Selection and Related problems

We will examine several related problems:

- Finding the maximum: How often do we update the "candidate maximum"?
- Finding the second largest element ("tournament algorithm")
- Finding the maximum and minimum simultaneously
- Finding the kth smallest (selection problem).
  - Special case of selection: finding the median
  - Median: kth smallest element, where $k = \lceil \frac{n}{2} \rceil$
  - Note: with this definition:
    - Median of 5 items is the 3rd smallest
    - Median of 6 items is the 3rd smallest

These problems can be solved in $O(n \log n)$ time by sorting, but we can do better.

Finding the maximum

Finding the maximum

```
v = -∞
for i = 0 to n-1:
   if A[i] > v:
      v = A[i]
return v
```

Finding the maximum

```
v = -∞
for i = 0 to n-1:
   if A[i] > v:
      v = A[i]
return v
```

- Worst-case number of comparisons is n.

Finding the maximum

```
v = -∞
for i = 0 to n-1:
    if A[i] > v:
        v = A[i]
return v
```

- Worst-case number of comparisons is n.
- This can be reduced to $n - 1$

Finding the maximum

```
v = -∞
for i = 0 to n-1:
   if A[i] > v:
      v = A[i]
return v
```

- Worst-case number of comparisons is n.
- This can be reduced to $n - 1$
- How many times is the running maximum updated?

Finding the maximum

```
v = -∞
for i = 0 to n-1:
   if A[i] > v:
      v = A[i]
return v
```

- Worst-case number of comparisons is n.
- This can be reduced to $n-1$
- How many times is the running maximum updated?
  - In the worst case n.

Finding the maximum

```
v = -∞
for i = 0 to n-1:
   if A[i] > v:
      v = A[i]
return v
```

- Worst-case number of comparisons is n.
- This can be reduced to $n - 1$
- How many times is the running maximum updated?
  - In the worst case n.
  - What about the average case? ...

Average number of updates to the running maximum

Average number of updates to the running maximum

- Assume
  - all possible orderings (permutations) of A are equally likely
  - all n elements of A are distinct.

Average number of updates to the running maximum

- Assume
  - all possible orderings (permutations) of A are equally likely
  - all n elements of A are distinct.
- The running maximum gets updated on iteration i of the loop iff $\max\{A[0], \ldots, A[i]\} = A[i]$.

Average number of updates to the running maximum

- Assume
  - all possible orderings (permutations) of A are equally likely
  - all n elements of A are distinct.
- The running maximum gets updated on iteration i of the loop iff $\max\{A[0], \ldots, A[i]\} = A[i]$.
- The probability of this happening is $1/(i+1)$.

Average number of updates to the running maximum

- Assume
  - all possible orderings (permutations) of A are equally likely
  - all n elements of A are distinct.
- The running maximum gets updated on iteration i of the loop iff $\max\{A[0], \ldots, A[i]\} = A[i]$.
- The probability of this happening is $1/(i+1)$.
- Define indicator variables X_i:

$$X_i = \begin{cases} 1 & \text{if } v \text{ gets updated on iteration } \#i \\ 0 & \text{if } v \text{ does not get updated on iteration } \#i \end{cases}$$

Average number of updates to the running maximum

- Assume
  - all possible orderings (permutations) of A are equally likely
  - all n elements of A are distinct.
- The running maximum gets updated on iteration i of the loop iff $\max\{A[0], \ldots, A[i]\} = A[i]$.
- The probability of this happening is $1/(i+1)$.
- Define indicator variables X_i:

$$X_i = \begin{cases} 1 & \text{if } v \text{ gets updated on iteration } \#i \\ 0 & \text{if } v \text{ does not get updated on iteration } \#i \end{cases}$$

Then $E(X_i) = \frac{1}{i+1}$

Average number of updates to the running maximum

- Assume
  - all possible orderings (permutations) of A are equally likely
  - all n elements of A are distinct.
- The running maximum gets updated on iteration i of the loop iff $\max\{A[0], \ldots, A[i]\} = A[i]$.
- The probability of this happening is $1/(i+1)$.
- Define indicator variables X_i:

$$X_i = \begin{cases} 1 & \text{if } v \text{ gets updated on iteration } \#i \\ 0 & \text{if } v \text{ does not get updated on iteration } \#i \end{cases}$$

  Then $E(X_i) = \frac{1}{i+1}$

- The total number of times that v gets updated is:

$$X = \sum_{i=0}^{n-1} X_i$$

Average number of updates to the running maximum (continued)

The expected total number of times that v gets updated is:

Average number of updates to the running maximum (continued)

The expected total number of times that v gets updated is:

$E(X)$

Average number of updates to the running maximum (continued)

The expected total number of times that v gets updated is:

$$E(X) = E\left(\sum_{i=0}^{n-1} X_i\right)$$

Average number of updates to the running maximum (continued)

The expected total number of times that v gets updated is:

$$E(X) = E\left(\sum_{i=0}^{n-1} X_i\right) = \sum_{i=0}^{n-1} E\left(X_i\right)$$

Average number of updates to the running maximum (continued)

The expected total number of times that v gets updated is:

$$E(X) = E\left(\sum_{i=0}^{n-1} X_i\right) = \sum_{i=0}^{n-1} E\left(X_i\right) = \sum_{i=0}^{n-1} \frac{1}{i+1}$$

Average number of updates to the running maximum (continued)

The expected total number of times that v gets updated is:

$$E(X) = E\left(\sum_{i=0}^{n-1} X_i\right) = \sum_{i=0}^{n-1} E\left(X_i\right) = \sum_{i=0}^{n-1} \frac{1}{i+1} = \sum_{i=1}^{n} \frac{1}{i}$$

Average number of updates to the running maximum (continued)

The expected total number of times that v gets updated is:

$$E(X) = E\left(\sum_{i=0}^{n-1} X_i\right) = \sum_{i=0}^{n-1} E\left(X_i\right) = \sum_{i=0}^{n-1} \frac{1}{i+1} = \sum_{i=1}^{n} \frac{1}{i} = H_n$$

Average number of updates to the running maximum (continued)

The expected total number of times that v gets updated is:

$$E(X) = E\left(\sum_{i=0}^{n-1} X_i\right) = \sum_{i=0}^{n-1} E\left(X_i\right) = \sum_{i=0}^{n-1} \frac{1}{i+1} = \sum_{i=1}^{n} \frac{1}{i} = H_n = O(\log n)$$

Average number of updates to the running maximum (continued)

The expected total number of times that v gets updated is:

$$E(X) = E\left(\sum_{i=0}^{n-1} X_i\right) = \sum_{i=0}^{n-1} E\left(X_i\right) = \sum_{i=0}^{n-1} \frac{1}{i+1} = \sum_{i=1}^{n} \frac{1}{i} = H_n = O(\log n)$$

It can be shown that

$$H_n = \ln n + \gamma + o(1), \quad \text{where } \gamma = 0.5772157\ldots$$

Average number of updates to the running maximum (continued)

The expected total number of times that v gets updated is:

$$E(X) = E\left(\sum_{i=0}^{n-1} X_i\right) = \sum_{i=0}^{n-1} E\left(X_i\right) = \sum_{i=0}^{n-1} \frac{1}{i+1} = \sum_{i=1}^{n} \frac{1}{i} = H_n = O(\log n)$$

It can be shown that

$$H_n = \ln n + \gamma + o(1), \quad \text{where } \gamma = 0.5772157\ldots$$

If there are 30,000 elements in the list, the expected update count is about 10.9

Average number of updates to the running maximum (continued)

The expected total number of times that v gets updated is:

$$E(X) = E\left(\sum_{i=0}^{n-1} X_i\right) = \sum_{i=0}^{n-1} E\left(X_i\right) = \sum_{i=0}^{n-1} \frac{1}{i+1} = \sum_{i=1}^{n} \frac{1}{i} = H_n = O(\log n)$$

It can be shown that

$$H_n = \ln n + \gamma + o(1), \quad \text{where } \gamma = 0.5772157\ldots$$

If there are 30,000 elements in the list, the expected update count is about 10.9

If there are 3,000,000,000 elements in the list, the expected update count is about 22.4

Finding the second-largest element

Obvious algorithm:

1. Compute the maximum ($n - 1$ comparisons)
2. Delete the maximum
3. Compute the maximum of the remaining elements ($n - 2$ comparisons)

This requires $2n - 3$ comparisons. We can do better.

Finding the second-largest element

Finding the second-largest element

Use a tournament.

Finding the second-largest element

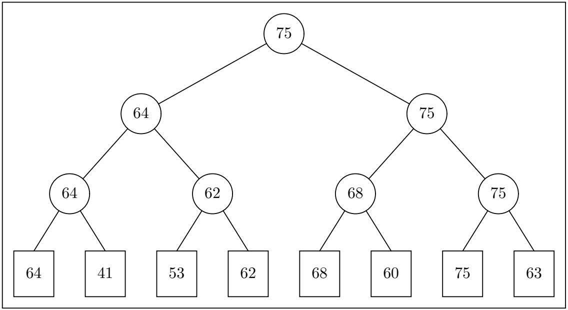
Use a tournament.

Example: Find second-largest from [64, 41, 53, 62, 68, 60, 75, 63]]

Finding the second-largest element

Use a tournament.

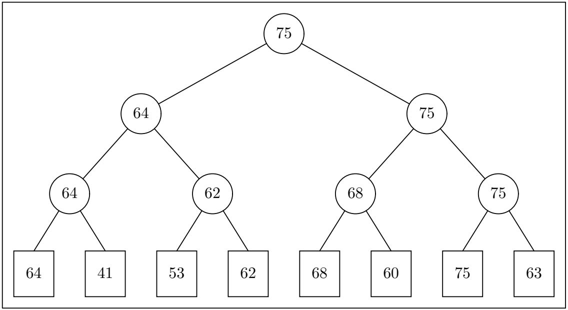
Example: Find second-largest from [64, 41, 53, 62, 68, 60, 75, 63]]



Finding the second-largest element

Use a tournament.

Example: Find second-largest from [64, 41, 53, 62, 68, 60, 75, 63]]



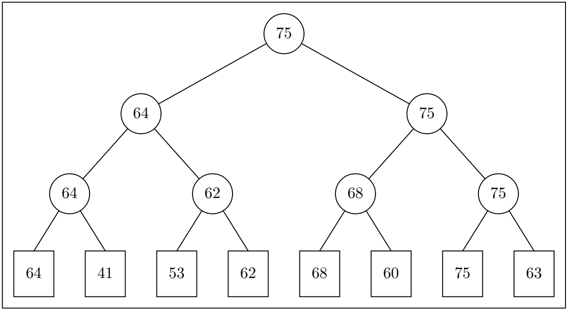
The second largest-element must be one of the elements that was compared directly with the largest element.

So it is the maximum of [63, 68, 64]

Finding the second-largest element

Use a tournament.

Example: Find second-largest from [64, 41, 53, 62, 68, 60, 75, 63]]



The second largest-element must be one of the elements that was compared directly with the largest element.

So it is the maximum of [63, 68, 64]

Total number of comparisons required is $7 + 2 = 9$

Analysis of tournament approach

Analysis of tournament approach

Tournament is a binary tree with n leaves, so its depth is $\lceil \lg n \rceil$.

Analysis of tournament approach

Tournament is a binary tree with n leaves, so its depth is $\lceil \lg n \rceil$.

Number of comparisons required:

Analysis of tournament approach

Tournament is a binary tree with n leaves, so its depth is $\lceil \lg n \rceil$.

Number of comparisons required:

1. $n - 1$ comparisons to find the maximum

Analysis of tournament approach

Tournament is a binary tree with n leaves, so its depth is $\lceil \lg n \rceil$.

Number of comparisons required:

1. $n - 1$ comparisons to find the maximum
2. $\lceil \lg n \rceil - 1$ comparisons to find the second largest

Analysis of tournament approach

Tournament is a binary tree with n leaves, so its depth is $\lceil \lg n \rceil$.

Number of comparisons required:

1. $n - 1$ comparisons to find the maximum
2. $\lceil \lg n \rceil - 1$ comparisons to find the second largest
   - (Computing the maximum of $\lceil \lg n \rceil$ candidates)

Total number of comparisons required is:

$$n + \lceil \lg n \rceil - 2$$

Analysis of tournament approach

Tournament is a binary tree with n leaves, so its depth is $\lceil \lg n \rceil$.

Number of comparisons required:

1. $n - 1$ comparisons to find the maximum
2. $\lceil \lg n \rceil - 1$ comparisons to find the second largest
   - (Computing the maximum of $\lceil \lg n \rceil$ candidates)

Total number of comparisons required is:

$$n + \lceil \lg n \rceil - 2$$

This is optimal. Proof can be found in [Baase, chapter 3].

Finding the maximum and the minimum

Obvious algorithm:

1. Compute the maximum ($n - 1$ comparisons)
2. Delete the maximum
3. Compute the minimum of the remaining elements ($n - 2$ comparisons)

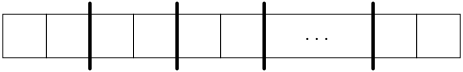
This requires $2n - 3$ comparisons. We can do better.

Finding the maximum and the minimum

Assume n is even.

Finding the maximum and the minimum

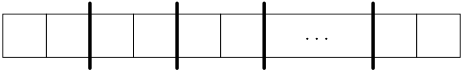
Assume n is even. Pair up the items:



Finding the maximum and the minimum

Assume n is even. Pair up the items:

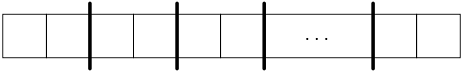
- First pair:



Finding the maximum and the minimum

Assume n is even. Pair up the items:

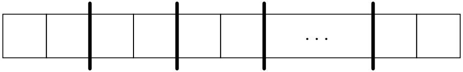
- First pair:
  1. Compare the items.



Finding the maximum and the minimum

Assume n is even. Pair up the items:

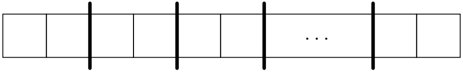
- First pair:
  1. Compare the items.
     - smaller item becomes running minimum



Finding the maximum and the minimum

Assume n is even. Pair up the items:

- First pair:
  1. Compare the items.
     - smaller item becomes running minimum
     - larger item becomes running maxium

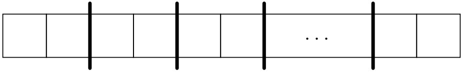


Finding the maximum and the minimum

Assume n is even. Pair up the items:

- First pair:
  1. Compare the items.
     - smaller item becomes running minimum
     - larger item becomes running maxium

     (1 comparison)

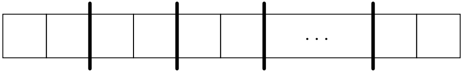


Finding the maximum and the minimum

Assume n is even. Pair up the items:

- First pair:
  1. Compare the items.
     - smaller item becomes running minimum
     - larger item becomes running maxium

  (1 comparison)
- Subsequent pairs:

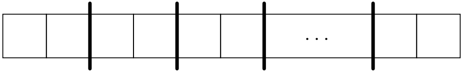


Finding the maximum and the minimum

Assume n is even. Pair up the items:

- First pair:
  1. Compare the items.
     - smaller item becomes running minimum
     - larger item becomes running maxium

     (1 comparison)
- Subsequent pairs:
  1. Compare the items

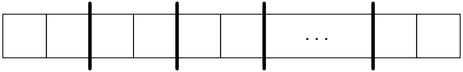


Finding the maximum and the minimum

Assume n is even. Pair up the items:

- First pair:
  1. Compare the items.
     - smaller item becomes running minimum
     - larger item becomes running maxium

  (1 comparison)
- Subsequent pairs:
  1. Compare the items
  2. Compare larger item in pair against running maximum. If it is larger it becomes the new running maximum

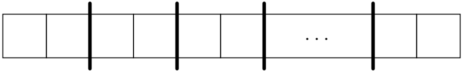


Finding the maximum and the minimum

Assume n is even. Pair up the items:

- First pair:
  1. Compare the items.
     - smaller item becomes running minimum
     - larger item becomes running maxium

     (1 comparison)
- Subsequent pairs:
  1. Compare the items
  2. Compare larger item in pair against running maximum. If it is larger it becomes the new running maximum
  3. Compare smaller item in pair against running minimum. If it is smaller it becomes the new running minimum



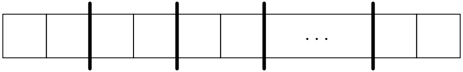
Finding the maximum and the minimum

Assume n is even. Pair up the items:

- First pair:
  1. Compare the items.
     - smaller item becomes running minimum
     - larger item becomes running maxium

     (1 comparison)
- Subsequent pairs:
  1. Compare the items
  2. Compare larger item in pair against running maximum. If it is larger it becomes the new running maximum
  3. Compare smaller item in pair against running minimum. If it is smaller it becomes the new running minimum

  (3 comparisons per pair, $\frac{n}{2} - 1$ pairs)



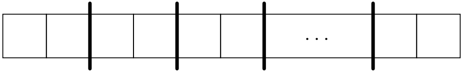
Finding the maximum and the minimum

Assume n is even. Pair up the items:

- First pair:
  1. Compare the items.
     - smaller item becomes running minimum
     - larger item becomes running maxium

  (1 comparison)
- Subsequent pairs:
  1. Compare the items
  2. Compare larger item in pair against running maximum. If it is larger it becomes the new running maximum
  3. Compare smaller item in pair against running minimum. If it is smaller it becomes the new running minimum

  (3 comparisons per pair, $\frac{n}{2} - 1$ pairs)



Total number of comparisons is $\frac{3n}{2} - 2$.

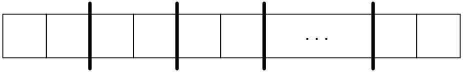
Finding the maximum and the minimum

Assume n is even. Pair up the items:

- First pair:
  1. Compare the items.
     - smaller item becomes running minimum
     - larger item becomes running maxium

  (1 comparison)
- Subsequent pairs:
  1. Compare the items
  2. Compare larger item in pair against running maximum. If it is larger it becomes the new running maximum
  3. Compare smaller item in pair against running minimum. If it is smaller it becomes the new running minimum

  (3 comparisons per pair, $\frac{n}{2} - 1$ pairs)



Total number of comparisons is $\frac{3n}{2} - 2$. This is optimal ...

Finding the maximum and the minimum

Finding the maximum and the minimum

We will show that there is not a better algorithm (with respect to the number of comparisons):

Finding the maximum and the minimum

We will show that there is not a better algorithm (with respect to the number of comparisons):

> *Any deterministic algorithm that computes the max and the min of n elements must perform at least $\frac{3n}{2} - 2$ comparisons in the worst case.*

Finding the maximum and the minimum

We will show that there is not a better algorithm (with respect to the number of comparisons):

> *Any deterministic algorithm that computes the max and the min of n elements must perform at least $\frac{3n}{2} - 2$ comparisons in the worst case.*

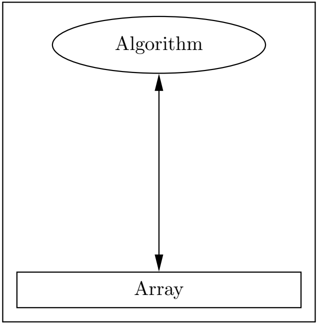
The proof is based on an adversary argument

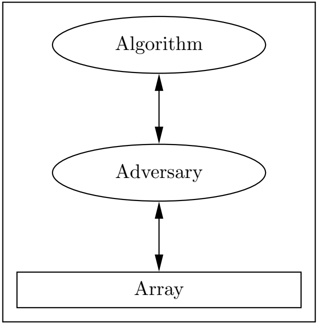
Adversary argument

Given any algorithm for the problem, we need to show that there is an input array of size n that forces the algorithm to either

- Do at least $\frac{3n}{2} - 2$ comparisons; or
- Give an incorrect answer.

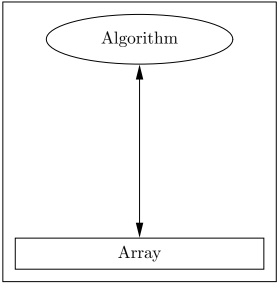
We construct such an array using an adversary.

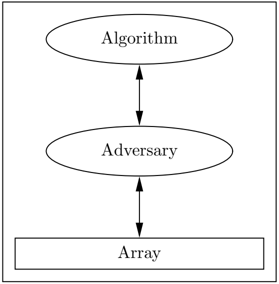




Adversary argument

- The adversary is allowed to and make up answers and set/modify array values
- The array must be consistent with all answers to comparison queries that the adversary has given
- If the algorithm were to run again on the final array, it would behave exactly the same.
- We will describe an adversary that forces any deterministic algorithm that computes maximum and minimum to perform $\frac{3n}{2} - 2$ comparisons.





Adversary strategy: the basics

- Terminology: Suppose we compare two array entries $A[i]$ and $A[j]$. If $A[i] > A[j]$, we say:
  - $A[i]$ wins the comparison
  - $A[j]$ loses the comparison
- A unit of information is the knowledge that a particular entry has won (or lost) at least one comparison.
- To correctly conclude that the maximum value is $A[r]$ and the minimum value is $A[s]$ is the minimum value, the algorithm must have determined that:
  - All elements other than $A[r]$ have lost at least one comparison. ($n - 1$ units of information)
  - All elements other than $A[s]$ have won at least one comparison. ($n - 1$ units of information)

  So the algorithm must collect $2n - 2$ units of information
- We will see that the adversary can force the algorithm to perform $\frac{3n}{2} - 2$ comparisons in order to collect these $2n - 2$ units of information.

Adversary strategy

- The adversary assigns each array entry value and a status.
- The initial value for each array entry is arbitrary. The adversary modifies the value for each array entry as the algorithm proceeds.
- The status is used to keep track of whether the entry has won or lost any comparisons.
- There are 4 possible status values:
  - **W**: The array entry has won at least one comparison and has not lost any
  - **L**: The array entry has lost at least one comparison and has not won any
  - **WL**: The array entry has won at least once and lost at least once
  - **N**: The array entry has not yet participated in any comparisons
- When the algorithm compares two array entries, the adversary:
  1. Examines statuses of the two entries
  2. Based on the statuses, decides who it will say won the comparison
  3. Modifies array entries if necessary to ensure that the array values are consistent with the returned result for this comparison and all previous comparisons.

Adversary strategy (Part 1 of 2)

Status $A[i]$	Status $A[j]$	Comparison Result	New Status $A[i]$	New Status $A[j]$	Units of Info.	Value Updates
N	N	$A[i] > A[j]$	W	L	2	Set $A[i]$, $A[j]$
N	W	$A[i] < A[j]$	L	W	1	Set $A[i]$
N	L	$A[i] > A[j]$	W	L	1	Set $A[i]$
N	WL	$A[i] > A[j]$	W	WL	1	Set $A[i]$
W	N	$A[i] > A[j]$	W	L	1	Set $A[j]$
W	W	$A[i] > A[j]$	W	WL	1	Increase $A[i]^*$
W	L	$A[i] > A[j]$	W	L	0	Increase $A[i]^*$
W	WL	$A[i] > A[j]$	W	WL	0	Increase $A[i]^*$
L	N	$A[i] < A[j]$	L	W	1	Set $A[j]$
L	W	$A[i] < A[j]$	L	W	0	Decrease $A[i]^*$
L	L	$A[i] < A[j]$	WL	L	1	Decrease $A[i]^*$
L	WL	$A[i] < A[j]$	L	WL	0	Decrease $A[i]^*$

* if necessary

Adversary strategy (Part 2 of 2)

Status $A[i]$	Status $A[j]$	Comparison Result	New Status $A[i]$	New Status $A[j]$	Units of Info.	value updates
WL	N	$A[i] < A[j]$	WL	W	1	Set $A[j]$
WL	W	$A[i] < A[j]$	WL	W	0	Increase $A[j]^*$
WL	L	$A[i] > A[j]$	WL	L	0	Decrease $A[j]^*$
WL	WL	consistent with values	WL	WL	0	—

* if necessary

Example of adversary strategy

Example of adversary strategy

Initially

Example of adversary strategy

Initially

	0	1	2	3	4	5	6
Status	N	N	N	N	N	N	N
Value							

Example of adversary strategy

Initially

	0	1	2	3	4	5	6
Status	N	N	N	N	N	N	N
Value							

$A[3] > A[1]$?

Example of adversary strategy

Initially

	0	1	2	3	4	5	6
Status	N	N	N	N	N	N	N
Value							

$A[3] > A[1]$?

	0	1	2	3	4	5	6
Status	N	L	N	W	N	N	N
Value		1		2			

True

Example of adversary strategy

Initially

	0	1	2	3	4	5	6
Status	N	N	N	N	N	N	N
Value							

$A[3] > A[1]?$

	0	1	2	3	4	5	6
Status	N	L	N	W	N	N	N
Value		1		2			

True

$A[1] > A[4]?$

Example of adversary strategy

Initially

	0	1	2	3	4	5	6
Status	N	N	N	N	N	N	N
Value							

$A[3] > A[1]$?

	0	1	2	3	4	5	6
Status	N	L	N	W	N	N	N
Value		1		2			

True

$A[1] > A[4]$?

	0	1	2	3	4	5	6
Status	N	L	N	W	W	N	N
Value		1		2	2		

False

Example of adversary strategy

Initially

	0	1	2	3	4	5	6
Status	N	N	N	N	N	N	N
Value							

$A[3] > A[1]$?

	0	1	2	3	4	5	6
Status	N	L	N	W	N	N	N
Value		1		2			

True

$A[1] > A[4]$?

	0	1	2	3	4	5	6
Status	N	L	N	W	W	N	N
Value		1		2	2		

False

$A[3] > A[4]$?

Example of adversary strategy

Initially

	0	1	2	3	4	5	6
Status	N	N	N	N	N	N	N
Value							

$A[3] > A[1]$?

	0	1	2	3	4	5	6
Status	N	L	N	W	N	N	N
Value		1		2			

True

$A[1] > A[4]$?

	0	1	2	3	4	5	6
Status	N	L	N	W	W	N	N
Value		1		2	2		

False

$A[3] > A[4]$?

	0	1	2	3	4	5	6
Status	N	L	N	W	WL	N	N
Value		1		3	2		

True

Example of adversary strategy

Initially

	0	1	2	3	4	5	6
Status	N	N	N	N	N	N	N
Value							

$A[3] > A[1]$?

	0	1	2	3	4	5	6
Status	N	L	N	W	N	N	N
Value		1		2			

True

$A[1] > A[4]$?

	0	1	2	3	4	5	6
Status	N	L	N	W	W	N	N
Value		1		2	2		

False

$A[3] > A[4]$?

	0	1	2	3	4	5	6
Status	N	L	N	W	WL	N	N
Value		1		3	2		

True

. . . . . .

Final steps in proof

Let c_1 = number of comparisons that yield 1 new unit of information

c_2 = number of comparisons that yield 2 new units of informatio n

c = total number of comparisons ($= c_1 + c_2$)

Facts:

1. Total units of information gleaned by algorithm $c_1 + 2c_2$
2. Total units of information must be at least $2n - 2$
3. At most $n/2$ comparisons (the NN comparisons) give the algorithm 2 units of information (so $c_2 \leq n/2$)

So

$$\begin{array}{rcll} c_1 + 2c_2 & \geq & 2n - 2 & \text{By Facts 1 and 2} \\ -c_2 & \geq & -\dfrac{n}{2} & \text{By Fact 3} \\ c = c_1 + c_2 & \geq & \dfrac{3n}{2} - 2 & \text{Add} \end{array}$$

Hence the total number of comparisons required is $\frac{3n}{2} - 2$.

Conclusions

The problem of finding the maximum and minimum of n elements

...

1. Can be solved using $\leq \frac{3n}{2} - 2$ comparisons in the worst case
   - Because there is an algorithm that solves the problem using $\frac{3n}{2} - 2$ in the worst case.
2. Requires $\geq \frac{3n}{2} - 2$ comparisons to solve in the worst case
   - Because as we just showed, for any algorithm that solves the problem an adversary can construct an input that forces the algorithm to perform

So we can conclude

- The algorithm we presented is optimal (by #2 above and the fact that our algorithm performs $\frac{3n}{2} - 2$ comparisons in the worst case)
- The lower bound of $\frac{3n}{2} - 2$ comparisons cannot be improved (by #1 above)

Selection

Selection

Selection Problem: From a collection of n items, find the kth smallest.

Selection

Selection Problem: From a collection of n items, find the kth smallest.

Earlier in these notes we examined $k = 2$ (or $k = n - 1$).

Selection

Selection Problem: From a collection of n items, find the kth smallest.

Earlier in these notes we examined $k = 2$ (or $k = n - 1$).

One obvious approach:

Selection

Selection Problem: From a collection of n items, find the kth smallest.

Earlier in these notes we examined $k = 2$ (or $k = n - 1$).

One obvious approach:

- Sort the n items, then return the element in position k.

Selection

Selection Problem: From a collection of n items, find the kth smallest.

Earlier in these notes we examined $k = 2$ (or $k = n - 1$).

One obvious approach:

- Sort the n items, then return the element in position k.
- This takes $O(n \log n)$.

Selection

Selection Problem: From a collection of n items, find the kth smallest.

Earlier in these notes we examined $k = 2$ (or $k = n - 1$).

One obvious approach:

- Sort the n items, then return the element in position k.
- This takes $O(n \log n)$.
- It can be improved to $O(k \log n)$ if we run heapsort and stop after computing the first k entries in sorted order.

Selection

Selection Problem: From a collection of n items, find the kth smallest.

Earlier in these notes we examined $k = 2$ (or $k = n - 1$).

One obvious approach:

- Sort the n items, then return the element in position k.
- This takes $O(n \log n)$.
- It can be improved to $O(k \log n)$ if we run heapsort and stop after computing the first k entries in sorted order.
  - If $k = O(n/\lg n)$, this runs in $O(n)$ time.

Selection

Selection Problem: From a collection of n items, find the kth smallest.

Earlier in these notes we examined $k = 2$ (or $k = n - 1$).

One obvious approach:

- Sort the n items, then return the element in position k.
- This takes $O(n \log n)$.
- It can be improved to $O(k \log n)$ if we run heapsort and stop after computing the first k entries in sorted order.
  - If $k = O(n/\lg n)$, this runs in $O(n)$ time.
  - However, if $k = \Theta(n)$ (e.g., if k is close to $n/2$), this takes $O(n \log n)$ time.

Selection

Selection Problem: From a collection of n items, find the kth smallest.

Earlier in these notes we examined $k = 2$ (or $k = n - 1$).

One obvious approach:

- Sort the n items, then return the element in position k.
- This takes $O(n \log n)$.
- It can be improved to $O(k \log n)$ if we run heapsort and stop after computing the first k entries in sorted order.
  - If $k = O(n/\lg n)$, this runs in $O(n)$ time.
  - However, if $k = \Theta(n)$ (e.g., if k is close to $n/2$), this takes $O(n \log n)$ time.

We will describe how to do selection in $O(n)$ time, irrespective of the value of k.

Selection

Selection

Problem formulation:

Selection

Problem formulation:

- S is a sequence containing n items (we will assume it is an array, but it could be a list).

Selection

Problem formulation:

- S is a sequence containing n items (we will assume it is an array, but it could be a list).
  - Duplicate values are allowed, e.g. [1,2,2,2,4,4,5]

Selection

Problem formulation:

- S is a sequence containing n items (we will assume it is an array, but it could be a list).
  - Duplicate values are allowed, e.g. [1,2,2,2,4,4,5]
- `Select(S,k)` returns a kth smallest item from the sequence S:

Selection

Problem formulation:

- S is a sequence containing n items (we will assume it is an array, but it could be a list).
  - Duplicate values are allowed, e.g. [1,2,2,2,4,4,5]
- `Select(S,k)` returns a kth smallest item from the sequence S:
  - If `Select(S,k)` returns x, there are at least $k-1$ items in S less than or equal to x, and at least $n-k$ values in S greater than or equal to x.

Selection

Problem formulation:

- S is a sequence containing n items (we will assume it is an array, but it could be a list).
  - Duplicate values are allowed, e.g. [1,2,2,2,4,4,5]
- `Select(S,k)` returns a kth smallest item from the sequence S:
  - If `Select(S,k)` returns x, there are at least $k - 1$ items in S less than or equal to x, and at least $n - k$ values in S greater than or equal to x.
  - Example: If $k = 3$ in list above, return 2.

Selection

Problem formulation:

- S is a sequence containing n items (we will assume it is an array, but it could be a list).
  - Duplicate values are allowed, e.g. [1,2,2,2,4,4,5]
- `Select(S,k)` returns a kth smallest item from the sequence S:
  - If `Select(S,k)` returns x, there are at least $k-1$ items in S less than or equal to x, and at least $n-k$ values in S greater than or equal to x.
  - Example: If $k=3$ in list above, return 2.

Our approach:

Selection

Problem formulation:

- S is a sequence containing n items (we will assume it is an array, but it could be a list).
  - Duplicate values are allowed, e.g. [1,2,2,2,4,4,5]
- `Select(S,k)` returns a kth smallest item from the sequence S:
  - If `Select(S,k)` returns x, there are at least $k-1$ items in S less than or equal to x, and at least $n-k$ values in S greater than or equal to x.
  - Example: If $k=3$ in list above, return 2.

Our approach:

- Algorithmic paradigm is Prune and Search (variant of Divide and Conquer). We will discuss two algorithms:

Selection

Problem formulation:

- S is a sequence containing n items (we will assume it is an array, but it could be a list).
  - Duplicate values are allowed, e.g. [1,2,2,2,4,4,5]
- `Select(S,k)` returns a kth smallest item from the sequence S:
  - If `Select(S,k)` returns x, there are at least $k-1$ items in S less than or equal to x, and at least $n-k$ values in S greater than or equal to x.
  - Example: If $k=3$ in list above, return 2.

Our approach:

- Algorithmic paradigm is Prune and Search (variant of Divide and Conquer). We will discuss two algorithms:
  1. Quickselect: randomized algorithm, similar to Quicksort. Runs in $O(n)$ average time, but worst case is $O(n^2)$.

Selection

Problem formulation:

- S is a sequence containing n items (we will assume it is an array, but it could be a list).
  - Duplicate values are allowed, e.g. [1,2,2,2,4,4,5]
- `Select(S,k)` returns a kth smallest item from the sequence S:
  - If `Select(S,k)` returns x, there are at least $k-1$ items in S less than or equal to x, and at least $n-k$ values in S greater than or equal to x.
  - Example: If $k=3$ in list above, return 2.

Our approach:

- Algorithmic paradigm is Prune and Search (variant of Divide and Conquer). We will discuss two algorithms:
  1. Quickselect: randomized algorithm, similar to Quicksort. Runs in $O(n)$ average time, but worst case is $O(n^2)$.
  2. Deterministic algorithm: runs in $O(n)$ worst-case time.

Selection

Problem formulation:

- S is a sequence containing n items (we will assume it is an array, but it could be a list).
  - Duplicate values are allowed, e.g. [1,2,2,2,4,4,5]
- `Select(S,k)` returns a kth smallest item from the sequence S:
  - If `Select(S,k)` returns x, there are at least $k-1$ items in S less than or equal to x, and at least $n-k$ values in S greater than or equal to x.
  - Example: If $k=3$ in list above, return 2.

Our approach:

- Algorithmic paradigm is Prune and Search (variant of Divide and Conquer). We will discuss two algorithms:
  1. Quickselect: randomized algorithm, similar to Quicksort. Runs in $O(n)$ average time, but worst case is $O(n^2)$.
  2. Deterministic algorithm: runs in $O(n)$ worst-case time.
- Both algorithms have the same high-level strategy.

Selection algorithms: High-level strategy

Selection algorithms: High-level strategy

Strategy common to the two algorithms:

Selection algorithms: High-level strategy

Strategy common to the two algorithms:

1. Choose an approximate median m^*.

Selection algorithms: High-level strategy

Strategy common to the two algorithms:

1. Choose an approximate median m^*.
2. Partition S into three subcollections:

L	E	G
$< m^*$	$= m^*$	$> m^*$

Selection algorithms: High-level strategy

Strategy common to the two algorithms:

1. Choose an approximate median m^*.
2. Partition S into three subcollections:
   - L: elements in S less than m^*.

L	E	G
$< m^*$	$= m^*$	$> m^*$

Selection algorithms: High-level strategy

Strategy common to the two algorithms:

1. Choose an approximate median m^*.
2. Partition S into three subcollections:
   - L: elements in S less than m^*.
   - E: elements in S equal to m^*.

L	E	G
$< m^*$	$= m^*$	$> m^*$

Selection algorithms: High-level strategy

Strategy common to the two algorithms:

1. Choose an approximate median m^*.
2. Partition S into three subcollections:
   - L: elements in S less than m^*.
   - E: elements in S equal to m^*.
   - G: elements in S greater than m^*.

L	E	G
$< m^*$	$= m^*$	$> m^*$

Selection algorithms: High-level strategy

Strategy common to the two algorithms:

1. Choose an approximate median m^*.
2. Partition S into three subcollections:
   - L: elements in S less than m^*.
   - E: elements in S equal to m^*.
   - G: elements in S greater than m^*.

L	E	G
$< m^*$	$= m^*$	$> m^*$

3. Recursively select L, E, or G as appropriate:

Selection algorithms: High-level strategy

Strategy common to the two algorithms:

1. Choose an approximate median m^*.
2. Partition S into three subcollections:
   - L: elements in S less than m^*.
   - E: elements in S equal to m^*.
   - G: elements in S greater than m^*.

L	E	G
$< m^*$	$= m^*$	$> m^*$

3. Recursively select L, E, or G as appropriate:

```
if k <= |L|:  return select(L,k)
else if k <= |L|+|E|:  then return m*
else:  return select(G,k-|L|-|E|)
```

Selection algorithms: High-level strategy

Strategy common to the two algorithms:

1. Choose an approximate median m^*.
2. Partition S into three subcollections:
   - L: elements in S less than m^*.
   - E: elements in S equal to m^*.
   - G: elements in S greater than m^*.

L	E	G
$< m^*$	$= m^*$	$> m^*$

3. Recursively select L, E, or G as appropriate:

```
if k <= |L|:  return select(L,k)
else if k <= |L|+|E|:  then return m*
else:  return select(G,k-|L|-|E|)
```

Quickselect and Determinisic Selection choose m^* differently.

QuickSelect

QuickSelect

QuickSelect

QuickSelect

- The approximate median m^* is chosen randomly from S.

QuickSelect

QuickSelect

- The approximate median m^* is chosen randomly from S.

```
def quickSelect(S,k):
    if |S| = 1:  return the (unique) element in S
    choose a random element m* from S
    split S into three sequences:
        L = all elements in S less than m*
        E = all elements in S equal to m*
        G = all elements in S greater than m*
    if k <= |L| then return quickSelect(L,k)
    else if k <= |L|+|E| then return m*
    else return quickSelect(G,k-|L|-|E|)
```

L	E	G
$< m^*$	$= m^*$	$> m^*$

Analysis of QuickSelect

Analysis of QuickSelect

Analysis of QuickSelect

Analysis of QuickSelect

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

Analysis of QuickSelect

Analysis of QuickSelect

- Worst-case running time is $\Theta(n^2)$.
- Expected running time is $O(n)$.

Analysis of QuickSelect

Analysis of QuickSelect

- Worst-case running time is $\Theta(n^2)$.
- Expected running time is $O(n)$.
- Idea behind proof:

Analysis of QuickSelect

Analysis of QuickSelect

- Worst-case running time is $\Theta(n^2)$.
- Expected running time is $O(n)$.
- Idea behind proof:
  - Half the time, at least $1/4$ of the elements are eliminated.

Analysis of QuickSelect

Analysis of QuickSelect

- Worst-case running time is $\Theta(n^2)$.
- Expected running time is $O(n)$.
- Idea behind proof:
  - Half the time, at least $1/4$ of the elements are eliminated.
  - Hence the expected running time can be bounded by a geometric series:

$$ET(n) \leq 2b\left(n + \left(\frac{3}{4}\right)n + \left(\frac{3}{4}\right)^2 n + \cdots\right) = O(n).$$

Here, b is a constant that accounts for the overhead of each recursive call.

Analysis of QuickSelect

Analysis of QuickSelect

- Worst-case running time is $\Theta(n^2)$.
- Expected running time is $O(n)$.
- Idea behind proof:
  - Half the time, at least $1/4$ of the elements are eliminated.
  - Hence the expected running time can be bounded by a geometric series:

$$ET(n) \leq 2b\left(n + \left(\frac{3}{4}\right)n + \left(\frac{3}{4}\right)^2 n + \cdots\right) = O(n).$$

    Here, b is a constant that accounts for the overhead of each recursive call.
  - See [GT], Section 9.2 for details

Analysis of QuickSelect

Analysis of QuickSelect

- Worst-case running time is $\Theta(n^2)$.
- Expected running time is $O(n)$.
- Idea behind proof:
  - Half the time, at least $1/4$ of the elements are eliminated.
  - Hence the expected running time can be bounded by a geometric series:

    $$ET(n) \leq 2b\left(n + \left(\frac{3}{4}\right)n + \left(\frac{3}{4}\right)^2 n + \cdots\right) = O(n).$$

    Here, b is a constant that accounts for the overhead of each recursive call.
  - See [GT], Section 9.2 for details
- Because the algorithm is randomized, no single input elicits the worst-case behavior.

Deterministic Selection

Deterministic Selection

- The approximate median m^* is carefully chosen to guarantee a certain balance condition on the split.

Deterministic Selection

- The approximate median m^* is carefully chosen to guarantee a certain balance condition on the split.
- The precise condition is:

Neither L nor G has more than $\frac{7n}{10} + 3$ elements.

Deterministic Selection Algorithm

Deterministic Selection Algorithm

`DSelect(S,n,k)` returns the k-th smallest element from the list S. The parameter n is the size of S.

Deterministic Selection Algorithm

`DSelect(S,n,k)` returns the k-th smallest element from the list S. The parameter n is the size of S.

1. Divide the n elements of the input set into $\lceil n/5 \rceil$ groups, each consisting of 5 elements (except for possibly the last group, which may have fewer)

Deterministic Selection Algorithm

`DSelect(S,n,k)` returns the k-th smallest element from the list S. The parameter n is the size of S.

1. Divide the n elements of the input set into $\lceil n/5 \rceil$ groups, each consisting of 5 elements (except for possibly the last group, which may have fewer)
2. Find the median of each group directly (e.g., by sorting and taking the middle element)

Deterministic Selection Algorithm

`DSelect(S,n,k)` returns the k-th smallest element from the list S. The parameter n is the size of S.

1. Divide the n elements of the input set into $\lceil n/5 \rceil$ groups, each consisting of 5 elements (except for possibly the last group, which may have fewer)
2. Find the median of each group directly (e.g., by sorting and taking the middle element)
3. Recursively call `DSelect` to find m^*, the median of the $\lceil n/5 \rceil$

Deterministic Selection Algorithm

`DSelect(S,n,k)` returns the k-th smallest element from the list S. The parameter n is the size of S.

1. Divide the n elements of the input set into $\lceil n/5 \rceil$ groups, each consisting of 5 elements (except for possibly the last group, which may have fewer)
2. Find the median of each group directly (e.g., by sorting and taking the middle element)
3. Recursively call `DSelect` to find m^*, the median of the $\lceil n/5 \rceil$ medians found in step 2.
4. Partition into L, E, and G.

L	E	G
$< m^*$	$= m^*$	$> m^*$

Deterministic Selection Algorithm

`DSelect(S,n,k)` returns the k-th smallest element from the list S. The parameter n is the size of S.

1. Divide the n elements of the input set into $\lceil n/5 \rceil$ groups, each consisting of 5 elements (except for possibly the last group, which may have fewer)
2. Find the median of each group directly (e.g., by sorting and taking the middle element)
3. Recursively call `DSelect` to find m^*, the median of the $\lceil n/5 \rceil$ medians found in step 2.
4. Partition into L, E, and G.

L	E	G
$< m^*$	$= m^*$	$> m^*$

5. Either return m^* or recursively call `DSelect` to search in L or G.

Deterministic Selection Algorithm

`DSelect(S,n,k)` returns the k-th smallest element from the list S. The parameter n is the size of S.

1. Divide the n elements of the input set into $\lceil n/5 \rceil$ groups, each consisting of 5 elements (except for possibly the last group, which may have fewer)
2. Find the median of each group directly (e.g., by sorting and taking the middle element)
3. Recursively call `DSelect` to find m^*, the median of the $\lceil n/5 \rceil$ medians found in step 2.
4. Partition into L, E, and G.

L	E	G
$< m^*$	$= m^*$	$> m^*$

5. Either return m^* or recursively call `DSelect` to search in L or G.

There are two recursive calls: One in Step 3 and one in Step 5.

Example

Example

$$S = [\,29, 90, 93, 14, 11, 38, 22, 81, 91, 17, 62, 83, 97, 64, 66, 26, 55, 89,$$
$$51, 24, 32, 78, 66, 47, 61, 76, 82, 70, 85, 73, 15, 99, 86, 13, 88]$$

$n = 35,\ k = 18$

Example

$S = [\,29, 90, 93, 14, 11, 38, 22, 81, 91, 17, 62, 83, 97, 64, 66, 26, 55, 89,$
$51, 24, 32, 78, 66, 47, 61, 76, 82, 70, 85, 73, 15, 99, 86, 13, 88]$
$n = 35,\ k = 18$

1. Divide S into groups of 5:

Example

$S = [\ 29, 90, 93, 14, 11, 38, 22, 81, 91, 17, 62, 83, 97, 64, 66, 26, 55, 89,$
$51, 24, 32, 78, 66, 47, 61, 76, 82, 70, 85, 73, 15, 99, 86, 13, 88]$
$n = 35,\ k = 18$

1. Divide S into groups of 5:

29	38	62	26	32	76	15
90	22	83	55	78	82	99
93	81	97	89	66	70	86
14	91	64	51	47	85	13
11	17	66	24	61	73	88

Example

$S = [\,29, 90, 93, 14, 11, 38, 22, 81, 91, 17, 62, 83, 97, 64, 66, 26, 55, 89,$
$51, 24, 32, 78, 66, 47, 61, 76, 82, 70, 85, 73, 15, 99, 86, 13, 88]$

$n = 35,\ k = 18$

1. Divide S into groups of 5:

29	38	62	26	32	76	15
90	22	83	55	78	82	99
93	81	97	89	66	70	86
14	91	64	51	47	85	13
11	17	66	24	61	73	88

2. Compute median of each group using brute force:

Example

$S = [\,29, 90, 93, 14, 11, 38, 22, 81, 91, 17, 62, 83, 97, 64, 66, 26, 55, 89,$
$51, 24, 32, 78, 66, 47, 61, 76, 82, 70, 85, 73, 15, 99, 86, 13, 88]$

$n = 35,\ k = 18$

1. Divide S into groups of 5:

29	38	62	26	32	76	15
90	22	83	55	78	82	99
93	81	97	89	66	70	86
14	91	64	51	47	85	13
11	17	66	24	61	73	88

2. Compute median of each group using brute force:

$\boxed{29}$	$\boxed{38}$	62	26	32	$\boxed{76}$	15
90	22	83	55	78	82	99
93	81	97	89	66	70	$\boxed{86}$
14	91	64	$\boxed{51}$	47	85	13
11	17	$\boxed{66}$	24	$\boxed{61}$	73	88

Example

$S = [\,29, 90, 93, 14, 11, 38, 22, 81, 91, 17, 62, 83, 97, 64, 66, 26, 55, 89,$
$51, 24, 32, 78, 66, 47, 61, 76, 82, 70, 85, 73, 15, 99, 86, 13, 88]$

$n = 35,\ k = 18$

1. Divide S into groups of 5:

29	38	62	26	32	76	15
90	22	83	55	78	82	99
93	81	97	89	66	70	86
14	91	64	51	47	85	13
11	17	66	24	61	73	88

2. Compute median of each group using brute force:

$\boxed{29}$	$\boxed{38}$	62	26	32	$\boxed{76}$	15
90	22	83	55	78	82	99
93	81	97	89	66	70	$\boxed{86}$
14	91	64	$\boxed{51}$	47	85	13
11	17	$\boxed{66}$	24	$\boxed{61}$	73	88

3. Compute m^* using recursive call:

Example

$$S = [\,29, 90, 93, 14, 11, 38, 22, 81, 91, 17, 62, 83, 97, 64, 66, 26, 55, 89,$$
$$51, 24, 32, 78, 66, 47, 61, 76, 82, 70, 85, 73, 15, 99, 86, 13, 88]$$

$n = 35,\ k = 18$

1. Divide S into groups of 5:

29	38	62	26	32	76	15
90	22	83	55	78	82	99
93	81	97	89	66	70	86
14	91	64	51	47	85	13
11	17	66	24	61	73	88

2. Compute median of each group using brute force:

$\boxed{29}$	$\boxed{38}$	62	26	32	$\boxed{76}$	15
90	22	83	55	78	82	99
93	81	97	89	66	70	$\boxed{86}$
14	91	64	$\boxed{51}$	47	85	13
11	17	$\boxed{66}$	24	$\boxed{61}$	73	88

3. Compute m^* using recursive call:
`Dselect([29,38,66,61,61,76,86],7,4)`

Example

$S = [\,29, 90, 93, 14, 11, 38, 22, 81, 91, 17, 62, 83, 97, 64, 66, 26, 55, 89,$
$51, 24, 32, 78, 66, 47, 61, 76, 82, 70, 85, 73, 15, 99, 86, 13, 88]$

$n = 35,\ k = 18$

1. Divide S into groups of 5:

29	38	62	26	32	76	15
90	22	83	55	78	82	99
93	81	97	89	66	70	86
14	91	64	51	47	85	13
11	17	66	24	61	73	88

2. Compute median of each group using brute force:

$\boxed{29}$	$\boxed{38}$	62	26	32	$\boxed{76}$	15
90	22	83	55	78	82	99
93	81	97	89	66	70	$\boxed{86}$
14	91	64	$\boxed{51}$	47	85	13
11	17	$\boxed{66}$	24	$\boxed{61}$	73	88

3. Compute m^* using recursive call:
`Dselect([29,38,66,61,61,76,86],7,4)` returns 61

Example, continued

$S = [\,29, 90, 93, 14, 11, 38, 22, 81, 91, 17, 62, 83, 97, 64, 66, 26, 55, 89,$
$51, 24, 32, 78, 66, 47, 61, 76, 82, 70, 85, 73, 15, 99, 86, 13, 88]$

$n = 35,\ k = 18$

From previous slide, $m^* = 61$.

Example, continued

$S = [\,29, 90, 93, 14, 11, 38, 22, 81, 91, 17, 62, 83, 97, 64, 66, 26, 55, 89,$
$51, 24, 32, 78, 66, 47, 61, 76, 82, 70, 85, 73, 15, 99, 86, 13, 88]$
$n = 35,\ k = 18$
From previous slide, $m^* = 61$.

4. Partition into L, E, and G:

Example, continued

$S = [\ 29, 90, 93, 14, 11, 38, 22, 81, 91, 17, 62, 83, 97, 64, 66, 26, 55, 89,$
$51, 24, 32, 78, 66, 47, 61, 76, 82, 70, 85, 73, 15, 99, 86, 13, 88]$
$n = 35,\ k = 18$
From previous slide, $m^* = 61$.

4. Partition into L, E, and G:
   $L = [\ 29, 14, 11, 38, 22, 17, 26, 55, 51, 24, 32, 47, 15, 13]$

Example, continued

$S = [\,29, 90, 93, 14, 11, 38, 22, 81, 91, 17, 62, 83, 97, 64, 66, 26, 55, 89,$
$51, 24, 32, 78, 66, 47, 61, 76, 82, 70, 85, 73, 15, 99, 86, 13, 88]$
$n = 35,\ k = 18$
From previous slide, $m^* = 61$.

4. Partition into L, E, and G:
   $L = [\,29, 14, 11, 38, 22, 17, 26, 55, 51, 24, 32, 47, 15, 13]$
   $E = [\,61]$

Example, continued

$S = [\,29, 90, 93, 14, 11, 38, 22, 81, 91, 17, 62, 83, 97, 64, 66, 26, 55, 89,$
$51, 24, 32, 78, 66, 47, 61, 76, 82, 70, 85, 73, 15, 99, 86, 13, 88]$
$n = 35,\ k = 18$
From previous slide, $m^* = 61$.

4. Partition into L, E, and G:
   $L = [\,29, 14, 11, 38, 22, 17, 26, 55, 51, 24, 32, 47, 15, 13]$
   $E = [\,61]$
   $G = [90, 93, 81, 91, 62, 83, 97, 64, 66, 89, 78, 66, 76, 82, 70, 85, 73, 99, 86, 88]$

Example, continued

$S = [\,29, 90, 93, 14, 11, 38, 22, 81, 91, 17, 62, 83, 97, 64, 66, 26, 55, 89,$
$51, 24, 32, 78, 66, 47, 61, 76, 82, 70, 85, 73, 15, 99, 86, 13, 88]$
$n = 35,\ k = 18$
From previous slide, $m^* = 61$.

4. Partition into L, E, and G:
   $L = [\,29, 14, 11, 38, 22, 17, 26, 55, 51, 24, 32, 47, 15, 13]$
   $E = [\,61]$
   $G = [90, 93, 81, 91, 62, 83, 97, 64, 66, 89, 78, 66, 76, 82, 70, 85, 73, 99, 86, 88]$
   $|L| = 14,\ |E| = 1, |G| = 20$

Example, continued

$S = [\,29, 90, 93, 14, 11, 38, 22, 81, 91, 17, 62, 83, 97, 64, 66, 26, 55, 89,$
$\quad 51, 24, 32, 78, 66, 47, 61, 76, 82, 70, 85, 73, 15, 99, 86, 13, 88]$
$n = 35,\ k = 18$
From previous slide, $m^* = 61$.

4. Partition into L, E, and G:
   $L = [\,29, 14, 11, 38, 22, 17, 26, 55, 51, 24, 32, 47, 15, 13]$
   $E = [\,61]$
   $G = [90, 93, 81, 91, 62, 83, 97, 64, 66, 89, 78, 66, 76, 82, 70, 85, 73, 99, 86, 88]$
   $|L| = 14,\ |E| = 1, |G| = 20$
5. Compute the result using a recursive call:
   `DSelect(G,20,3)`

Example, continued

$S = [\,29, 90, 93, 14, 11, 38, 22, 81, 91, 17, 62, 83, 97, 64, 66, 26, 55, 89,$
$51, 24, 32, 78, 66, 47, 61, 76, 82, 70, 85, 73, 15, 99, 86, 13, 88]$

$n = 35,\ k = 18$

From previous slide, $m^* = 61$.

4. Partition into L, E, and G:

   $L = [\,29, 14, 11, 38, 22, 17, 26, 55, 51, 24, 32, 47, 15, 13]$

   $E = [\,61]$

   $G = [90, 93, 81, 91, 62, 83, 97, 64, 66, 89, 78, 66, 76, 82, 70, 85, 73, 99, 86, 88]$

   $|L| = 14,\ |E| = 1, |G| = 20$

5. Compute the result using a recursive call:

   `DSelect(G,20,3)`

   (Because $k - |L| - |E| = 18 - 14 - 1 = 3$)

Example, continued

$S = [\,29, 90, 93, 14, 11, 38, 22, 81, 91, 17, 62, 83, 97, 64, 66, 26, 55, 89,$
$51, 24, 32, 78, 66, 47, 61, 76, 82, 70, 85, 73, 15, 99, 86, 13, 88]$
$n = 35,\ k = 18$
From previous slide, $m^* = 61$.

4. Partition into L, E, and G:

   $L = [\,29, 14, 11, 38, 22, 17, 26, 55, 51, 24, 32, 47, 15, 13]$

   $E = [\,61]$

   $G = [90, 93, 81, 91, 62, 83, 97, 64, 66, 89, 78, 66, 76, 82, 70, 85, 73, 99, 86, 88]$

   $|L| = 14,\ |E| = 1, |G| = 20$

5. Compute the result using a recursive call:

   `DSelect(G,20,3)`

   (Because $k - |L| - |E| = 18 - 14 - 1 = 3$)

Again, note that there are two recursive calls: one in Step 3, and one in step 5

Pseudocode for Deterministic Selection

```
def DSelect(S,k,n):// Parameter n is not really necessary
    if n ≤ 5:  return(bruteForceSelect(S,k,n))
    // Step 1:  Divide into groups of 5
    for i = 0 to n-1:  group[floor(i/5)][mod(i,5)] = S[i])
    // Step 2:  Compute median of each group oof 5 non-recursively
    for i = 1 to ceil(n/5):
        groupMedian[i] = bruteForceSelect({group[i][j]:j=0..4},3,5)
    // Step 3:  Compute m* = median of group medians
    m* = DSelect(groupMedian,ceil(ceil(n/5)/2),ceil(n/5))
    // Step 4:  Partition elements of S in L, E, G
    Allocate three empty lists L, E, and G
    for i = 0 to n-1:
        if S[i] < m*:  L = L + {S[i]}
        else if S[i] = m*:  E = E + {S[i]}
        else:  G = G + {S[i]} // S[i] > m*
    // Step 5:  Return m* or recursively call DSelect in L or G
    if k ≤ |L|:  return DSelect(L,k,|L|)
    else if k ≤ |L| + |E|:  return m*
    else:  return DSelect(G,k-|L|-|E|,|G|)
```

Analysis of Deterministic Selection Algorithm

Analysis of Deterministic Selection Algorithm

We first show that neither L nor G has more than $\frac{7n}{10} + 3$ items.

Analysis of Deterministic Selection Algorithm

We first show that neither L nor G has more than $\frac{7n}{10} + 3$ items.We will show this for G.

Analysis of Deterministic Selection Algorithm

We first show that neither L nor G has more than $\frac{7n}{10} + 3$ items.We will show this for G.

To get an upper bound on the size of G, we show a lower bound on the size of the complement of G.

Analysis of Deterministic Selection Algorithm

We first show that neither L nor G has more than $\frac{7n}{10} + 3$ items.We will show this for G.

To get an upper bound on the size of G, we show a lower bound on the size of the complement of G.

When we compute m^*:

Analysis of Deterministic Selection Algorithm

We first show that neither L nor G has more than $\frac{7n}{10} + 3$ items.We will show this for G.

To get an upper bound on the size of G, we show a lower bound on the size of the complement of G.

When we compute m^*:

- There are $\lceil \frac{n}{5} \rceil$ groups

Analysis of Deterministic Selection Algorithm

We first show that neither L nor G has more than $\frac{7n}{10} + 3$ items.We will show this for G.

To get an upper bound on the size of G, we show a lower bound on the size of the complement of G.

When we compute m^*:

- There are $\left\lceil \frac{n}{5} \right\rceil$ groups
- There are at least $\left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil$ groups with median $\leq m^*$.

Analysis of Deterministic Selection Algorithm

We first show that neither L nor G has more than $\frac{7n}{10} + 3$ items.We will show this for G.

To get an upper bound on the size of G, we show a lower bound on the size of the complement of G.

When we compute m^*:

- There are $\left\lceil \frac{n}{5} \right\rceil$ groups
- There are at least $\left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil$ groups with median $\leq m^*$.
- There are at least $\left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil - 1$ full groups with median $\leq m^*$.

Analysis of Deterministic Selection Algorithm

We first show that neither L nor G has more than $\frac{7n}{10} + 3$ items.We will show this for G.

To get an upper bound on the size of G, we show a lower bound on the size of the complement of G.

When we compute m^*:

- There are $\left\lceil \frac{n}{5} \right\rceil$ groups
- There are at least $\left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil$ groups with median $\leq m^*$.
- There are at least $\left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil - 1$ full groups with median $\leq m^*$. Each of these groups has at least 3 items $\leq m^*$.

Analysis of Deterministic Selection Algorithm

We first show that neither L nor G has more than $\frac{7n}{10} + 3$ items.We will show this for G.

To get an upper bound on the size of G, we show a lower bound on the size of the complement of G.

When we compute m^*:

- There are $\left\lceil \frac{n}{5} \right\rceil$ groups
- There are at least $\left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil$ groups with median $\leq m^*$.
- There are at least $\left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil - 1$ full groups with median $\leq m^*$. Each of these groups has at least 3 items $\leq m^*$.
- It follows that the number of items not in G is at least:

Analysis of Deterministic Selection Algorithm

We first show that neither L nor G has more than $\frac{7n}{10}+3$ items.We will show this for G.

To get an upper bound on the size of G, we show a lower bound on the size of the complement of G.

When we compute m^*:

- There are $\left\lceil \frac{n}{5} \right\rceil$ groups
- There are at least $\left\lceil \frac{1}{2}\left\lceil \frac{n}{5} \right\rceil \right\rceil$ groups with median $\leq m^*$.
- There are at least $\left\lceil \frac{1}{2}\left\lceil \frac{n}{5} \right\rceil \right\rceil - 1$ full groups with median $\leq m^*$. Each of these groups has at least 3 items $\leq m^*$.
- It follows that the number of items not in G is at least:

$$3\left(\left\lceil \frac{1}{2}\left\lceil \frac{n}{5} \right\rceil \right\rceil - 1\right)$$

Analysis of Deterministic Selection Algorithm

We first show that neither L nor G has more than $\frac{7n}{10} + 3$ items.We will show this for G.

To get an upper bound on the size of G, we show a lower bound on the size of the complement of G.

When we compute m^*:

- There are $\lceil \frac{n}{5} \rceil$ groups
- There are at least $\lceil \frac{1}{2} \lceil \frac{n}{5} \rceil \rceil$ groups with median $\leq m^*$.
- There are at least $\lceil \frac{1}{2} \lceil \frac{n}{5} \rceil \rceil - 1$ full groups with median $\leq m^*$. Each of these groups has at least 3 items $\leq m^*$.
- It follows that the number of items not in G is at least:

$$3\left(\left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil - 1\right) \geq 3\left(\frac{n}{10} - 1\right)$$

Analysis of Deterministic Selection Algorithm

We first show that neither L nor G has more than $\frac{7n}{10}+3$ items.We will show this for G.

To get an upper bound on the size of G, we show a lower bound on the size of the complement of G.

When we compute m^*:

- There are $\left\lceil \frac{n}{5} \right\rceil$ groups
- There are at least $\left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil$ groups with median $\leq m^*$.
- There are at least $\left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil - 1$ full groups with median $\leq m^*$. Each of these groups has at least 3 items $\leq m^*$.
- It follows that the number of items not in G is at least:

$$3\left(\left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil - 1\right) \geq 3\left(\frac{n}{10} - 1\right) = \frac{3n}{10} - 3.$$

Analysis of Deterministic Selection Algorithm

We first show that neither L nor G has more than $\frac{7n}{10} + 3$ items.We will show this for G.

To get an upper bound on the size of G, we show a lower bound on the size of the complement of G.

When we compute m^*:

- There are $\left\lceil \frac{n}{5} \right\rceil$ groups
- There are at least $\left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil$ groups with median $\leq m^*$.
- There are at least $\left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil - 1$ full groups with median $\leq m^*$. Each of these groups has at least 3 items $\leq m^*$.
- It follows that the number of items not in G is at least:
$$3\left(\left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil - 1\right) \geq 3\left(\frac{n}{10} - 1\right) = \frac{3n}{10} - 3.$$
- Hence, the number of elements that are in G satisfies the inequality:

Analysis of Deterministic Selection Algorithm

We first show that neither L nor G has more than $\frac{7n}{10} + 3$ items.We will show this for G.

To get an upper bound on the size of G, we show a lower bound on the size of the complement of G.

When we compute m^*:

- There are $\left\lceil \frac{n}{5} \right\rceil$ groups
- There are at least $\left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil$ groups with median $\leq m^*$.
- There are at least $\left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil - 1$ full groups with median $\leq m^*$. Each of these groups has at least 3 items $\leq m^*$.
- It follows that the number of items not in G is at least:

$$3\left(\left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil - 1\right) \geq 3\left(\frac{n}{10} - 1\right) = \frac{3n}{10} - 3.$$

- Hence, the number of elements that are in G satisfies the inequality:

$$|G| \leq n - \left(\frac{3n}{10} - 3\right)$$

Analysis of Deterministic Selection Algorithm

We first show that neither L nor G has more than $\frac{7n}{10} + 3$ items.We will show this for G.

To get an upper bound on the size of G, we show a lower bound on the size of the complement of G.

When we compute m^*:

- There are $\left\lceil \frac{n}{5} \right\rceil$ groups
- There are at least $\left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil$ groups with median $\leq m^*$.
- There are at least $\left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil - 1$ full groups with median $\leq m^*$. Each of these groups has at least 3 items $\leq m^*$.
- It follows that the number of items not in G is at least:
$$3\left(\left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil - 1\right) \geq 3\left(\frac{n}{10} - 1\right) = \frac{3n}{10} - 3.$$
- Hence, the number of elements that are in G satisfies the inequality:
$$|G| \leq n - \left(\frac{3n}{10} - 3\right) = \frac{7n}{10} + 3.$$

Analysis of Deterministic Selection Algorithm

We first show that neither L nor G has more than $\frac{7n}{10} + 3$ items.We will show this for G.

To get an upper bound on the size of G, we show a lower bound on the size of the complement of G.

When we compute m^*:

- There are $\left\lceil \frac{n}{5} \right\rceil$ groups
- There are at least $\left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil$ groups with median $\leq m^*$.
- There are at least $\left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil - 1$ full groups with median $\leq m^*$. Each of these groups has at least 3 items $\leq m^*$.
- It follows that the number of items not in G is at least:
$$3\left(\left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil - 1\right) \geq 3\left(\frac{n}{10} - 1\right) = \frac{3n}{10} - 3.$$
- Hence, the number of elements that are in G satisfies the inequality:
$$|G| \leq n - \left(\frac{3n}{10} - 3\right) = \frac{7n}{10} + 3.$$

Analysis of Deterministic Selection Algorithm

Analysis of Deterministic Selection Algorithm

Let $T(n)$ =worst-case time required by `DSelect` on an input of size n.

Analysis of Deterministic Selection Algorithm

Let $T(n)$ =worst-case time required by `DSelect` on an input of size n.

Cost of each step:

Analysis of Deterministic Selection Algorithm

Let $T(n)$ =worst-case time required by `DSelect` on an input of size n.

Cost of each step:

1. Divide into groups of 5:

Analysis of Deterministic Selection Algorithm

Let $T(n)$ =worst-case time required by `DSelect` on an input of size n.

Cost of each step:

1. Divide into groups of 5: $O(n)$

Analysis of Deterministic Selection Algorithm

Let $T(n)$ =worst-case time required by `DSelect` on an input of size n.

Cost of each step:

1. Divide into groups of 5: $O(n)$
2. Compute group medians:

Analysis of Deterministic Selection Algorithm

Let $T(n)$ =worst-case time required by `DSelect` on an input of size n.

Cost of each step:

1. Divide into groups of 5: $O(n)$
2. Compute group medians: $O(n)$ ($O(1)$ each, performed $\lceil n/5 \rceil$ times)

Analysis of Deterministic Selection Algorithm

Let $T(n)$ =worst-case time required by `DSelect` on an input of size n.

Cost of each step:

1. Divide into groups of 5: $O(n)$
2. Compute group medians: $O(n)$ ($O(1)$ each, performed $\lceil n/5 \rceil$ times)
3. Recursive call to compute m^*:

Analysis of Deterministic Selection Algorithm

Let $T(n)$ =worst-case time required by `DSelect` on an input of size n.

Cost of each step:

1. Divide into groups of 5: $O(n)$
2. Compute group medians: $O(n)$ ($O(1)$ each, performed $\lceil n/5 \rceil$ times)
3. Recursive call to compute m^*: $T\left(\lceil \frac{n}{5} \rceil\right)$

Analysis of Deterministic Selection Algorithm

Let $T(n) =$worst-case time required by `DSelect` on an input of size n.

Cost of each step:

1. Divide into groups of 5: $O(n)$
2. Compute group medians: $O(n)$ ($O(1)$ each, performed $\lceil n/5 \rceil$ times)
3. Recursive call to compute m^*: $T\left(\lceil \frac{n}{5} \rceil\right)$
4. Partition into L, E, and G:

Analysis of Deterministic Selection Algorithm

Let $T(n)$ =worst-case time required by `DSelect` on an input of size n.

Cost of each step:

1. Divide into groups of 5: $O(n)$
2. Compute group medians: $O(n)$ ($O(1)$ each, performed $\lceil n/5 \rceil$ times)
3. Recursive call to compute m^*: $T\left(\left\lceil \frac{n}{5} \right\rceil\right)$
4. Partition into L, E, and G: $O(n)$

Analysis of Deterministic Selection Algorithm

Let $T(n) =$worst-case time required by `DSelect` on an input of size n.

Cost of each step:

1. Divide into groups of 5: $O(n)$
2. Compute group medians: $O(n)$ ($O(1)$ each, performed $\lceil n/5 \rceil$ times)
3. Recursive call to compute m^*: $T\left(\lceil \frac{n}{5} \rceil\right)$
4. Partition into L, E, and G: $O(n)$
5. Final recursive call:

Analysis of Deterministic Selection Algorithm

Let $T(n)$ =worst-case time required by `DSelect` on an input of size n.

Cost of each step:

1. Divide into groups of 5: $O(n)$
2. Compute group medians: $O(n)$ ($O(1)$ each, performed $\lceil n/5 \rceil$ times)
3. Recursive call to compute m^*: $T\left(\lceil \frac{n}{5} \rceil\right)$
4. Partition into L, E, and G: $O(n)$
5. Final recursive call: $\leq T\left(\frac{7n}{10} + 3\right)$

Analysis of Deterministic Selection Algorithm

Let $T(n)$ =worst-case time required by `DSelect` on an input of size n.

Cost of each step:

1. Divide into groups of 5: $O(n)$
2. Compute group medians: $O(n)$ ($O(1)$ each, performed $\lceil n/5 \rceil$ times)
3. Recursive call to compute m^*: $T\left(\lceil \frac{n}{5} \rceil\right)$
4. Partition into L, E, and G: $O(n)$
5. Final recursive call: $\leq T\left(\frac{7n}{10} + 3\right)$

So we get the recurrence equation:

$$T(n) \leq T\left(\left\lceil \frac{n}{5} \right\rceil\right) + T\left(\frac{7n}{10} + 3\right)$$

Analysis of Deterministic Selection Algorithm

$$T(n) \le T\left(\left\lceil \frac{n}{5} \right\rceil\right) + T\left(\frac{7n}{10} + 3\right) + O(n)$$

Analysis of Deterministic Selection Algorithm

$$T(n) \leq T\left(\left\lceil \frac{n}{5} \right\rceil\right) + T\left(\frac{7n}{10} + 3\right) + O(n)$$

- This is not a standard divide-and-conquer recurrence, and we cannot use the Master Method or the Alternative Method to solve it.

Analysis of Deterministic Selection Algorithm

$$T(n) \leq T\left(\left\lceil \frac{n}{5} \right\rceil\right) + T\left(\frac{7n}{10} + 3\right) + O(n)$$

- This is not a standard divide-and-conquer recurrence, and we cannot use the Master Method or the Alternative Method to solve it.
- The solution is $T(n) = O(n)$.

Analysis of Deterministic Selection Algorithm

$$T(n) \leq T\left(\left\lceil \frac{n}{5} \right\rceil\right) + T\left(\frac{7n}{10} + 3\right) + O(n)$$

- This is not a standard divide-and-conquer recurrence, and we cannot use the Master Method or the Alternative Method to solve it.
- The solution is $T(n) = O(n)$.
- Intuitively, the reason why it is $O(n)$ is because

Analysis of Deterministic Selection Algorithm

$$T(n) \leq T\left(\left\lceil \frac{n}{5} \right\rceil\right) + T\left(\frac{7n}{10} + 3\right) + O(n)$$

- This is not a standard divide-and-conquer recurrence, and we cannot use the Master Method or the Alternative Method to solve it.
- The solution is $T(n) = O(n)$.
- Intuitively, the reason why it is $O(n)$ is because
  - The total size of the two recursive calls is $\leq c \cdot n$ for some constant $c < 1$; and

Analysis of Deterministic Selection Algorithm

$$T(n) \leq T\left(\left\lceil \frac{n}{5} \right\rceil\right) + T\left(\frac{7n}{10} + 3\right) + O(n)$$

- This is not a standard divide-and-conquer recurrence, and we cannot use the Master Method or the Alternative Method to solve it.
- The solution is $T(n) = O(n)$.
- Intuitively, the reason why it is $O(n)$ is because
  - The total size of the two recursive calls is $\leq c \cdot n$ for some constant $c < 1$; and
  - The time required to set up and process the recursions is $O(n)$.

Analysis of Deterministic Selection Algorithm

$$T(n) \leq T\left(\left\lceil \frac{n}{5} \right\rceil\right) + T\left(\frac{7n}{10} + 3\right) + O(n)$$

- This is not a standard divide-and-conquer recurrence, and we cannot use the Master Method or the Alternative Method to solve it.
- The solution is $T(n) = O(n)$.
- Intuitively, the reason why it is $O(n)$ is because
  - The total size of the two recursive calls is $\leq c \cdot n$ for some constant $c < 1$; and
  - The time required to set up and process the recursions is $O(n)$.

  So it is "similar to" $T(n) = T\left(\frac{9}{10}\right) + O(n)$.

Analysis of Deterministic Selection Algorithm

$$T(n) \le T\left(\left\lceil \frac{n}{5} \right\rceil\right) + T\left(\frac{7n}{10} + 3\right) + O(n)$$

- This is not a standard divide-and-conquer recurrence, and we cannot use the Master Method or the Alternative Method to solve it.
- The solution is $T(n) = O(n)$.
- Intuitively, the reason why it is $O(n)$ is because
  - The total size of the two recursive calls is $\le c \cdot n$ for some constant $c < 1$; and
  - The time required to set up and process the recursions is $O(n)$.

So it is "similar to" $T(n) = T\left(\frac{9}{10}\right) + O(n)$. But this is not a proof.

Analysis of Deterministic Selection Algorithm

$$T(n) \leq T\left(\left\lceil \frac{n}{5} \right\rceil\right) + T\left(\frac{7n}{10} + 3\right) + O(n)$$

- This is not a standard divide-and-conquer recurrence, and we cannot use the Master Method or the Alternative Method to solve it.
- The solution is $T(n) = O(n)$.
- Intuitively, the reason why it is $O(n)$ is because
  - The total size of the two recursive calls is $\leq c \cdot n$ for some constant $c < 1$; and
  - The time required to set up and process the recursions is $O(n)$.

  So it is "similar to" $T(n) = T\left(\frac{9}{10}\right) + O(n)$. But this is not a proof.
- A proof that the solution is $T(n) = O(n)$ is given in [GT, Section 9.2].

Analysis of Deterministic Selection Algorithm

$$T(n) \leq T\left(\left\lceil \frac{n}{5} \right\rceil\right) + T\left(\frac{7n}{10} + 3\right) + O(n)$$

- This is not a standard divide-and-conquer recurrence, and we cannot use the Master Method or the Alternative Method to solve it.
- The solution is $T(n) = O(n)$.
- Intuitively, the reason why it is $O(n)$ is because
  - The total size of the two recursive calls is $\leq c \cdot n$ for some constant $c < 1$; and
  - The time required to set up and process the recursions is $O(n)$.

  So it is "similar to" $T(n) = T\left(\frac{9}{10}\right) + O(n)$. But this is not a proof.
- A proof that the solution is $T(n) = O(n)$ is given in [GT, Section 9.2].
  - The value of the hidden constant given there is 330.

Analysis of Deterministic Selection Algorithm

$$T(n) \le T\left(\left\lceil \frac{n}{5} \right\rceil\right) + T\left(\frac{7n}{10} + 3\right) + O(n)$$

- This is not a standard divide-and-conquer recurrence, and we cannot use the Master Method or the Alternative Method to solve it.
- The solution is $T(n) = O(n)$.
- Intuitively, the reason why it is $O(n)$ is because
  - The total size of the two recursive calls is $\le c \cdot n$ for some constant $c < 1$; and
  - The time required to set up and process the recursions is $O(n)$.

  So it is "similar to" $T(n) = T\left(\frac{9}{10}\right) + O(n)$. But this is not a proof.
- A proof that the solution is $T(n) = O(n)$ is given in [GT, Section 9.2].
  - The value of the hidden constant given there is 330.
  - This is not necessarily the best possible constant, but it does suggest that `DSelect` is considerably slower than `QuickSelect`.

Final remarks on Selection Algorithms

Final remarks on Selection Algorithms

- Selection can be solved in $O(n)$ time.

Final remarks on Selection Algorithms

- Selection can be solved in $O(n)$ time.
- For most practical purposes, `QuickSelect` is preferable:

Final remarks on Selection Algorithms

- Selection can be solved in $O(n)$ time.
- For most practical purposes, `QuickSelect` is preferable:
  - Runs in expected linear time

Final remarks on Selection Algorithms

- Selection can be solved in $O(n)$ time.
- For most practical purposes, `QuickSelect` is preferable:
  - Runs in expected linear time
  - Simple algorithm, hidden constant is small

Final remarks on Selection Algorithms

- Selection can be solved in $O(n)$ time.
- For most practical purposes, `QuickSelect` is preferable:
  - Runs in expected linear time
  - Simple algorithm, hidden constant is small
- `DSelect`

Final remarks on Selection Algorithms

- Selection can be solved in $O(n)$ time.
- For most practical purposes, `QuickSelect` is preferable:
  - Runs in expected linear time
  - Simple algorithm, hidden constant is small
- `DSelect`
  - Shows that selection can be solved in worst-case linear time.

Final remarks on Selection Algorithms

- Selection can be solved in $O(n)$ time.
- For most practical purposes, `QuickSelect` is preferable:
  - Runs in expected linear time
  - Simple algorithm, hidden constant is small
- `DSelect`
  - Shows that selection can be solved in worst-case linear time.
  - Slower in practice.

Final remarks on Selection Algorithms

- Selection can be solved in $O(n)$ time.
- For most practical purposes, `QuickSelect` is preferable:
  - Runs in expected linear time
  - Simple algorithm, hidden constant is small
- `DSelect`
  - Shows that selection can be solved in worst-case linear time.
  - Slower in practice.
  - Might be useful in situations where there is a time bound that cannot be exceeded