j]] = C[A[j]] - 1
```

- ➢ $line\ 2\sim3$: takes $\theta(k)$
- ➢ $line\ 4\sim5$: takes $\theta(n)$
- ➢ $line\ 7\sim8$: takes $\theta(k)$
- ➢ $line\ 10\sim12$: takes $\theta(n)$

→ the overall running time is $\theta(k+n)$

→ we usually use counting sort when $k = O(n)$, because the performance of counting sort is up to constant k

❖ Counting sort

- Conclusion
  - ➢ Counting sort beats the $\Omega(n \log n)$ which is the lower bound of comparison sort model
    - ✓ Counting sort is not a comparison sort
    - ✓ no comparisons between input elements occur anywhere in the code
  - ➢ Counting sort uses the actual values of the elements to index into an array
  - ➢ It is stable algorithm
    - ✓ Numbers with the same value appear in the output array in the same order as they do in the input array
    - ✓ This property is used in radix sort

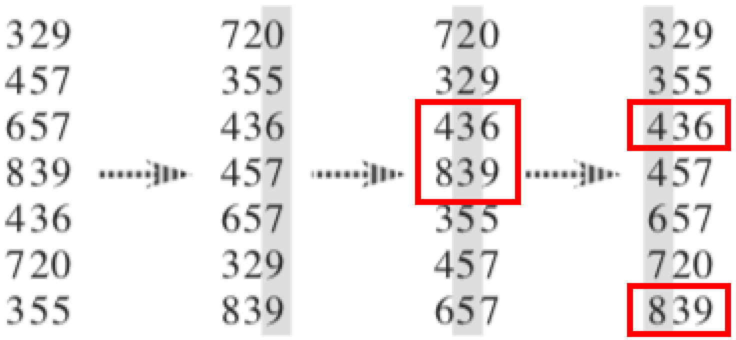
	1	2	3	4	5	6	7	8
A	2	5	3	0	2	3	0	3

	1	2	3	4	5	6	7	8
B	0	0	2	2	3	3	3	5

Sorting in Linear Time

❖ Radix sort

- Definition & Pseudocode
  - ➢ for decimal digit, each column uses only 10 places
  - ➢ In generally, d –digit number occupy a field of d column
  - ➢ Radix sort solves the problem counterintuitively by sorting on the least significant digit first
  - ➢ In order for radix sort to work correctly, the digit sorts must be stable

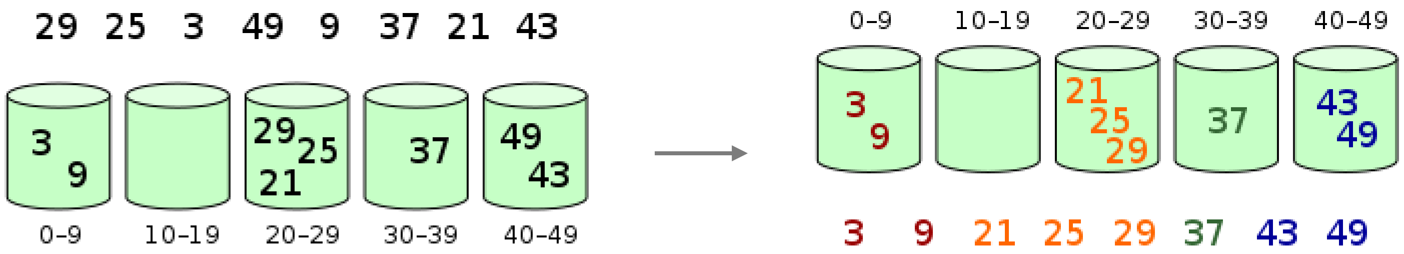



```
RADIX-SORT(A, d)
for i = 1 to d
    use a stable sort to sort array A on digit i
```

❖ Bucket sort

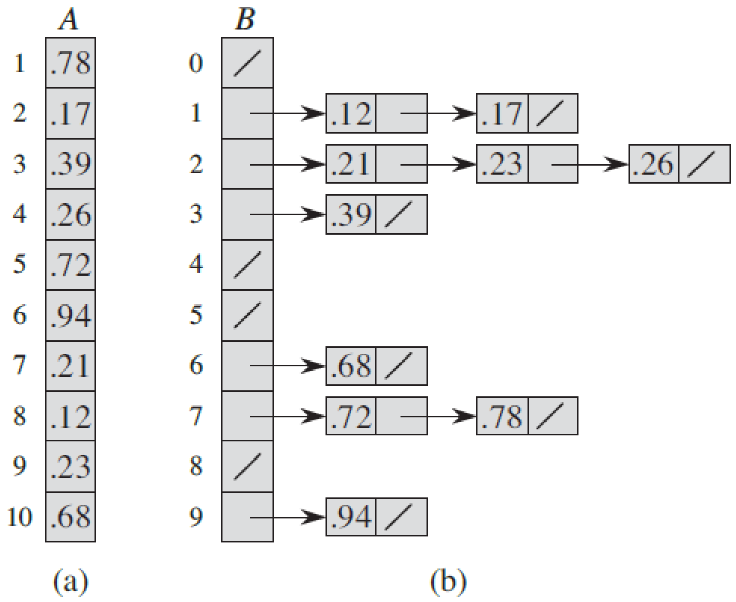
- Introduction
  - ➢ assume that the input is drawn from a uniform distribution and has an average-case running time of $O(n)$
  - ➢ assume that the input is generated by a random process that distributes elements uniformly and independently over the interval [0, 1)

- Definition
  - ➢ Divide the interval [0,1) into equal-sized subintervals (buckets)
  - ➢ Distributes the n input numbers into the buckets
  - ➢ Sort the numbers in each bucket → go through the bucket in order → listing the elements in each



❖ Bucket sort

- Definition
  - ➢ A : input array B : sorted linked-list (buckets) the number of buckets(n) : 10



- ➢ Bucket i ($B[i]$) holds values in the interval $[i/10, (i+1)/10)$
- ➢ The input elements are distributed according to the interval
- ➢ The sorted output consists of a concatenation in order of the list $B[0], B[1], \dots, B[9]$

Sorting in Linear Time

❖ Bucket sort

- Pseudocode
  - A : input array B : sorted linked list (buckets)   the number of buckets(n) : 10
  - Assume that Each element $A[i]$ in the array satisfies $0 \le A[i] \le 1$

```
BUCKET-SORT(A)
1  let B[0..n − 1] be a new array
2  n = A.length
3  for i = 0 to n − 1
4      make B[i] an empty list
5  for i = 1 to n
6      insert A[i] into list B[⌊nA[i]⌋]
7  for i = 0 to n − 1
8      sort list B[i] with insertion sort
9  concatenate the lists B[0], B[1], ..., B[n − 1] together in order
```

  - $line$ 3~4 : initialize the bucket
  - $line$ 5~6 : distributes the n input numbers into the buckets
  - $line$ 7~8 : sort the numbers in each bucket
  - $line$ 9 : concatenate the bucket in order

  - Consider two elements $A[i]$ and $A[j]$, assume that $A[i] \le A[j] \rightarrow \lfloor nA[i] \rfloor \le \lfloor nA[j] \rfloor$
  - If $\lfloor nA[i] \rfloor = \lfloor nA[j] \rfloor$, the elements $A[i]$ and $A[j]$ in the same bucket → $line$ 8 sort the elements
  - If $\lfloor nA[i] \rfloor < \lfloor nA[j] \rfloor$, the elements $A[i]$ and $A[j]$ in the different bucket → $line$ 9 sort the elements

    → bucket sort works correctly

Sorting in Linear Time

❖ Bucket sort

- Running Time

```
BUCKET-SORT(A)
1  let B[0..n − 1] be a new array
2  n = A.length
3  for i = 0 to n − 1
4       make B[i] an empty list
5  for i = 1 to n
6       insert A[i] into list B[⌊n A[i]⌋]
7  for i = 0 to n − 1
8       sort list B[i] with insertion sort
9  concatenate the lists B[0], B[1], ..., B[n − 1] together in order
```

➢ All lines except $line$ 8 take $O(n)$ time in the worst case

➢ we need to analyze the total time of n calls to insertion sort in $line$ 8

➢ Let n_i be the number of elements placed in bucket $B[i]$

$$T(n) = \Theta(n) + \sum_{i=0}^{n-1} O(n_i^2) \longrightarrow$$

$$\begin{aligned} \mathrm{E}[T(n)] &= \mathrm{E}\left[\Theta(n) + \sum_{i=0}^{n-1} O(n_i^2)\right] \\ &= \Theta(n) + \sum_{i=0}^{n-1} \mathrm{E}[O(n_i^2)] \\ &= \Theta(n) + \sum_{i=0}^{n-1} O(\mathrm{E}[n_i^2]) \end{aligned}$$

$$\xrightarrow{\quad} \quad \mathrm{E}[n_i^2] = 2 - 1/n$$

$$\Theta(n) + n \cdot O(2 - 1/n)$$

$$= \Theta(n)$$

Contents

❖ Chapter.2 Sorting and Order Statistics

❖ Minimum and Maximum

- Introduction
  - ➢ *Order statistics* : the ith order statistic of a set of n elements is the ith smallest element
  - ➢ *Minimum* : the first order statistic ($i = 1$)
  - ➢ *Maximum* : the last order statistic ($i = n$)
  - ➢ *median* : "halfway point" of the set
    - ✓ n is odd – the median is unique $i = (n + 1)/2$
    - ✓ n is even – there are two medians
      - i. $i = \lfloor (n + 1)/2 \rfloor$: the lower median
      - ii. $i = \lceil (n + 1)/2 \rceil$: the upper median
  - ➢ This chapter address the problem of selecting ith statistic from a set of n distinct numbers

    Input: A set A of n (distinct) numbers and an integer i, with $1 \le i \le n$.

    Output: The element $x \in A$ that is larger than exactly $i - 1$ other elements of A.
  - ➢ We can solve the selection problem in $O(n\log n)$ times
    - ✓ Sort the numbers using heap or merge sort and then simply index ith element
    - ✓ This chapter present the faster algorithm

❖ Minimum and Maximum

- Pseudocode – Minimum
  - ➢ A : input array

```
MINIMUM(A)
min = A[1]
for i = 2 to A.length
    if min > A[i]
        min = A[i]
return min
```

- ➢ Easily obtain an upper bound of n−1 comparisons
- ➢ Examine each element of the set and keep track of the smallest element

- ➢ The algorithm $MINIMUM$ is optimal with respect to the number of comparisons performed
  - ✓ Examine each element of the set and keep track of the smallest element

❖ Minimum and Maximum

- **Simultaneous minimum and maximum**
  - ➢ to determine both the minimum and the maximum of n elements : $\theta(n)$ comparisons
  - ➢ Asymptotically optimal : simply find the minimum and maximum independently
    - ✓ Using $n-1$ comparisons for each → total $2n-2$ comparisons
  - ➢ Can find both minimum and the maximum value using at most $3[n/2]$
    - ✓ Compare pairs of elements from the input : $n/2$
    - ✓ Compare lower element of pairs with minimum : $n/2$
    - ✓ Compare larger element of pairs with maximum : $n/2$

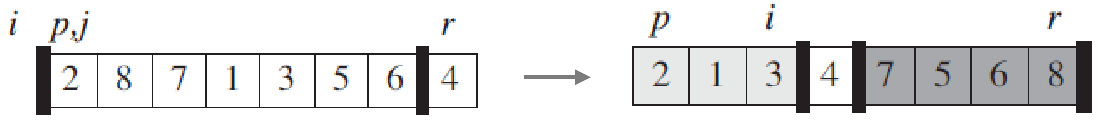
    → total $3n/2$, in even case $3[n/2]$

❖ Selection in expected Linear time

- Introduction
  - ➢ The general selection problem appears more difficult than the simple problem of finding a minimum
  - ➢ Asymptotic running time is $\theta(n)$ → same with finding minimum
  - ➢ Use divide-and-conquer algorithm for the selection problem
    - ✓ quicksort works recursively process both sides of the partition → $\theta(n \log n)$
    - ✓ $RANDOMIZED - SELECT$ works on only one side of the partition → $\theta(n)$

  - ➢ $RANDOMIZED - SELECT$ uses the procedure $RANDOMIZED - PARTITION$

```
RANDOMIZED-PARTITION(A, p, r)
i = RANDOM(p, r)
exchange A[r] with A[i]
return PARTITION(A, p, r)
```

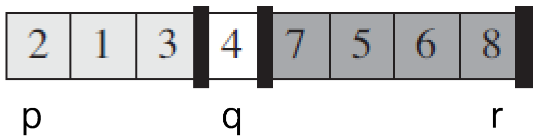


❖ Selection in expected Linear time

- Pseudocode
  - ➢ A : input array $A[p..r]$ i: the index number of elements to select

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

- ➢ *line* 1~2 : check for the base case of the recursion
  - ✓ consists of just one element
  - ✓ $i = 1$
  - ✓ Simply return $A[p]$
- ➢ *line* 3 : $RANDOMIZED - PARTITION$ partitions the array $A[p..r]$ into two subarrays $A[p..q-1]$ and $A[q+1..r]$
  - ✓ All elements of $A[p..q-1]$ is less or equal to $A[q]$
  - ✓ All elements of $A[q+1..r]$ is larger or equal to $A[q]$



❖ Selection in expected Linear time

- Pseudocode
  - ➢ A : input array $A[p..r]$ i: the index number of elements to select

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

✓ If $i > k$

i. elements lies on the high side

ii. The desire element is the $(i-k)$th smallest element of $A[q+1..r]$

➢ *line* 4 : computes the number of elements in the subarray $A[p..q]$

➢ *line* 5~6 : check whether $A[q]$ is the ith smallest element, if it is return $A[q]$

➢ *line* 7~9 : check which subarray contains ith element

✓ If $i < k$

i. elements lies on the low side

ii. Recursively select it from the $A[p..q-1]$

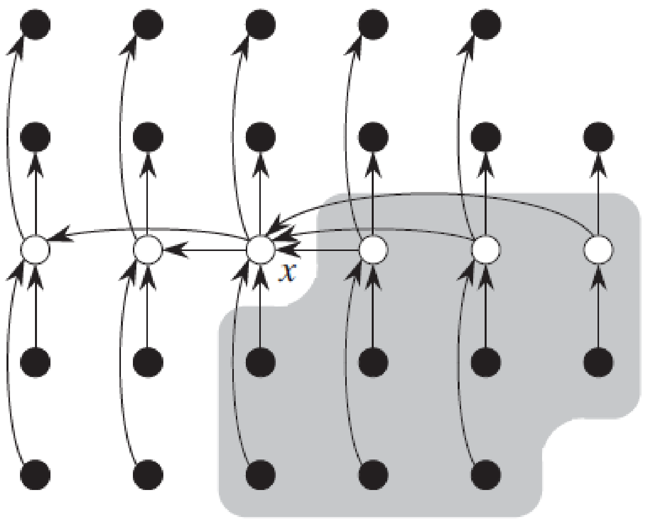
2	1	3	4	7	5	6	8
p			q				r

❖ Selection in worst-case Linear time

- Introduction
  - ➢ In this chapter, examine a selection algorithm in the worst case
  - ➢ Guarantee a good split upon partitioning the array

- definition

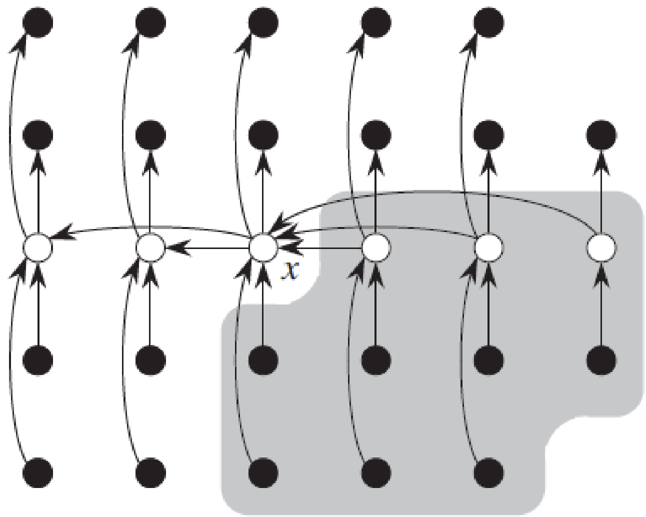
1. Divide the n elements of the input array into $\lfloor n/5 \rfloor$ groups of 5 elements each and at most one group made up of the remaining $n \bmod 5$ elements.
2. Find the median of each of the $\lceil n/5 \rceil$ groups by first insertion-sorting the elements of each group (of which there are at most 5) and then picking the median from the sorted list of group elements.
3. Use SELECT recursively to find the median x of the $\lceil n/5 \rceil$ medians found in step 2. (If there are an even number of medians, then by our convention, x is the lower median.)



❖ Selection in worst-case Linear time

- Definition

4. Partition the input array around the median-of-medians x using the modified version of PARTITION. Let k be one more than the number of elements on the low side of the partition, so that x is the kth smallest element and there are $n-k$ elements on the high side of the partition.
5. If $i = k$, then return x. Otherwise, use SELECT recursively to find the ith smallest element on the low side if $i < k$, or the $(i-k)$th smallest element on the high side if $i > k$.



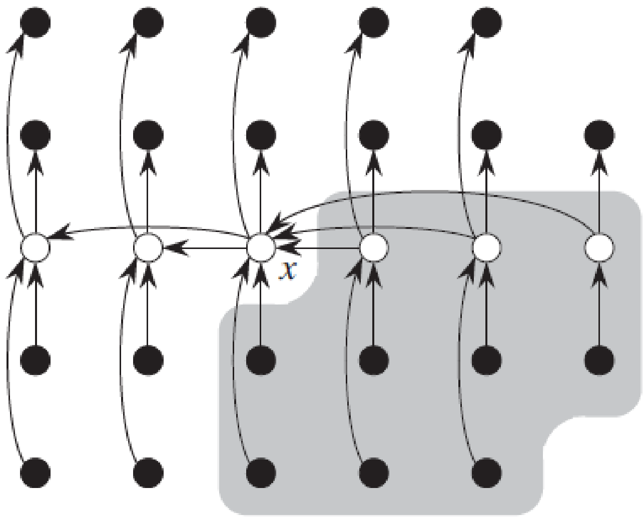
- Running time
  - Worst-case running time of the algorithm $SELECT$: $T(n)$
  - Step 1, 2, and 4 take $O(n)$ time
  - Step 3 takes $T(\lceil n/5 \rceil)$ time → use $SELECT$ function recursively
  - Step 5 takes at most $T(7n/10 + 6)$ time

❖ Selection in worst-case Linear time

- Running time
  - Step 5 takes at most $T(7n/10 + 6)$ time
    - ✓ Already know $3n/10 - 6$ elements are larger than median x

$$3\left(\underbrace{\left\lceil \frac{1}{2}\left\lceil \frac{n}{5} \right\rceil \right\rceil}_{\text{The number of groups larger than median}} \underbrace{- 2}_{\text{Except the median \& last group}}\right) \geq \frac{3n}{10} - 6$$

    - ✓ Only partition in $n - (3n/10 - 6) = 7n/10 + 6$ elements



  - $T(n) \leq T\left(\left\lceil \frac{n}{5} \right\rceil\right) + T\left(\frac{7n}{10} + 6\right) + O(n)$
  - Assume $T(n) \leq cn$ → linear time algorithm

$$\begin{aligned} T(n) &\leq c\lceil n/5 \rceil + c(7n/10 + 6) + an \\ &\leq cn/5 + c + 7cn/10 + 6c + an \\ &= 9cn/10 + 7c + an \\ &= cn + (-cn/10 + 7c + an), \end{aligned}$$

→ for large constant c, the worst-case running time of $SELECT$ is linear

Thank You!