CSC 317: Data Structures and Algorithm Analysis

Dilip Sarkar

Department of Computer Science
University of Miami

Outline I

- Order Statistics
- Minimum and maximum
- Selection in worst-case linear time

Order Statistics

- The *i*th **order statistic** of a set of n elements is the *i*th smallest element.
- Minimum is the 1st order statistic, because $i = 1$.
- Maximum is the *n*th order statistic, because $i = n$.
- A **median** is a "halfway point"
- If n is odd, the unique **median** is $(n+1)/2$
- If n is even, there are two **medians**
  - The *lower median* occurring at $n/2$, and
  - The *upper median* occurring at $(n/2+1)$.

The SELECTION problem.

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

Selection of min with $(n-1)$ comparisons

- **A[1..n]** is an array of n elements
- *min* holds the minimum value found, and
- *A.length* is the number of elements in A
- Line 1: initial **minimum** is **A[1]**
- Line 2: a for loop that executes (*A.length*) times
  - Why?
- Line 3: executes (*A.length* -1) times
- Line 4: in executes (*A.length* -1) times, in the worst-case
- Line 5: returns the minimum value found

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

- What is the complexity of the algorithm?
- $(n-1)$ comparisons are performed in the loop
- How do we prove correctness of the algorithm?
- Use **loop invariant**
- How many comparisons are necessary to find both **minimum** and **maximum**?

Selection of min and max with $3\lfloor n/2 \rfloor$ comparisons

- We can find the *mim* and *max* in $(2n-2)$ comparisons
- Modify MINIMUM(A)
- But we can find both **min** and **max** together
- The algorithm is shown to the right for odd n
- Line 02 and 03: initialize **min** and **max**
- Line 04: **for loop** is executed only $n/2$ times
- Lines 05 to 14: in the **for loop** only **three** comparisons are made
- Total comparisons are $3\lfloor n/2 \rfloor$
- How many comparisons are necessary to find kth element for $2 \leq k \leq (n-2)$?
- Can we do it in linear time?
- Yes, use **divide-and-conquer** algorithm

```
MINUMUN-MAXIMUM(A)
01 Assume number of elements n is odd
02     min = A[1]
03     max = A[1]
04     for i = 1 to (A.length/2)
05         if A[2*i] > A[2*i +1]
06             if min > A[2*i +1]
07                 min = A[2*i +1]
08             if max < A[2*i]
09                 max = A[2*i]
10         else {A[2*i] < A[2*i +1]}
11             if min > A[2*i]
12                 min = A[2*i]
13             if max < A[2*i+1]
14                 max = A[2*i+1]
15     return min max
```


Selection in worst-case linear time selection: An example

- Let us consider an example first
- suppose we have 75 unordered integers
- We want to find 17th integer
- Step 1: Arrange the elements into five rows

Selection in worst-case linear time selection : An example

- Let us consider an example first
- suppose we have 75 unordered integers
- We want to find 17th integer
- Arrange the elements into five rows

71	31	77	10	88	13	55	41	60	25	52	38	82	4	19
74	37	68	9	93	16	61	47	68	31	57	41	84	3	21
69	33	79	11	90	19	57	98	65	26	54	39	86	6	25
73	36	78	8	92	18	59	97	66	30	53	42	85	1	22
70	34	67	12	91	14	62	42	63	28	58	44	83	7	24

- Sort each column in increasing order, from top

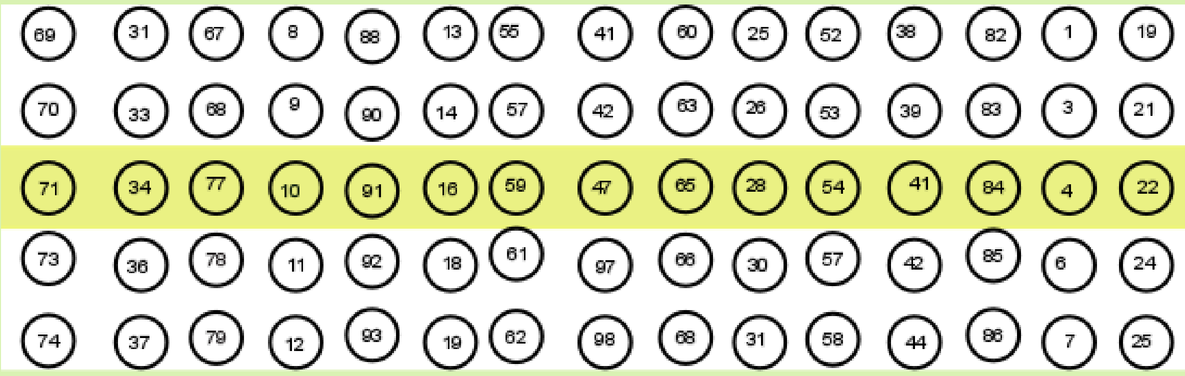
Selection in worst-case linear time selection : An example

- We get

69	31	67	8	88	13	55	41	60	25	52	38	82	1	19
70	33	68	9	90	14	57	42	63	26	53	39	83	3	21
71	34	77	10	91	16	59	47	65	28	54	41	84	4	22
73	36	78	11	92	18	61	97	66	30	57	42	85	6	24
74	37	79	12	93	19	62	98	68	31	58	44	86	7	25

- Third-row is the median of each column

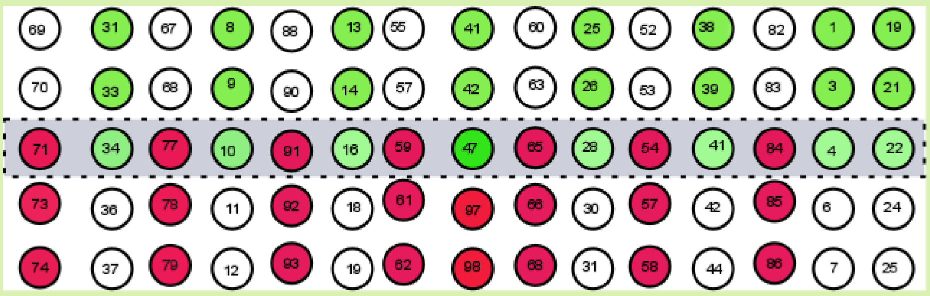
Selection in worst-case linear time selection : An example



- Third-row is the median of each column
- The algorithm works on the elements of the 3rd row
- Recursively, it finds the median of this row
- After finding the median value, the size of the problem is reduced

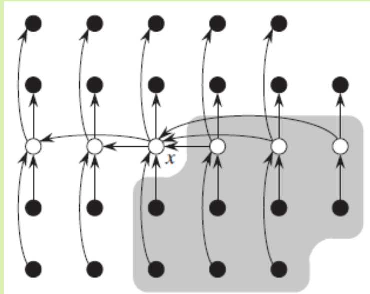
Minimum and maximum and Selection in worst-case linear time

Selection in worst-case linear time selection : An example



- The median value of the middle row is 47
- Seven values smaller than 47 are 34, 10, 16, 28, 41, 4, and 22
- Minimum # of values smaller than 47 are:
  - (Two values above 8 medians) + (7 medians smaller than 47)
  - 2*8 + 7 = 23 values in the green circles are smaller than 47
- Similarly, 23 values in the red circles are bigger than 47
- Since we want to find 17th integer, we can eliminate 23 values in the red circles,
- We now have to find the 17th integer from (75-23) = 52 integers

Selection in worst-case linear time selection : An example

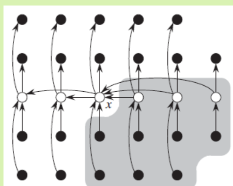


- Out of 75 integer, 23 are eliminated from selection at the first step!
- This is about 30%
- This elimination of about 30% continues, until
  - The desired 17th integer is found, or
  - The list is small enough to sort and
  - The desired 17th integer is directly selected

- The figure above is from the textbook
- It shows the same concept, but little abstract
- Lets put everything together as an algorithm

Selection in worst-case linear time selection

$\textsc{Select}(i\text{th}, n, a_1, a_2, \cdots, a_n)$
01 Divide $a_1, a_2, \cdots, a_n$ into groups of 5
02 Find medians $m_1, m_2, \cdots, m_{n/5}$ of $n/5$ groups
03 $x = \textsc{Select}(n/10, n/5, m_1, m_2, \cdots, m_{n/5})$
04 k = rank (x) { by counting number of elements }
05 { smaller than x; this takes linear time }
06 **if** $i == k$ { Case 1}
07 **return** x
08 **else if** $(i < k)$ {Case 2 }
09 $b[\,]$ = all items of $a[\,]$ less than x
10 **return** $\textsc{Select}(i\text{th}, k-1, b_1, b_2, \cdots, b_{k-1})$
11 **else if** $(i > k)$ {Case 3 }
12 $c[\,]$ = all items of $a[\,]$ greater than x
13 **return** $\textsc{Select}((i-k)\text{th}, n-k, c_1, c_2, \cdots, c_{n-k})$



Selection Algorithm Analysis

- $T(n)$ be the time complexity
- Lines 01, 02, 04, 07, 08, 11 takes $c_1 n = O(n)$ time
- Line 03 takes $T(n/5)$ time (approximately)
- Line 09 and 10 **OR** 12 and 13 are executed
- They take $T(7n/10)$ time, thus
- $T(n) \cong T(n/5) + T(7n/10) + cn$