Analysis of Algorithms

Medians and Order Statistics

Andres Mendez-Vazquez

September 30, 2018

Outline

Outline

Introduction

Fact:

The ith order statistic of a set of n elements is the ith smallest element.

Introduction

Fact:

The ith order statistic of a set of n elements is the ith smallest element.

Examples

- $i = 1$ we are talking the minimum.
- $i = n$ we are talking the maximum.
- When n is an odd number, the position i of the median is defined by $i = \frac{n+1}{2}$

Introduction

Fact:

The ith order statistic of a set of n elements is the ith smallest element.

Examples

- $i = 1$ we are talking the minimum.
- $i = n$ we are talking the maximum.
- When n is an odd number, the position i of the median is defined by $i = \frac{n+1}{2}$

Introduction

Fact:

The ith order statistic of a set of n elements is the ith smallest element.

Examples

- $i = 1$ we are talking the minimum.
- $i = n$ we are talking the maximum.
- When n is an odd number, the position i of the median is defined by $i = \frac{n+1}{2}$

Outline

Selection Problem

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.

Selection Problem

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.

Outline

Minimum-Maximum

Minimum using $n-1$ comparissons

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

Minimum-Maximum

Minimum using $n-1$ comparissons

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

Minimum-Maximum

Minimum using $n-1$ comparissons

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

Minimum-Maximum

Minimum using $n-1$ comparissons

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

Minimum-Maximum

Minimum using $n-1$ comparissons

Minimum(A)

$min = A[1]$
for $i = 2$ **to A.length**
    if $min > A[i]$
        $min = A[i]$
return min

Minimum-Maximum

Minimum using $n-1$ comparissons

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

Using an $O(n \log n)$ Algorithm

Assumption

Suppose n elements are sorted by an $O(n \log n)$ algorithm, e.g., MERGE-SORT.

Using an $O(n \log n)$ Algorithm

Assumption

Suppose n elements are sorted by an $O(n \log n)$ algorithm, e.g., MERGE-SORT.

Then the following properties hold for the ordered set

- Minimum: The first element.
- Maximum: The last element.
- The ith order statistic corresponds to the ith element.
- For the median:
  - If n is odd, then the median is equal to the $\frac{n+1}{2}$ th element.
  - If n is even:
    - The lower median is equal to the $\lfloor \frac{n+1}{2} \rfloor$ th element.
    - The upper median is equal to the $\lceil \frac{n+1}{2} \rceil$ th element.

Using an $O(n \log n)$ Algorithm

Assumption

Suppose n elements are sorted by an $O(n \log n)$ algorithm, e.g., MERGE-SORT.

Then the following properties hold for the ordered set

- Minimum: The first element.
- Maximum: The last element.
- The ith order statistic corresponds to the ith element.
- For the median:
  - If n is odd, then the median is equal to the $\frac{n+1}{2}$ th element.
  - If n is even:
    - The lower median is equal to the $\lfloor \frac{n+1}{2} \rfloor$ th element.
    - The upper median is equal to the $\lceil \frac{n+1}{2} \rceil$ th element.

Using an $O(n \log n)$ Algorithm

Assumption

Suppose n elements are sorted by an $O(n \log n)$ algorithm, e.g., MERGE-SORT.

Then the following properties hold for the ordered set

- Minimum: The first element.
- Maximum: The last element.
- The ith order statistic corresponds to the ith element.
- For the median:
  - If n is odd, then the median is equal to the $\frac{n+1}{2}$ th element.
  - If n is even:
    - The lower median is equal to the $\lfloor \frac{n+1}{2} \rfloor$ th element.
    - The upper median is equal to the $\lceil \frac{n+1}{2} \rceil$ th element.

Using an $O(n \log n)$ Algorithm

Assumption

Suppose n elements are sorted by an $O(n \log n)$ algorithm, e.g., MERGE-SORT.

Then the following properties hold for the ordered set

- Minimum: The first element.
- Maximum: The last element.
- The ith order statistic corresponds to the ith element.
- For the median:
  - If n is odd, then the median is equal to the $\frac{n+1}{2}$ th element.
  - If n is even:
    - The lower median is equal to the $\lfloor \frac{n+1}{2} \rfloor$ th element.
    - The upper median is equal to the $\lceil \frac{n+1}{2} \rceil$ th element.

Using an $O(n \log n)$ Algorithm

Assumption

Suppose n elements are sorted by an $O(n \log n)$ algorithm, e.g., MERGE-SORT.

Then the following properties hold for the ordered set

- Minimum: The first element.
- Maximum: The last element.
- The ith order statistic corresponds to the ith element.
- For the median:
  - If n is odd, then the median is equal to the $\frac{n+1}{2}$ th element.
  - If n is even:
    - The lower median is equal to the $\lfloor \frac{n+1}{2} \rfloor$ th element.
    - The upper median is equal to the $\lceil \frac{n+1}{2} \rceil$ th element.

Using an $O(n \log n)$ Algorithm

Assumption

Suppose n elements are sorted by an $O(n \log n)$ algorithm, e.g., MERGE-SORT.

Then the following properties hold for the ordered set

- Minimum: The first element.
- Maximum: The last element.
- The ith order statistic corresponds to the ith element.
- For the median:
  - If n is odd, then the median is equal to the $\frac{n+1}{2}$ th element.
  - If n is even:
    - The lower median is equal to the $\left\lfloor \frac{n+1}{2} \right\rfloor$ th element.
    - The upper median is equal to the $\left\lceil \frac{n+1}{2} \right\rceil$ th element.

Using an $O(n \log n)$ Algorithm

Assumption

Suppose n elements are sorted by an $O(n \log n)$ algorithm, e.g., MERGE-SORT.

Then the following properties hold for the ordered set

- Minimum: The first element.
- Maximum: The last element.
- The ith order statistic corresponds to the ith element.
- For the median:
  - If n is odd, then the median is equal to the $\frac{n+1}{2}$ th element.
  - If n is even:
    - The lower median is equal to the $\lfloor \frac{n+1}{2} \rfloor$ th element.
    - The upper median is equal to the $\lceil \frac{n+1}{2} \rceil$ th element.

Using an $O(n \log n)$ Algorithm

Important fact!

All selections can be done in $O(1)$, so total: $O(n \log n)$.

Question!!!

- Can we do better?

Using an $O(n \log n)$ Algorithm

Important fact!

All selections can be done in $O(1)$, so total: $O(n \log n)$.

Question!!!

- Can we do better?
- How many comparisons are needed to get the max and min of n elements?

Using an $O(n \log n)$ Algorithm

Important fact!

All selections can be done in $O(1)$, so total: $O(n \log n)$.

Question!!!

- Can we do better?
- How many comparisons are needed to get the max and min of n elements?

Minimum and Maximum at the same time... better choice

Naively

The naïve Maximum and Minimum at the same will take $2n - 2$ comparisons.

Minimum and Maximum at the same time... better choice

Naively

The naïve Maximum and Minimum at the same will take $2n - 2$ comparisons.

Something better?

- Take two elements at the same time.
- Compare them to get the min and max in the tuple.
- Compare the min in the tuple with the global min and do the same with the tuple max.

Minimum and Maximum at the same time... better choice

Naively

The naïve Maximum and Minimum at the same will take $2n - 2$ comparisons.

Something better?

- Take two elements at the same time.
- Compare them to get the min and max in the tuple.
- Compare the min in the tuple with the global min and do the same with the tuple max.

Minimum and Maximum at the same time... better choice

Naively

The naïve Maximum and Minimum at the same will take $2n - 2$ comparisons.

Something better?

- Take two elements at the same time.
- Compare them to get the min and max in the tuple.
- Compare the min in the tuple with the global min and do the same with the tuple max.

Minimum and Maximum at the same time... better choice

This will give you

$3\lfloor \frac{n}{2} \rfloor$comparisons.

Why?

Let's see!

Minimum and Maximum at the same time... better choice

This will give you

$3\lfloor \frac{n}{2} \rfloor$comparisons.

Why?

Let's see!

Outline

Selection in expected linear time $O(n)$

Selecting in expected linear time implies:

- Selecting the ith element.
- Use the divide and conquer algorithm RANDOMIZED-SELECT.
  - Similar to Quicksort, partition the input array recursively.
  - Unlike Quicksort, which works on both sides of the partition, just work on one side of the partition. This is called PRUNE-AND-SEARCH, prune one side, just search the other.

Selection in expected linear time $O(n)$

Selecting in expected linear time implies:

- Selecting the ith element.
- Use the divide and conquer algorithm RANDOMIZED-SELECT.
  - Similar to Quicksort, partition the input array recursively.
  - Unlike Quicksort, which works on both sides of the partition, just work on one side of the partition. This is called PRUNE-AND-SEARCH, prune one side, just search the other.

Homework

Please review or read Quicksort in Cormen's book (chapter 7).

Selection in expected linear time $O(n)$

Selecting in expected linear time implies:

- Selecting the ith element.
- Use the divide and conquer algorithm RANDOMIZED-SELECT.
  - Similar to Quicksort, partition the input array recursively.
  - Unlike Quicksort, which works on both sides of the partition, just work on one side of the partition. This is called PRUNE-AND-SEARCH, prune one side, just search the other.

Homework

Please review or read Quicksort in Cormen's book (chapter 7).

Selection in expected linear time $O(n)$

Selecting in expected linear time implies:

- Selecting the ith element.
- Use the divide and conquer algorithm RANDOMIZED-SELECT.
  - Similar to Quicksort, partition the input array recursively.
  - Unlike Quicksort, which works on both sides of the partition, just work on one side of the partition. This is called PRUNE-AND-SEARCH, prune one side, just search the other.

Homework

Please review or read Quicksort in Cormen's book (chapter 7).

Selection in expected linear time $O(n)$

Selecting in expected linear time implies:

- Selecting the ith element.
- Use the divide and conquer algorithm RANDOMIZED-SELECT.
  - Similar to Quicksort, partition the input array recursively.
  - Unlike Quicksort, which works on both sides of the partition, just work on one side of the partition. This is called PRUNE-AND-SEARCH, prune one side, just search the other.

Homework

Please review or read Quicksort in Cormen's book (chapter 7).

Outline

Strategy

First partition the set of n elements

- How? Remember Randomized Quicksort!!!

Thus

We get a q from the random partition!!!

Strategy

First partition the set of n elements

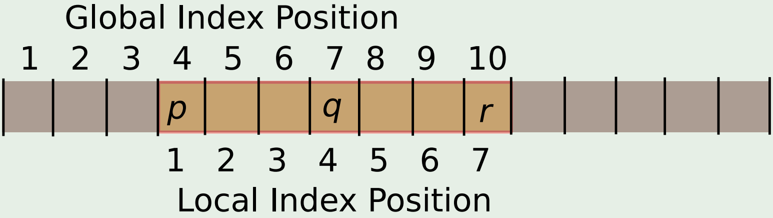
- How? Remember Randomized Quicksort!!!

Thus

We get a q from the random partition!!!

Example

Positions



p-q=3 then k=p-q+1 =4

Thus, the Strategy

If $i < k$

The possible ith smallest element is between p and $q - 1$.

Thus, the Strategy

If $i < k$

The possible ith smallest element is between p and $q - 1$.

If $i > k$

- The possible ith smallest element is between $q + 1$ and r.
- But the new $i' = i - k$, this will work because we start at $p = 1$ and $r = n$.

Thus, the Strategy

If $i < k$

The possible ith smallest element is between p and $q - 1$.

If $i > k$

- The possible ith smallest element is between $q + 1$ and r.
- But the new $i' = i - k$, this will work because we start at $p = 1$ and $r = n$.

If $i == k \Leftarrow$ We need to convert this to local index too

- return $A[q]$

Thus, the Strategy

If $i < k$

The possible ith smallest element is between p and $q - 1$.

If $i > k$

- The possible ith smallest element is between $q + 1$ and r.
- But the new $i' = i - k$, this will work because we start at $p = 1$ and $r = n$.

If $i == k \leftarrow$ We need to convert this to local index too

- return $A[q]$

RANDOMIZED-SELECT

RANDOMIZED-SELECT Algorithm

Randomized-Select(A, p, r, i)

1. **if** $p == r$
2. **return** $A[p]$
3. $q =$ Randomized-Partition(A, p, r)
4. $k = q - p + 1$ // **Local Index**
5. **if** $i == k$ // **The Answer**
6. **return** $A[q]$
7. **elseif** $i < k$
8. **return** Randomized-Select$(A, p, q - 1, i)$

   // **Converting to local index**

9. **else return** Randomized-Select$(A, q + 1, r, i - k)$

RANDOMIZED-SELECT

RANDOMIZED-SELECT Algorithm

Randomized-Select(A, p, r, i)

1. **if** $p == r$
2. **return** $A[p]$
3. $q =$ Randomized-Partition(A, p, r)
4. $k = q - p + 1$ // Local Index
5. **if** $i == k$ // The Answer
6. **return** $A[q]$
7. **elseif** $i < k$
8. **return** Randomized-Select$(A, p, q - 1, i)$

   // Converting to local index
9. **else return** Randomized-Select$(A, q + 1, r, i - k)$

RANDOMIZED-SELECT

RANDOMIZED-SELECT Algorithm

Randomized-Select(A, p, r, i)

if $p == r$
  return $A[p]$
$q =$ Randomized-Partition(A, p, r)
$k = q - p + 1$ // Local Index
if $i == k$ // The Answer
  return $A[q]$
elseif $i < k$
  return Randomized-Select$(A, p, q - 1, i)$
// Converting to local index
else return Randomized-Select$(A, q + 1, r, i - k)$

RANDOMIZED-SELECT

RANDOMIZED-SELECT Algorithm

Randomized-Select(A, p, r, i)

1. **if** $p == r$
2. **return** $A[p]$
3. $q =$ Randomized-Partition(A, p, r)
4. $k = q - p + 1$ // Local Index
5. **if** $i == k$ // The Answer
6. **return** $A[q]$
7. **elseif** $i < k$
8. **return** Randomized-Select$(A, p, q - 1, i)$

   // Converting to local index
9. **else return** Randomized-Select$(A, q + 1, r, i - k)$

RANDOMIZED-SELECT

RANDOMIZED-SELECT Algorithm

Randomized-Select(A, p, r, i)

1. **if** $p == r$
2.   **return** $A[p]$
3. $q =$ Randomized-Partition(A, p, r)
4. $k = q - p + 1$ // **Local Index**
5. **if** $i == k$ // The Answer
6.   **return** $A[q]$
7. **elseif** $i < k$
8.   **return** Randomized-Select$(A, p, q-1, i)$

   // Converting to local index
9. **else return** Randomized-Select$(A, q+1, r, i-k)$

RANDOMIZED-SELECT

RANDOMIZED-SELECT Algorithm

Randomized-Select(A, p, r, i)

if $p == r$
    return $A[p]$
$q =$ Randomized-Partition(A, p, r)
$k = q - p + 1$ // **Local Index**
if $i == k$ // **The Answer**
    return $A[q]$
elseif $i < k$
    return Randomized-Select$(A, p, q-1, i)$
// Converting to local index
else return Randomized-Select$(A, q+1, r, i-k)$

RANDOMIZED-SELECT

RANDOMIZED-SELECT Algorithm

Randomized-Select(A, p, r, i)

1. **if** $p == r$
2. **return** $A[p]$
3. $q =$ Randomized-Partition(A, p, r)
4. $k = q - p + 1$ // **Local Index**
5. **if** $i == k$ // **The Answer**
6. **return** $A[q]$
7. **elseif** $i < k$
8. **return** Randomized-Select$(A, p, q-1, i)$

   // Converting to local index
9. **else return** Randomized-Select$(A, q+1, r, i-k)$

RANDOMIZED-SELECT

RANDOMIZED-SELECT Algorithm

Randomized-Select(A, p, r, i)

```
if p == r
        return A[p]
q = Randomized-Partition(A, p, r)
k = q − p + 1 // Local Index
if i == k // The Answer
        return A[q]
elseif i < k
        return Randomized-Select(A, p, q − 1, i)
// Converting to local index
else return Randomized-Select(A, q + 1, r, i − k)
```

RANDOMIZED-SELECT

RANDOMIZED-SELECT Algorithm

Randomized-Select(A, p, r, i)

1. **if** $p == r$
2.   **return** $A[p]$
3. q = Randomized-Partition(A, p, r)
4. $k = q - p + 1$ // **Local Index**
5. **if** $i == k$ // **The Answer**
6.   **return** $A[q]$
7. **elseif** $i < k$
8.   **return** Randomized-Select$(A, p, q-1, i)$

    // Converting to local index

9. **else return** Randomized-Select$(A, q+1, r, i-k)$

RANDOMIZED-SELECT

RANDOMIZED-SELECT Algorithm

Randomized-Select(A, p, r, i)

1. **if** $p == r$
2. **return** $A[p]$
3. $q =$ Randomized-Partition(A, p, r)
4. $k = q - p + 1$ // **Local Index**
5. **if** $i == k$ // **The Answer**
6. **return** $A[q]$
7. **elseif** $i < k$
8. **return** Randomized-Select$(A, p, q - 1, i)$

   // **Converting to local index**
9. **else return** Randomized-Select$(A, q + 1, r, i - k)$

RANDOMIZED-SELECT

RANDOMIZED-SELECT Algorithm

Randomized-Select(A, p, r, i)

1. **if** $p == r$
2. **return** $A[p]$
3. $q =$ Randomized-Partition(A, p, r)
4. $k = q - p + 1$ // **Local Index**
5. **if** $i == k$ // **The Answer**
6. **return** $A[q]$
7. **elseif** $i < k$
8. **return** Randomized-Select$(A, p, q-1, i)$

   // **Converting to local index**
9. **else return** Randomized-Select$(A, q+1, r, i-k)$

Analysis of RANDOMIZED-SELECT

Worst-case running time $\Theta(n^2)$. Why?

An empty side and a side with remaining elements. So every partitioning of m elements will take $\Theta(m)$ time where $m = n, n-1, ..., 2$. Thus in total

$$\Theta(n) + \Theta(n-1) + ... + \Theta(2) = \Theta(\frac{n(n-1)}{2} - 1) = \Theta(n^2).$$

Analysis of RANDOMIZED-SELECT

Worst-case running time $\Theta(n^2)$. Why?

An empty side and a side with remaining elements. So every partitioning of m elements will take $\Theta(m)$ time where $m = n, n-1, ..., 2$. Thus in total

$$\Theta(n) + \Theta(n-1) + ... + \Theta(2) = \Theta\left(\frac{n(n-1)}{2} - 1\right) = \Theta(n^2).$$

Moreover

- No particular input elicits the worst-case behavior.
- In average, RANDOMIZED-SELECT is good because of the randomness.

Analysis of RANDOMIZED-SELECT

Worst-case running time $\Theta(n^2)$. Why?

An empty side and a side with remaining elements. So every partitioning of m elements will take $\Theta(m)$ time where $m = n, n-1, ..., 2$. Thus in total

$$\Theta(n) + \Theta(n-1) + ... + \Theta(2) = \Theta\left(\frac{n(n-1)}{2} - 1\right) = \Theta(n^2).$$

Moreover

- No particular input elicits the worst-case behavior.
- In average, RANDOMIZED-SELECT is good because of the randomness.

Analysis of RANDOMIZED-SELECT

Worst-case running time $\Theta(n^2)$. Why?

An empty side and a side with remaining elements. So every partitioning of m elements will take $\Theta(m)$ time where $m = n, n-1, ..., 2$. Thus in total

$$\Theta(n) + \Theta(n-1) + ... + \Theta(2) = \Theta(\frac{n(n-1)}{2} - 1) = \Theta(n^2).$$

Moreover

- No particular input elicits the worst-case behavior.
- In average, RANDOMIZED-SELECT is good because of the randomness.

Outline

Selection in worst-case linear time $O(n)$.

Goal:

Select the ith smallest element of $S = \{a_1, a_2, ..., a_n\}$.

Selection in worst-case linear time $O(n)$.

Goal:

Select the ith smallest element of $S = \{a_1, a_2, ..., a_n\}$.

Solution:

- Use the so called PRUNE-AND-SEARCH technique:
  - Let $x \in S$, and partition S into three subsets.
  - $S_1 = \{a_j | a_j < x\}$, $S_2 = \{a_j | a_j = x\}$, $S_3 = \{a_j | a_j > x\}$.
  - If $|S_1| > i$, search ith smallest elements in S_1 recursively, (prune S_2 and S_3 away).
  - Else If $|S_1| + |S_2| > i$, then return x (the ith smallest element).
  - Else search the $(i - (|S_1| + |S_2|))$th element in S_3 recursively (prune S_1 and S_2 away).

A question arises

How to select x such that S_1 and S_3 are nearly equal in cardinality? Force an even search!!!

Selection in worst-case linear time $O(n)$.

Goal:

Select the ith smallest element of $S = \{a_1, a_2, ..., a_n\}$.

Solution:

- Use the so called PRUNE-AND-SEARCH technique:
  - Let $x \in S$, and partition S into three subsets.
  - $S_1 = \{a_j | a_j < x\}$, $S_2 = \{a_j | a_j = x\}$, $S_3 = \{a_j | a_j > x\}$.
  - If $|S_1| > i$, search ith smallest elements in S_1 recursively, (prune S_2 and S_3 away).
  - Else If $|S_1| + |S_2| > i$, then return x (the ith smallest element).
  - Else search the $(i - (|S_1| + |S_2|))$th element in S_3 recursively (prune S_1 and S_2 away).

A question arises

How to select x such that S_1 and S_3 are nearly equal in cardinality? Force an even search!!!

Selection in worst-case linear time $O(n)$.

Goal:

Select the ith smallest element of $S = \{a_1, a_2, ..., a_n\}$.

Solution:

- Use the so called PRUNE-AND-SEARCH technique:
  - Let $x \in S$, and partition S into three subsets.
  - $S_1 = \{a_j | a_j < x\}$, $S_2 = \{a_j | a_j = x\}$, $S_3 = \{a_j | a_j > x\}$.
  - If $|S_1| > i$, search ith smallest elements in S_1 recursively, (prune S_2 and S_3 away).
  - Else If $|S_1| + |S_2| > i$, then return x (the ith smallest element).
  - Else search the $(i - (|S_1| + |S_2|))$th element in S_3 recursively (prune S_1 and S_2 away).

A question arises

How to select x such that S_1 and S_3 are nearly equal in cardinality? Force an even search!!!

Selection in worst-case linear time $O(n)$.

Goal:

Select the ith smallest element of $S = \{a_1, a_2, ..., a_n\}$.

Solution:

- Use the so called PRUNE-AND-SEARCH technique:
  - Let $x \in S$, and partition S into three subsets.
  - $S_1 = \{a_j | a_j < x\}$, $S_2 = \{a_j | a_j = x\}$, $S_3 = \{a_j | a_j > x\}$.
  - If $|S_1| > i$, search ith smallest elements in S_1 recursively, (prune S_2 and S_3 away).
  - Else If $|S_1| + |S_2| > i$, then return x (the ith smallest element).
  - Else search the $(i - (|S_1| + |S_2|))$th element in S_3 recursively (prune S_1 and S_2 away).

A question arises

How to select x such that S_1 and S_3 are nearly equal in cardinality? Force an even search!!!

Selection in worst-case linear time $O(n)$.

Goal:

Select the ith smallest element of $S = \{a_1, a_2, ..., a_n\}$.

Solution:

- Use the so called PRUNE-AND-SEARCH technique:
  - Let $x \in S$, and partition S into three subsets.
  - $S_1 = \{a_j | a_j < x\}$, $S_2 = \{a_j | a_j = x\}$, $S_3 = \{a_j | a_j > x\}$.
  - If $|S_1| > i$, search ith smallest elements in S_1 recursively, (prune S_2 and S_3 away).
  - Else If $|S_1| + |S_2| > i$, then return x (the ith smallest element).
  - Else search the $(i - (|S_1| + |S_2|))$th element in S_3 recursively (prune S_1 and S_2 away).

A question arises

How to select x such that S_1 and S_3 are nearly equal in cardinality? Force an even search!!!

Selection in worst-case linear time $O(n)$.

Goal:

Select the ith smallest element of $S = \{a_1, a_2, ..., a_n\}$.

Solution:

- Use the so called PRUNE-AND-SEARCH technique:
  - Let $x \in S$, and partition S into three subsets.
  - $S_1 = \{a_j | a_j < x\}$, $S_2 = \{a_j | a_j = x\}$, $S_3 = \{a_j | a_j > x\}$.
  - If $|S_1| > i$, search ith smallest elements in S_1 recursively, (prune S_2 and S_3 away).
  - Else If $|S_1| + |S_2| > i$, then return x (the ith smallest element).
  - Else search the $(i - (|S_1| + |S_2|))$th element in S_3 recursively (prune S_1 and S_2 away).

A question arises

How to select x such that S_1 and S_3 are nearly equal in cardinality? Force an even search!!!

Selection in worst-case linear time $O(n)$.

Goal:

Select the ith smallest element of $S = \{a_1, a_2, ..., a_n\}$.

Solution:

- Use the so called PRUNE-AND-SEARCH technique:
  - Let $x \in S$, and partition S into three subsets.
  - $S_1 = \{a_j | a_j < x\}$, $S_2 = \{a_j | a_j = x\}$, $S_3 = \{a_j | a_j > x\}$.
  - If $|S_1| > i$, search ith smallest elements in S_1 recursively, (prune S_2 and S_3 away).
  - Else If $|S_1| + |S_2| > i$, then return x (the ith smallest element).
  - Else search the $(i - (|S_1| + |S_2|))$th element in S_3 recursively (prune S_1 and S_2 away).

A question arises

How to select x such that S_1 and S_3 are nearly equal in cardinality? Force an even search!!!

Selection in worst-case linear time $O(n)$.

Goal:

Select the ith smallest element of $S = \{a_1, a_2, ..., a_n\}$.

Solution:

- Use the so called PRUNE-AND-SEARCH technique:
  - Let $x \in S$, and partition S into three subsets.
  - $S_1 = \{a_j | a_j < x\}$, $S_2 = \{a_j | a_j = x\}$, $S_3 = \{a_j | a_j > x\}$.
  - If $|S_1| > i$, search ith smallest elements in S_1 recursively, (prune S_2 and S_3 away).
  - Else If $|S_1| + |S_2| > i$, then return x (the ith smallest element).
  - Else search the $(i - (|S_1| + |S_2|))$th element in S_3 recursively (prune S_1 and S_2 away).

A question arises

How to select x such that S_1 and S_3 are nearly equal in cardinality? Force an even search!!!

Outline

The way to select x

Divide elements into $\left\lceil \frac{n}{5} \right\rceil$ groups of 5 elements each and find the median of each one

- We cannot say anything about the order between elements, but between median an elements
- Thus, arrows go from less to greater!!!

The way to select x

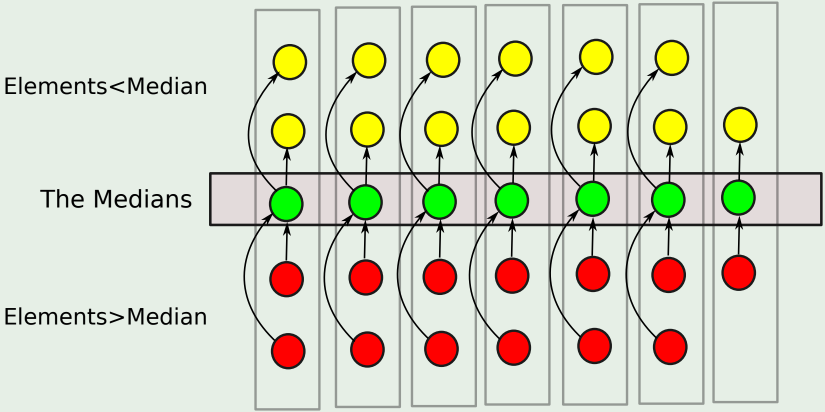
Divide elements into $\left\lceil \frac{n}{5} \right\rceil$ groups of 5 elements each and find the median of each one

- We cannot say anything about the order between elements, but between median an elements
- Thus, arrows go from less to greater!!!

The way to select x

Divide elements into $\left\lceil \frac{n}{5} \right\rceil$ groups of 5 elements each and find the median of each one

- We cannot say anything about the order between elements, but between median an elements
- Thus, arrows go from less to greater!!!



The way to select x

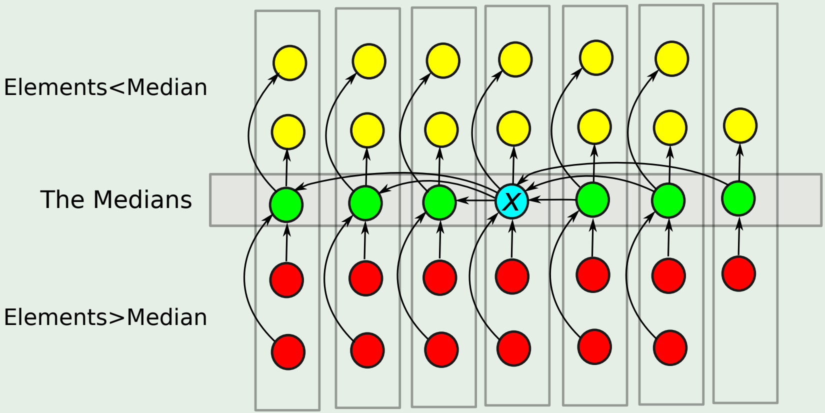
Find the Median of the Medians

- Again the arrows indicate the order from greater to less

The way to select x

Find the Median of the Medians

- Again the arrows indicate the order from greater to less



The way to select x

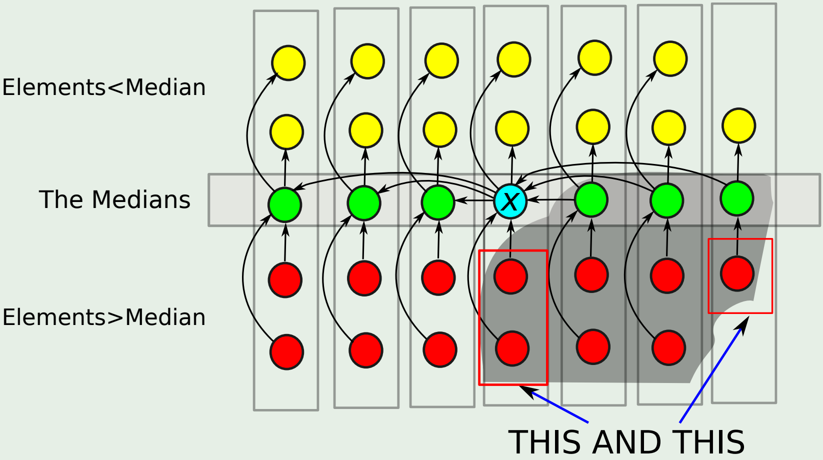
We have (Here, again the worst case scenario!!!)

- At least $\frac{1}{2}\left\lceil \frac{n}{5} \right\rceil - 2$ possible groups with 3 elements **greater** than x

The way to select x

We have (Here, again the worst case scenario!!!)

- At least $\frac{1}{2}\left\lceil\frac{n}{5}\right\rceil - 2$ possible groups with 3 elements **greater** than x



The way to select x

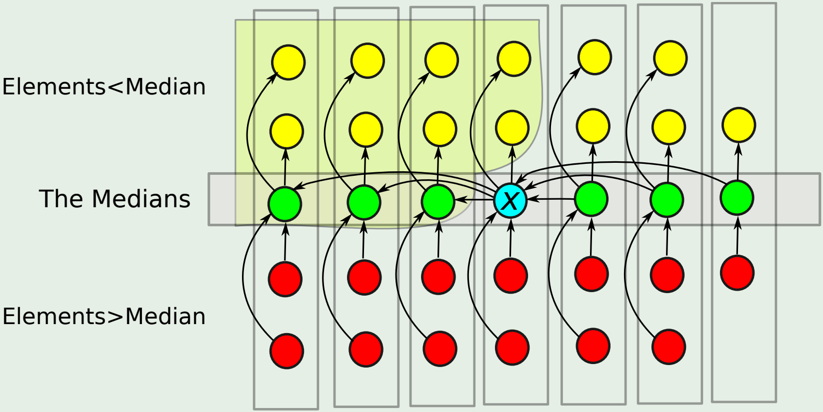
We have (Here, again the worst case scenario!!!)

- At least $\frac{1}{2}\left\lceil \frac{n}{5} \right\rceil - 2$ possible groups with 3 elements **less** than x

The way to select x

We have (Here, again the worst case scenario!!!)

- At least $\frac{1}{2}\left\lceil\frac{n}{5}\right\rceil - 2$ possible groups with 3 elements **less** than x



Thus, we have

First

$3\left(\frac{1}{2}\left\lceil\frac{n}{5}\right\rceil - 2\right) = \frac{3n}{10} - 6$ elements $< x$

Second

$3\left(\frac{1}{2}\left\lceil\frac{n}{5}\right\rceil - 2\right) = \frac{3n}{10} - 6$ elements $> x$

Thus, we have

First

$3\left(\frac{1}{2}\left\lceil\frac{n}{5}\right\rceil-2\right)=\frac{3n}{10}-6$ elements $< x$

Second

$3\left(\frac{1}{2}\left\lceil\frac{n}{5}\right\rceil-2\right)=\frac{3n}{10}-6$ elements $> x$

Outline

SELECT the ith element in n elements

Proceed as follows:

1. Divide n elements into $\lceil \frac{n}{5} \rceil$ groups of 5 elements.
2. Find the median of each group.
3. Use SELECT recursively to find the median x of the above$\lceil \frac{n}{5} \rceil$medians.
4. Partition n elements around x into S_1, S_2, and S_3.
5. If $|S_1| > i$, search ith smallest element in S_1 recursively.
   - Else If $|S_1| + |S_2| > i$, then return x (the ith smallest element).
   - Else search $(i - (|S_1| + |S_2|))$ th in S_3 recursively

SELECT the ith element in n elements

Proceed as follows:

1. Divide n elements into $\lceil \frac{n}{5} \rceil$ groups of 5 elements.
2. Find the median of each group.
3. Use SELECT recursively to find the median x of the above$\lceil \frac{n}{5} \rceil$medians.
4. Partition n elements around x into S_1, S_2, and S_3.
5. If $|S_1| > i$, search ith smallest element in S_1 recursively.
   - Else If $|S_1| + |S_2| > i$, then return x (the ith smallest element).
   - Else search $(i - (|S_1| + |S_2|))$ th in S_3 recursively

SELECT the ith element in n elements

Proceed as follows:

1. Divide n elements into $\lceil \frac{n}{5} \rceil$ groups of 5 elements.
2. Find the median of each group.
3. Use SELECT recursively to find the median x of the above $\lceil \frac{n}{5} \rceil$ medians.
4. Partition n elements around x into S_1, S_2, and S_3.
5. If $|S_1| > i$, search ith smallest element in S_1 recursively.
   - Else If $|S_1| + |S_2| > i$, then return x (the ith smallest element).
   - Else search $(i - (|S_1| + |S_2|))$ th in S_3 recursively

SELECT the ith element in n elements

Proceed as follows:

1. Divide n elements into $\lceil \frac{n}{5} \rceil$ groups of 5 elements.
2. Find the median of each group.
3. Use SELECT recursively to find the median x of the above $\lceil \frac{n}{5} \rceil$ medians.
4. Partition n elements around x into S_1, S_2, and S_3.
5. If $|S_1| > i$, search ith smallest element in S_1 recursively.
   - Else If $|S_1| + |S_2| > i$, then return x (the ith smallest element).
   - Else search $(i - (|S_1| + |S_2|))$ th in S_3 recursively

SELECT the ith element in n elements

Proceed as follows:

1. Divide n elements into $\lceil \frac{n}{5} \rceil$ groups of 5 elements.
2. Find the median of each group.
3. Use SELECT recursively to find the median x of the above $\lceil \frac{n}{5} \rceil$ medians.
4. Partition n elements around x into S_1, S_2, and S_3.
5. If $|S_1| > i$, search ith smallest element in S_1 recursively.
   - Else If $|S_1| + |S_2| > i$, then return x (the ith smallest element).
   - Else search $(i - (|S_1| + |S_2|))$ th in S_3 recursively

cinvestav

SELECT the ith element in n elements

Proceed as follows:

1. Divide n elements into $\lceil \frac{n}{5} \rceil$ groups of 5 elements.
2. Find the median of each group.
3. Use SELECT recursively to find the median x of the above $\lceil \frac{n}{5} \rceil$ medians.
4. Partition n elements around x into S_1, S_2, and S_3.
5. If $|S_1| > i$, search ith smallest element in S_1 recursively.
    - Else If $|S_1| + |S_2| > i$, then return x (the ith smallest element).
    - Else search $(i - (|S_1| + |S_2|))$ th in S_3 recursively

SELECT the ith element in n elements

Proceed as follows:

1. Divide n elements into $\lceil \frac{n}{5} \rceil$ groups of 5 elements.
2. Find the median of each group.
3. Use SELECT recursively to find the median x of the above $\lceil \frac{n}{5} \rceil$ medians.
4. Partition n elements around x into S_1, S_2, and S_3.
5. If $|S_1| > i$, search ith smallest element in S_1 recursively.
   - Else If $|S_1| + |S_2| > i$, then return x (the ith smallest element).
   - Else search $(i - (|S_1| + |S_2|))$ th in S_3 recursively

SELECT the ith element in n elements

Proceed as follows:

1. Divide n elements into $\lceil \frac{n}{5} \rceil$ groups of 5 elements.
2. Find the median of each group.
3. Use SELECT recursively to find the median x of the above$\lceil \frac{n}{5} \rceil$medians.
4. Partition n elements around x into S_1, S_2, and S_3.
5. If $|S_1| > i$, search ith smallest element in S_1 recursively.
   - Else If $|S_1| + |S_2| > i$, then return x (the ith smallest element).
   - Else search $(i - (|S_1| + |S_2|))$ th in S_3 recursively

Outline

Analysis of SELECT

Analysing complexity

- Steps 1,2 and 4 take O(n).
- Step 3 takes $T\left(\left\lceil \frac{n}{5} \right\rceil\right)$.

Analysis of SELECT

Analysing complexity

- Steps 1,2 and 4 take O(n).
- Step 3 takes $T(\lceil \frac{n}{5} \rceil)$.

Let us see step 5:

- At least half of the medians in step 2 are greater or equal than x, thus at least $\frac{1}{2}\lceil \frac{n}{5} \rceil - 2$ groups contribute 3 elements which are greater or equal than x. i.e., $3(\left\lceil \frac{1}{2}\lceil \frac{n}{5} \rceil \right\rceil - 2) \geq \frac{3n}{10} - 6$.
- Similarly, the number of elements less or equal than x is also at least $\frac{3n}{10} - 6$.
- Thus, $|S_1|$ is at most $\frac{7n}{10} + 6$, similarly for $|S_3|$.
- Thus SELECT in step 5 is called recursively on at most $\frac{7n}{10} + 6$ elements.

Analysis of SELECT

Analysing complexity

- Steps 1,2 and 4 take O(n).
- Step 3 takes $T(\lceil \frac{n}{5} \rceil)$.

Let us see step 5:

- At least half of the medians in step 2 are greater or equal than x, thus at least $\frac{1}{2}\lceil \frac{n}{5} \rceil - 2$ groups contribute 3 elements which are greater or equal than x. i.e., $3(\left\lceil \frac{1}{2}\lceil \frac{n}{5} \rceil \right\rceil - 2) \geq \frac{3n}{10} - 6$.
- Similarly, the number of elements less or equal than x is also at least $\frac{3n}{10} - 6$.
- Thus, $|S_1|$ is at most $\frac{7n}{10} + 6$, similarly for $|S_3|$.
- Thus SELECT in step 5 is called recursively on at most $\frac{7n}{10} + 6$ elements.

Analysis of SELECT

Analysing complexity

- Steps 1,2 and 4 take O(n).
- Step 3 takes $T(\lceil \frac{n}{5} \rceil)$.

Let us see step 5:

- At least half of the medians in step 2 are greater or equal than x, thus at least $\frac{1}{2}\lceil \frac{n}{5} \rceil - 2$ groups contribute 3 elements which are greater or equal than x. i.e., $3(\left\lceil \frac{1}{2}\lceil \frac{n}{5} \rceil \right\rceil - 2) \geq \frac{3n}{10} - 6$.
- Similarly, the number of elements less or equal than x is also at least $\frac{3n}{10} - 6$.
- Thus, $|S_1|$ is at most $\frac{7n}{10} + 6$, similarly for $|S_3|$.
- Thus SELECT in step 5 is called recursively on at most $\frac{7n}{10} + 6$ elements.

Analysis of SELECT

Analysing complexity

- Steps 1,2 and 4 take O(n).
- Step 3 takes $T(\lceil \frac{n}{5} \rceil)$.

Let us see step 5:

- At least half of the medians in step 2 are greater or equal than x, thus at least $\frac{1}{2}\lceil \frac{n}{5} \rceil - 2$ groups contribute 3 elements which are greater or equal than x. i.e., $3(\left\lceil \frac{1}{2}\lceil \frac{n}{5} \rceil \right\rceil - 2) \geq \frac{3n}{10} - 6$.
- Similarly, the number of elements less or equal than x is also at least $\frac{3n}{10} - 6$.
- Thus, $|S_1|$ is at most $\frac{7n}{10} + 6$, similarly for $|S_3|$.
- Thus SELECT in step 5 is called recursively on at most $\frac{7n}{10} + 6$ elements.

Analysis of SELECT

Analysing complexity

- Steps 1,2 and 4 take O(n).
- Step 3 takes $T(\lceil \frac{n}{5} \rceil)$.

Let us see step 5:

- At least half of the medians in step 2 are greater or equal than x, thus at least $\frac{1}{2}\lceil \frac{n}{5} \rceil - 2$ groups contribute 3 elements which are greater or equal than x. i.e., $3(\left\lceil \frac{1}{2}\lceil \frac{n}{5} \rceil \right\rceil - 2) \geq \frac{3n}{10} - 6$.
- Similarly, the number of elements less or equal than x is also at least $\frac{3n}{10} - 6$.
- Thus, $|S_1|$ is at most $\frac{7n}{10} + 6$, similarly for $|S_3|$.
- Thus SELECT in step 5 is called recursively on at most $\frac{7n}{10} + 6$ elements.

Final Recursion

We have then

$$T(n) = \begin{cases} O(1) & \text{if } n < \text{some value (i.e. 140)} \\ T\left(\left\lceil \frac{n}{5} \right\rceil\right) + T\left(\frac{7n}{10} + 6\right) + O(n) & \text{if } n \geq \text{some value (i.e. 140)} \end{cases}$$

Solve recurrence by substitution

Suppose $T(n) \leq cn$ for some c

$$T(n) \leq c\lceil \frac{n}{5} \rceil + c\left(\frac{7n}{10} + 6\right) + an$$

$$\leq \frac{1}{5}cn + c + \frac{7}{10}cn + 6c + an$$

$$\leq \frac{9}{10}cn + 7c + an$$

$$\leq cn + \left(-\frac{1}{10}cn + an + 7c\right)$$

Solve recurrence by substitution

Suppose $T(n) \leq cn$ for some c

$$
\begin{aligned}
T(n) &\leq c\left\lceil \frac{n}{5} \right\rceil + c\left(\frac{7n}{10} + 6\right) + an \\
&\leq \frac{1}{5}cn + c + \frac{7}{10}cn + 6c + an \\
&\leq \frac{9}{10}cn + 7c + an \\
&\leq cn + \left(-\frac{1}{10}cn + an + 7c\right)
\end{aligned}
$$

Solve recurrence by substitution

Suppose $T(n) \leq cn$ for some c

$$
\begin{aligned}
T(n) &\leq c\lceil \frac{n}{5} \rceil + c\left(\frac{7n}{10} + 6\right) + an \\
&\leq \frac{1}{5}cn + c + \frac{7}{10}cn + 6c + an \\
&\leq \frac{9}{10}cn + 7c + an \\
&\leq cn + \left(-\frac{1}{10}cn + an + 7c\right)
\end{aligned}
$$

Solve recurrence by substitution

Suppose $T(n) \leq cn$ for some c

$$
\begin{aligned}
T(n) &\leq c\lceil \frac{n}{5} \rceil + c\left(\frac{7n}{10} + 6\right) + an \\
&\leq \frac{1}{5}cn + c + \frac{7}{10}cn + 6c + an \\
&\leq \frac{9}{10}cn + 7c + an \\
&\leq cn + \left(-\frac{1}{10}cn + an + 7c\right)
\end{aligned}
$$

Solve recurrence by substitution.

$T(n)$ is at most cn

- If $-\frac{1}{10}cn + an + 7c < 0$.
  - i.e., $c \geq 10a\left(\frac{n}{n-70}\right)$ when $n > 70$.
- So, select $n = 140$, and then $c \geq 20a$.

Solve recurrence by substitution.

$T(n)$ is at most cn

- If $-\frac{1}{10}cn + an + 7c < 0$.
    - i.e., $c \geq 10a(\frac{n}{n-70})$ when $n > 70$.
- So, select $n = 140$, and then $c \geq 20a$.

Note:

n may not be 140, any integer greater than 70 is OK.

Solve recurrence by substitution.

$T(n)$ is at most cn

- If $-\frac{1}{10}cn + an + 7c < 0$.
  - i.e., $c \geq 10a(\frac{n}{n-70})$ when $n > 70$.
- So, select $n = 140$, and then $c \geq 20a$.

Note:

n may not be 140, any integer greater than 70 is OK.

Solve recurrence by substitution.

$T(n)$ is at most cn

- If $-\frac{1}{10}cn + an + 7c < 0$.
  - i.e., $c \geq 10a(\frac{n}{n-70})$ when $n > 70$.
- So, select $n = 140$, and then $c \geq 20a$.

Note:

n may not be 140, any integer greater than 70 is OK.

Final Thoughts

Why group of size 5?

Using groups of 3 does not work, you can try and plug it into the claculations

What about 7 or bigger odd number

It does not change the computations, only by a constant

Final Thoughts

Why group of size 5?

Using groups of 3 does not work, you can try and plug it into the claculations

What about 7 or bigger odd number

It does not change the computations, only by a constant

Applications

Computer Vision

In the Median Filter:

- Given a neighborhood of n members of x, you need to find the median to substitute the value in x

Statistical Applications

Confidence intervals for quantiles

Applications

Computer Vision

In the Median Filter:

- Given a neighborhood of n members of x, you need to find the median to substitute the value in x

Statistical Applications

Confidence intervals for quantiles

- A machine may run on 10 batteries and shuts off when the ith battery dies. You will want to know the distribution of $X_{(i)}$.
- In machine-learning if you want to convert a continuous valued feature into Boolean features by bucketing it, one common approach is to partition it by percentile so that the cardinality of each Boolean feature is somewhat similar.

Applications

Computer Vision

In the Median Filter:

- Given a neighborhood of n members of x, you need to find the median to substitute the value in x

Statistical Applications

Confidence intervals for quantiles

- A machine may run on 10 batteries and shuts off when the ith battery dies. You will want to know the distribution of $X_{(i)}$.
- In machine-learning if you want to convert a continuous valued feature into Boolean features by bucketing it, one common approach is to partition it by percentile so that the cardinality of each Boolean feature is somewhat similar.

Applications

Computer Vision

In the Median Filter:

- Given a neighborhood of n members of x, you need to find the median to substitute the value in x

Statistical Applications

Confidence intervals for quantiles

- A machine may run on 10 batteries and shuts off when the ith battery dies. You will want to know the distribution of $X_{(i)}$.
- In machine-learning if you want to convert a continuous valued feature into Boolean features by bucketing it, one common approach is to partition it by percentile so that the cardinality of each Boolean feature is somewhat similar.

Outline

Summary.

The ith order statistic of n elements

$S = \{a_1, a_2, ..., a_n\}$:$i$th smallest elements:

- Minimum and maximum.
- Median, lower median, upper median.

Summary.

The ith order statistic of n elements

$S = \{a_1, a_2, ..., a_n\}$:$i$th smallest elements:

- Minimum and maximum.
- Median, lower median, upper median.

Selection in expected average linear time

- Worst case running time
- PRUNE-AND-SEARCH

- It is still unknown exactly how many comparisons are needed to determine the median.

Summary.

The ith order statistic of n elements

$S = \{a_1, a_2, ..., a_n\}$:$i$th smallest elements:

- Minimum and maximum.
- Median, lower median, upper median.

Selection in expected average linear time

- Worst case running time
- PRUNE-AND-SEARCH

Selection in worst-case linear time

- The fast randomized version is due to Hoare.
- **It is still unknown exactly how many comparisons are needed to determine the median.**

Summary.

The ith order statistic of n elements

$S = \{a_1, a_2, ..., a_n\}$:$i$th smallest elements:

- Minimum and maximum.
- Median, lower median, upper median.

Selection in expected average linear time

- Worst case running time
- PRUNE-AND-SEARCH

Selection in worst-case linear time

- The fast randomized version is due to Hoare.
- **It is still unknown exactly how many comparisons are needed to determine the median.**

Summary.

The ith order statistic of n elements

$S = \{a_1, a_2, ..., a_n\}$:$i$th smallest elements:

- Minimum and maximum.
- Median, lower median, upper median.

Selection in expected average linear time

- Worst case running time
- PRUNE-AND-SEARCH

Selection in worst-case linear time

- The fast randomized version is due to Hoare.
- It is still unknown exactly how many comparisons are needed to determine the median.

Summary.

The ith order statistic of n elements

$S = \{a_1, a_2, ..., a_n\}$:$i$th smallest elements:

- Minimum and maximum.
- Median, lower median, upper median.

Selection in expected average linear time

- Worst case running time
- PRUNE-AND-SEARCH

Selection in worst-case linear time

- The fast randomized version is due to Hoare.
- It is still unknown exactly how many comparisons are needed to determine the median.

Summary.

The ith order statistic of n elements

$S = \{a_1, a_2, ..., a_n\}$:$i$th smallest elements:

- Minimum and maximum.
- Median, lower median, upper median.

Selection in expected average linear time

- Worst case running time
- PRUNE-AND-SEARCH

Selection in worst-case linear time

- The fast randomized version is due to Hoare.
- **It is still unknown exactly how many comparisons are needed to determine the median.**

Exercises

From Cormen's Book Chapter 9

- 9.1-1
- 9.2-3
- 9.3-4
- 9.3-8
- 9.2