Lecture 4: Order Statistics

Instructor: Saravanan Thirumuruganathan

Outline

1. Order Statistics
    - Min, Max
    - k^{th}-smallest and largest
    - Median
    - Mode and Majority

In-Class Quizzes

- **URL:** `http://m.socrative.com/`
- **Room Name: 4f2bb99e**

Order Statistics

- i^{th} Order Statistic of a set of n elements is the i^{th} smallest element
- **Selection** Problem
  - **Input:** A set A of n (distinct) numbers and an integer i with $1 \leq i \leq n$
  - **Output:** i^{th} smallest element in A
    - The element $x \in A$ that is larger than exactly $i - 1$ other elements of A
    - Select element with **rank** i

Popular Order Statistics

- $i = 1$
- $i = n$
- $i = \lfloor \frac{n+1}{2} \rfloor$ and $i = \lceil \frac{n+1}{2} \rceil$

Popular Order Statistics

- Minimum: $i = 1$
- Maximum: $i = n$
- Median: $i = \lfloor \frac{n+1}{2} \rfloor$ (lower) and $i = \lceil \frac{n+1}{2} \rceil$ (upper)

Selection Problem

- **Input:** A set A of n (distinct) numbers and an integer i with $1 \leq i \leq n$
- **Output:** i^{th} smallest element in A
- Naive Solution?

Selection Problem

- **Input:** A set A of n (distinct) numbers and an integer i with $1 \leq i \leq n$
- **Output:** i^{th} smallest element in A
- Naive Solution?
  - Sort A and pick $A[i]$
  - Time Complexity: $O(n \log n)$

Finding the Minimum

Finding the Minimum

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

Analysis:

Finding the Minimum

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

Analysis:

- Complexity Measure: Number of Comparisons

Finding the Minimum

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

Analysis:

- Complexity Measure: Number of Comparisons
- Number of Comparisons: $n - 1$
- Time Complexity: $O(n)$

Finding the Maximum

```
Maximum(A):
    max = A[1]
    for i = 2 to A.length
        if max < A[i]
            max = A[i]
    return max
```

Analysis:

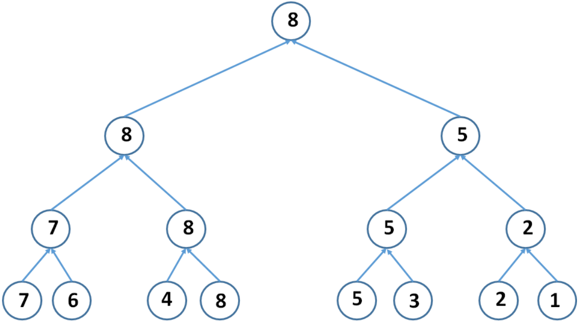
- Complexity Measure: Number of Comparisons
- Number of Comparisons: $n - 1$
- Time Complexity: $O(n)$

Recursive Maximum

Idea: Use Divide and Conquer to find Maximum

Recursive Maximum

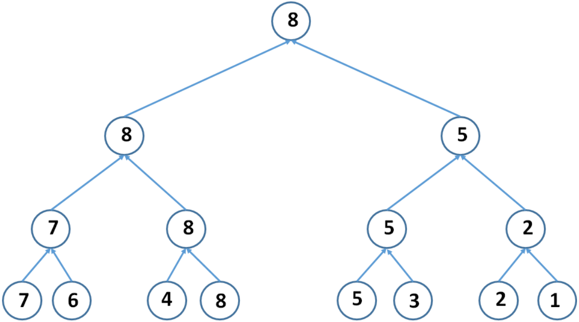
Idea: Use Divide and Conquer to find Maximum



Analysis:

Recursive Maximum

Idea: Use Divide and Conquer to find Maximum



Analysis:

- Recurrence Relation: $T(n) = 2T(\frac{n}{2}) + 1 = O(n)$
- Number of Comparisons: $n - 1$ (Intuition)

Simultaneous Maximum and Minimum

Aim: Find the maximum and minimum of array A

Simultaneous Maximum and Minimum

Aim: Find the maximum and minimum of array A

```
Minimum-Maximum(A):
    min = Minimum(A)
    max = Maximum(A)
    return min, max
```

Analysis:

Simultaneous Maximum and Minimum

Aim: Find the maximum and minimum of array A

```
Minimum-Maximum(A):
    min = Minimum(A)
    max = Maximum(A)
    return min, max
```

Analysis:

- Number of Comparisons: $(n-1)+(n-1)=2n-2$

Simultaneous Maximum and Minimum

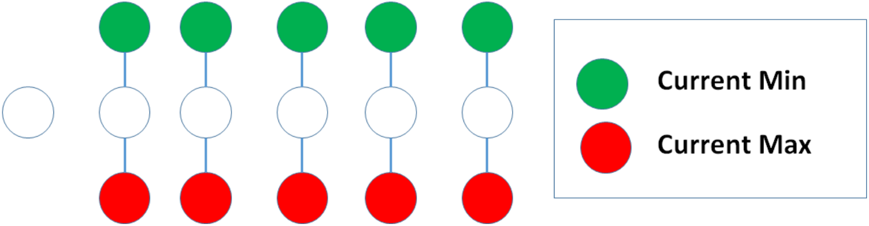
Aim: Find the maximum and minimum of array A

```
Minimum-Maximum(A):
    min = Minimum(A)
    max = Maximum(A)
    return min, max
```

Analysis:

- Number of Comparisons: $(n-1)+(n-1)=2n-2$
- Slightly better: $(n-1)+(n-2)=2n-3$ (for e.g., by swapping min with first element of array)

Simultaneous Maximum and Minimum - Visualization

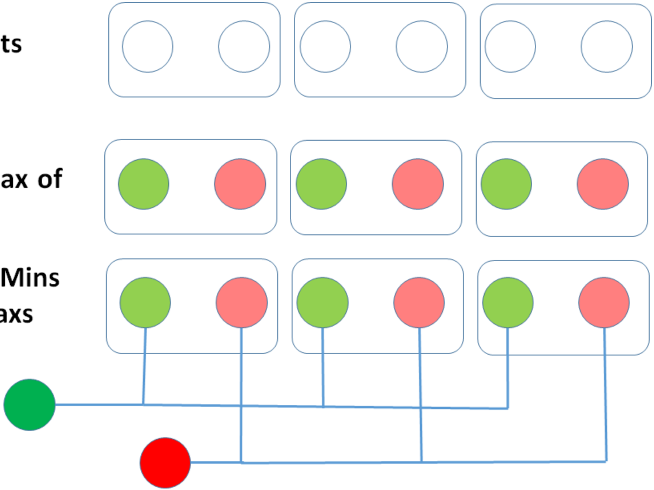


Simultaneous Maximum and Minimum - Better Algorithm

1. Pair Elements

2. Find Min/Max of Pairs

3. Find Min of Mins and Max of Maxs



Simultaneous Maximum and Minimum

Analysis:

Simultaneous Maximum and Minimum

Analysis:

- Number of Comparisons (approximate): Pairwise + Min of Mins + Max of Maxs

$$(\frac{n}{2}) + (\frac{n}{2}) + (\frac{n}{2}) = \frac{3n}{2}$$

Finding Second Largest Element - Naive Method

Finding Second Largest Element - Naive Method

```
Find-Second-Largest(A):
    max = Maximum(A)
    Swap A[n] with max
    secondMax = Maximum(A[1:n-1])
    return secondMax
```

Analysis:

Finding Second Largest Element - Naive Method

```
Find-Second-Largest(A):
    max = Maximum(A)
    Swap A[n] with max
    secondMax = Maximum(A[1:n-1])
    return secondMax
```

Analysis:

- $n-1$: for finding maximum
- $n-2$: for finding 2nd maximum
- $2n-3$: total

Finding Second Largest Element - Tournament Method

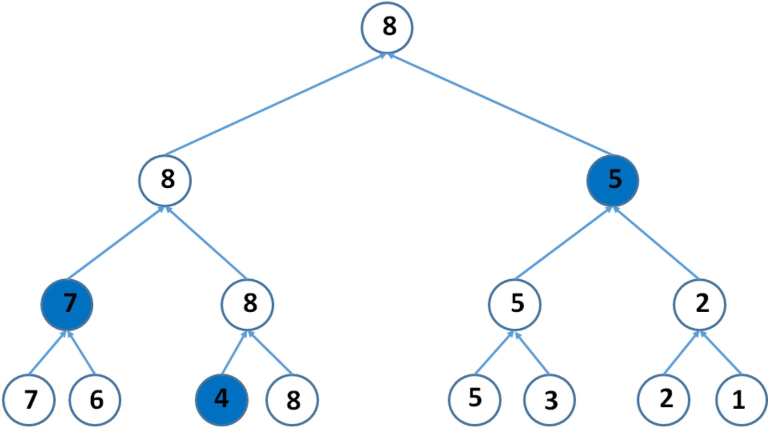
Finding Second Largest Element - Tournament Method

Observation:

- In a tournament, second best person could have only be defeated by the best person.
- It is not necessarily the other element in the final "match"

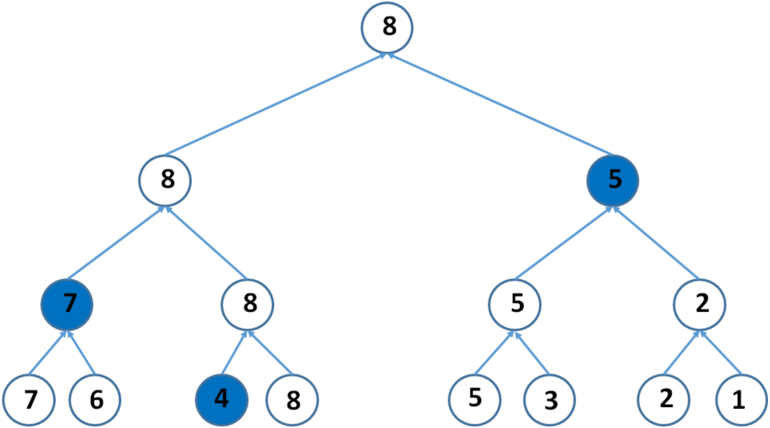
```
Find-Second-Largest(A):
    max = Recursive-Maximum(A)
    candidates = list of all elements of A that were
                    directly compared with max
    secondMax = Maximum(candidates)
    return secondMax
```

Finding Second Largest Element - Tournament Method



Analysis:

Finding Second Largest Element - Tournament Method



Analysis:

Number of Comparisons: $(n-1) + (\lceil \lg n \rceil - 1) = n + \lceil \lg n \rceil - 2$

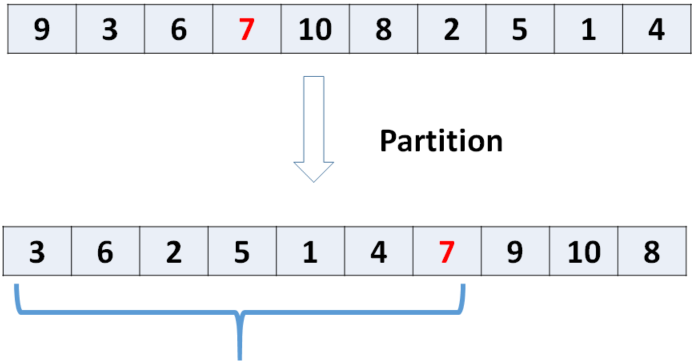
Selection Problem

- **Input:** A set A of n (distinct) numbers and an integer i with $1 \leq i \leq n$
- **Output:** i^{th} smallest element in A
- Naive Solution?
  - Sort A and pick $A[i]$
  - Time Complexity: $O(n \log n)$
- **Surprising Result**: Can be solved in $O(n)$ time!

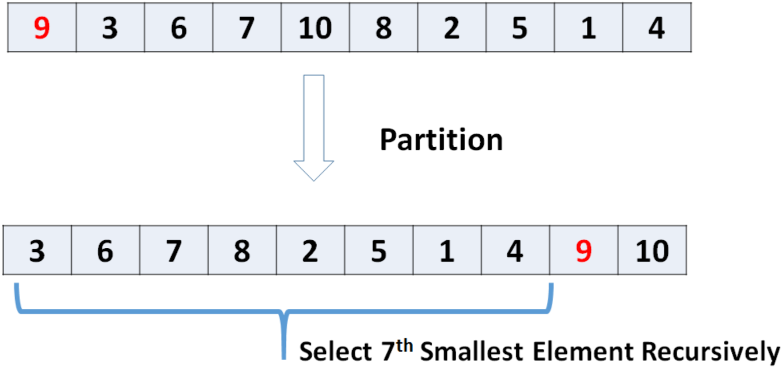
QuickSelect

- Divide and Conquer Strategy - Ideas?
- Called QuickSelect or Randomized-Select
- Invented by Tony Hoare
- Works excellent in practice

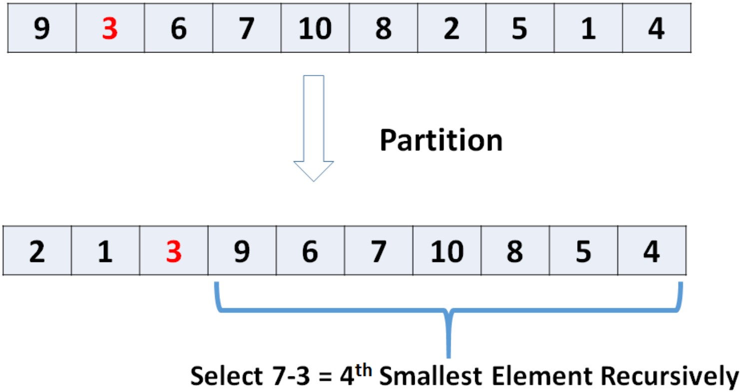
QuickSelect - Case 1



QuickSelect - Case 2



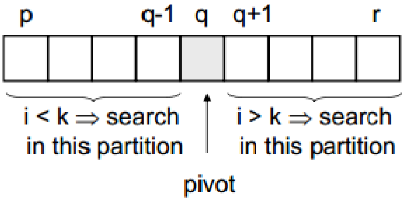
QuickSelect - Case 3



QuickSelect PseudoCode

```
Randomized-Select(A, p, r, i)
    if p == r:
        return A[p]
    q = Randomized-Partition(A, p, r)
    k = q - p + 1
    if i == k
        return A[q]
    elseif i < k
        return Randomized-Select(A, p, q-1, i)
    else
        return Randomized-Select(A, q+1, r, i-k)
```

QuickSelect - Intuition



QuickSelect - Analysis

- Recurrence Relation:

QuickSelect - Analysis

- Recurrence Relation:

$$T(n) = T(|L|) + n \text{ or } T(n) = T(|R|) + n$$

- Best Case:

QuickSelect - Analysis

- Recurrence Relation:

$$T(n) = T(|L|) + n \text{ or } T(n) = T(|R|) + n$$

- Best Case: $T(n) = T(\frac{n}{2}) + n \Rightarrow T(n) = O(n)$
- Worst Case:

QuickSelect - Analysis

- Recurrence Relation:

$$T(n) = T(|L|) + n \text{ or } T(n) = T(|R|) + n$$

- Best Case: $T(n) = T(\frac{n}{2}) + n \Rightarrow T(n) = O(n)$
- Worst Case: $T(n) = T(n-1) + n \Rightarrow T(n) = O(n^2)$
  - Worst than sorting !
- Lucky Case: (assume a 1:9 split)

QuickSelect - Analysis

- Recurrence Relation:

$$T(n) = T(|L|) + n \text{ or } T(n) = T(|R|) + n$$

- Best Case: $T(n) = T(\frac{n}{2}) + n \Rightarrow T(n) = O(n)$
- Worst Case: $T(n) = T(n-1) + n \Rightarrow T(n) = O(n^2)$
  - Worst than sorting !
- Lucky Case: (assume a 1:9 split)
  - $T(n) = T(\frac{9n}{10}) + n \Rightarrow T(n) = O(n)$

Similarities:

Similarities:

- Both invented by Tony Hoare
- Both use D&C and randomization
- Best and Average case behavior is good but has bad worst case behavior (same: $O(n^2)$)
- Works very well in practice

Differences:

QuickSelect and QuickSort

Similarities:

- Both invented by Tony Hoare
- Both use D&C and randomization
- Best and Average case behavior is good but has bad worst case behavior (same: $O(n^2)$)
- Works very well in practice

Differences:

- QuickSelect iterates on one partition only while QuickSort on both
- Objective: Sorting vs Selection

Median of Median Algorithm

- QuickSelect works well in Practice - linear expected time
- Worst case is worse than sorting $O(n^2)$
- Can we solve Selection problem in worst case Linear time?

Median of Median Algorithm

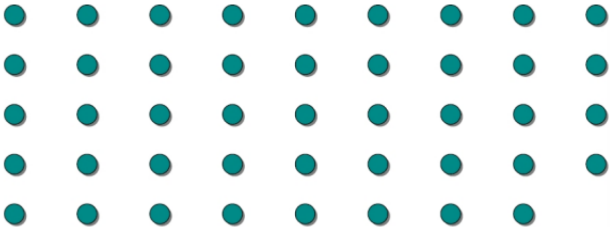
- QuickSelect works well in Practice - linear expected time
- Worst case is worse than sorting $O(n^2)$
- Can we solve Selection problem in worst case Linear time?
  - Yes!
  - Designed by Blum, Floyd, Pratt, Rivest & Tarjan in 1973
  - Basic Idea: Identify a good pivot so that partition is "balanced"
  - Aka "Median of Median" or "Worst case Linear time Order Statistics"

Median of Median Algorithm

SELECT(A, i, n):

1. Divide n elements into groups of 5. Last group might have less than 5 elements
2. Sort each group insertion sort. Find the median of each group
3. Use SELECT recursively to find median x of the $\lfloor \frac{n}{5} \rfloor$ medians
4. Partition A around x. Let position x be k
5. If $i = k$ then return x
6. If $i < k$, use SELECT recursively on the low side to find i^{th} smallest element
7. If $i > k$, use SELECT recursively on the high side to find $(i - k)^{th}$ smallest element

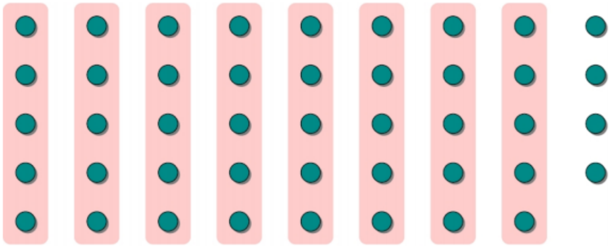
Median of Median - Visualization[1]



- Original input array A with n elements

[1] http://www.cs.gmu.edu/~ashehu/sites/default/files/cs583/ShehuLecture04.pdf

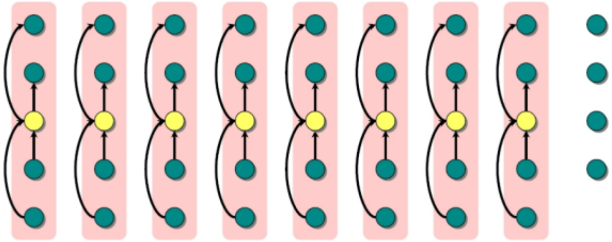
Median of Median - Visualization[2]



- Step 1: Divide A into groups of 5

[2] http://www.cs.gmu.edu/~ashehu/sites/default/files/cs583/ShehuLecture04.pdf

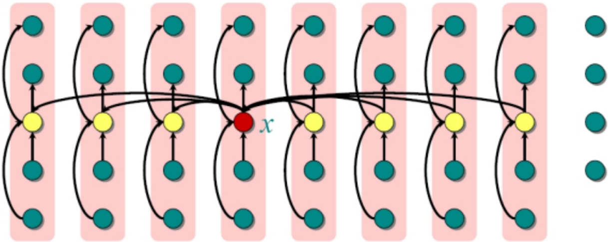
Median of Median - Visualization[3]



- Step 2: Sort each group by Insertion sort. Find its median.
- $A \to B$ means $A > B$

[3]http://www.cs.gmu.edu/~ashehu/sites/default/files/cs583/ShehuLecture04.pdf

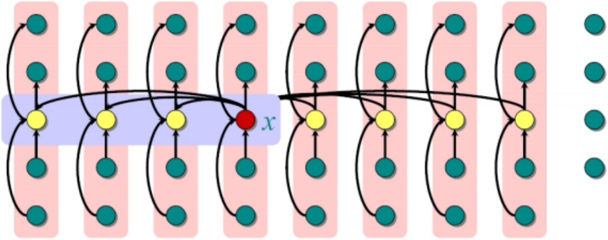
Median of Median - Visualization[4]



- Step 3: Use SELECT recursively to find median x of the $\lfloor \frac{n}{5} \rfloor$ medians.
- $A \rightarrow B$ means $A > B$

[4] http://www.cs.gmu.edu/~ashehu/sites/default/files/cs583/ShehuLecture04.pdf

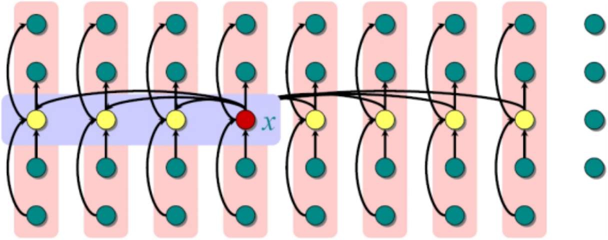
Median of Median - Visualization[5]



- Question: How many medians are less than x?

[5] http://www.cs.gmu.edu/~ashehu/sites/default/files/cs583/ShehuLecture04.pdf

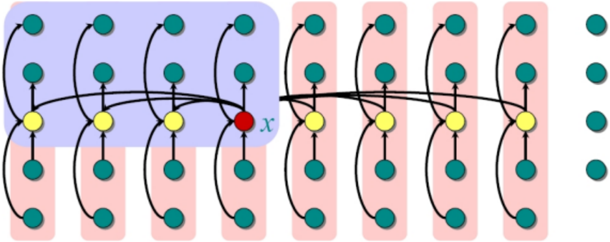
Median of Median - Visualization[5]



- Question: How many medians are less than x?
- At least half of the group medians are $\leq x$
- So at least $\lfloor \lfloor \left(\frac{n}{5}\right) /2 \rfloor \rfloor = \lfloor \frac{n}{10} \rfloor$

[5] http://www.cs.gmu.edu/~ashehu/sites/default/files/cs583/ShehuLecture04.pdf

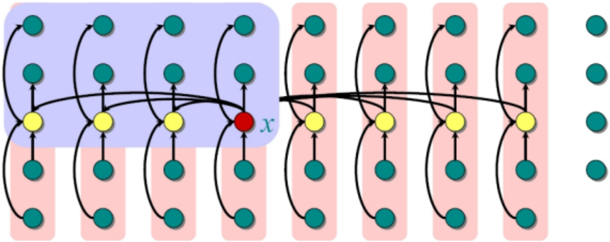
Median of Median - Visualization[6]



- Question: How many elements in A are **smaller** i.e. $\leq x$?

[6] http://www.cs.gmu.edu/~ashehu/sites/default/files/cs583/ShehuLecture04.pdf

Median of Median - Visualization[6]



- Question: How many elements in A are **smaller** i.e. $\leq x$?
- All elements smaller than the medians that were in turn smaller than x
  - $\lfloor \frac{n}{10} \rfloor$ medians were $\leq x$
  - So, $\lfloor \frac{3n}{10} \rfloor$ elements in A are $\leq x$

[6] http://www.cs.gmu.edu/~ashehu/sites/default/files/cs583/ShehuLecture04.pdf

Median of Median - Visualization[7]

One iteration on a randomized set of 100 elements from 0 to 99

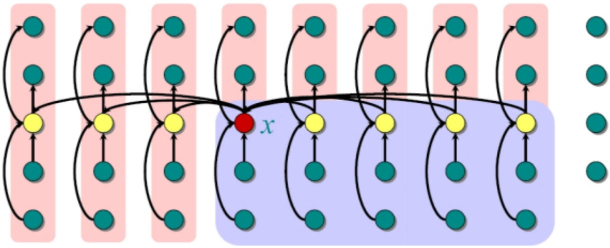
	12	15	11	2	9	5	0	7	3	21	44	40	1	18	20	32	19	35	37	39
	13	16	14	8	10	26	6	33	4	27	49	46	52	25	51	34	43	56	72	79
Medians	17	23	24	28	29	30	31	36	42	47	50	55	58	60	63	65	66	67	81	83
	22	45	38	53	61	41	62	82	54	48	59	57	71	78	64	80	70	76	85	87
	96	95	94	86	89	69	68	97	73	92	74	88	99	84	75	90	77	93	98	91

(red = "(one of the two possible) median of medians", gray = "number < red", white = "number > red")

- Note that some elements such as $22, 45, 38, 41$ are smaller than x
- But we don't count them as we are not sure

[7] http://en.wikipedia.org/wiki/Median_of_medians

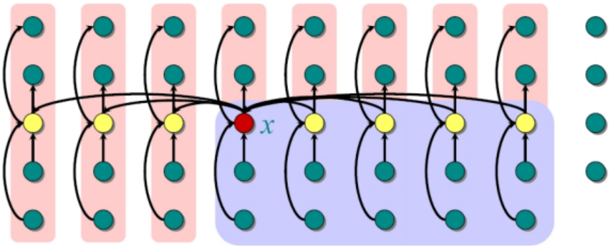
Median of Median - Visualization[8]



- Question: How many elements in A are **larger** i.e. $\geq x$?

[8] http://www.cs.gmu.edu/~ashehu/sites/default/files/cs583/ShehuLecture04.pdf

Median of Median - Visualization[8]



- Question: How many elements in A are **larger** i.e. $\geq x$?
- All elements larger than the medians that were in turn $\geq x$
  - $\lfloor \frac{n}{10} \rfloor$ medians were $\geq x$
  - So, $\lfloor \frac{3n}{10} \rfloor$ elements in A are $\geq x$

[8] `http://www.cs.gmu.edu/~ashehu/sites/default/files/cs583/ShehuLecture04.pdf`

Median of Median Algorithm - Analysis

SELECT(A, i, n):

1. Divide n elements into groups of 5. Last group might have less than 5 elements
2. Sort each group insertion sort. Find the median of each group
3. Use SELECT recursively to find median x of the $\lfloor \frac{n}{5} \rfloor$ medians
4. Partition A around x. Let position x be k
5. If $i = k$ then return x
6. If $i < k$, use SELECT recursively on the low side to find i^{th} smallest element
7. If $i > k$, use SELECT recursively on the high side to find $(i - k)^{th}$ smallest element

Median of Median Algorithm - Analysis

- Line 1:

Median of Median Algorithm - Analysis

- Line 1: $O(n)$
- Line 2:

Median of Median Algorithm - Analysis

- Line 1: $O(n)$
- Line 2: $\frac{n}{5} \times c_1 = O(n)$.
  - Sorting 5 elements requires constant c_1 comparisons
- Line 3:

Median of Median Algorithm - Analysis

- Line 1: $O(n)$
- Line 2: $\frac{n}{5} \times c_1 = O(n)$.
  - Sorting 5 elements requires constant c_1 comparisons
- Line 3: $T(\frac{n}{5})$
- Line 4:

Median of Median Algorithm - Analysis

- Line 1: $O(n)$
- Line 2: $\frac{n}{5} \times c_1 = O(n)$.
  - Sorting 5 elements requires constant c_1 comparisons
- Line 3: $T(\frac{n}{5})$
- Line 4: $T(n)$
- Line 5-7:

Median of Median Algorithm - Analysis

- Line 1: $O(n)$
- Line 2: $\frac{n}{5} \times c_1 = O(n)$.
    - Sorting 5 elements requires constant c_1 comparisons
- Line 3: $T(\frac{n}{5})$
- Line 4: $T(n)$
- Line 5-7: Worst Case Analysis : $T(\frac{3n}{4})$
- Size of largest partition: $(n - \lfloor \frac{3n}{10} \rfloor) = \lfloor \frac{7n}{10} \rfloor$
- But for $n \geq 50$, we have $\lfloor \frac{3n}{10} \rfloor \geq \frac{n}{4}$
- So, size of largest partition is $(n - \frac{n}{4}) = \frac{3n}{4}$
- **Final recurrence:** $T(n) = T(\frac{n}{5}) + O(n) + T(\frac{3n}{4})$
- **Solution :** $O(n)$

Median of Median - Conclusions

- Even though the algorithm is $O(n)$ (asymptotically linear), it has a huge *hidden* constant
- So, in practice, it runs much slower
- Use QuickSelect in practice
- To think about: What happens when we divide them into
  - Groups of 7?
  - Groups of 3?

Selection Problem - Applications

- Note: SELECT algorithm is a general purpose algorithm
  - Can solve Selection problem for any i (not just the median)
- How to find the i^{th} **largest**?

Selection Problem - Applications

- Note: SELECT algorithm is a general purpose algorithm
  - Can solve Selection problem for any i (not just the median)
- How to find the i^{th} **largest**?
  - Call SELECT to find $(n - i + 1)^{th}$ smallest element
- How to find median?

Selection Problem - Applications

- Note: SELECT algorithm is a general purpose algorithm
  - Can solve Selection problem for any i (not just the median)
- How to find the i^{th} **largest**?
  - Call SELECT to find $(n - i + 1)^{th}$ smallest element
- How to find median?
  - Call SELECT with $i = \lfloor \frac{n+1}{2} \rfloor$
  - By convention, lower median is chosen for even sized sets

Finding Majority

- **Majority:** The majority of a set of numbers is defined as a number that repeats at least $\frac{n}{2}$ times in the set.
- **Problem:** Find majority of a set A **if** one exists.
- **Naive Algorithm:**

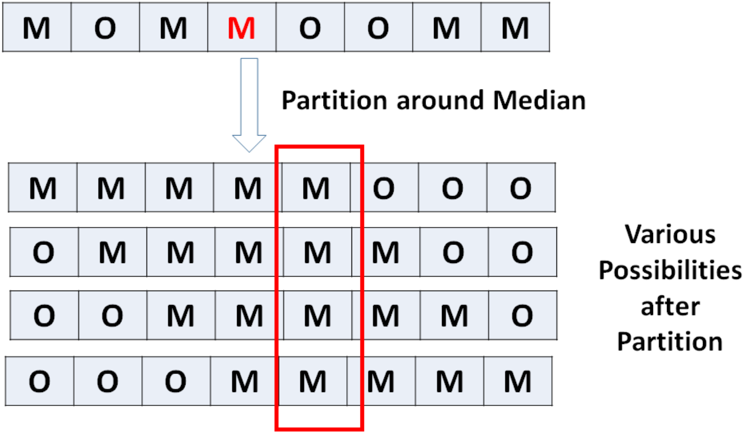
Finding Majority

- **Majority:** The majority of a set of numbers is defined as a number that repeats at least $\frac{n}{2}$ times in the set.
- **Problem:** Find majority of a set A **if** one exists.
- **Naive Algorithm:**
  - Sort and check if it has a majority element: $O(n \lg n)$
- **Better Algorithm:**

Finding Majority

- **Majority:** The majority of a set of numbers is defined as a number that repeats at least $\frac{n}{2}$ times in the set.
- **Problem:** Find majority of a set A **if** one exists.
- **Naive Algorithm:**
  - Sort and check if it has a majority element: $O(n \lg n)$
- **Better Algorithm:**
  - Use QuickSelect to find Median m
  - Partition A around median m
  - Verify if median is the majority element
  - Why does it work?
- **Time Complexity:** $O(n) + n + n = O(n)$

Finding Majority - Visualization



Finding Majority - Boyer-Moore Algorithm

- Boyer-Moore: one pass algorithm to find Majority candidate
  - Don't confuse it with Boyer-Moore String Search algorithm
- **Algorithm:**
  - Maintain current candidate (initially None) and a counter (initially 0).
  - Sweep the array from left to right
  - When we move the pointer forward over an element e:
    - If the counter is 0, we set the current candidate to e and we set the counter to 1.
    - If the counter is not 0, we increment the counter if e is the current candidate.
    - If the counter is not 0, we decrement the counter if e is not the current candidate.
- Visualization: `http://www.cs.utexas.edu/~moore/best-ideas/mjrty/example.html`

Finding Mode

- **Mode:** The mode of a set of numbers is the element that occurs most often.
- **Algorithm:** Sort and find the longest sequence $O(n \lg n)$.

Summary

Major Concepts:

- Concept of Order Statistics and Rank
- Popular Order Statistics - Min, Max, Median
- Selection Problem
- Mode and Majority
- Cool, non-obvious algorithms for even the simplest problems!