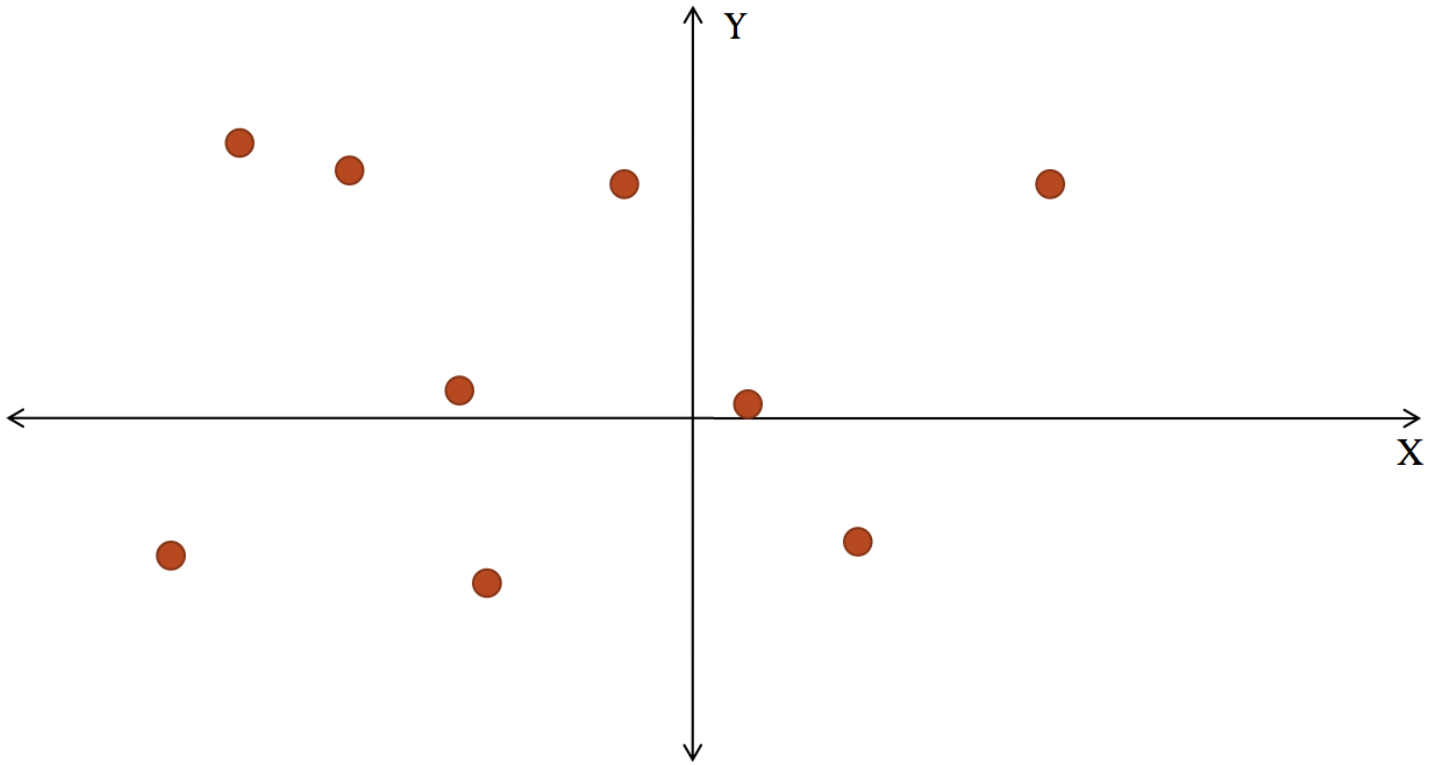
COL351: Analysis and Design of Algorithms

Ragesh Jaiswal, CSE, IITD

Divide and Conquer

Problem

You are given n points on a two dimensional plane. Each point i is defined by a pair $(x(i), y(i))$ of coordinates. Design an algorithm that outputs the closest pair of points.



Problem

You are given n points on a two dimensional plane. Each point i is defined by a pair $(x(i), y(i))$ of coordinates. Design an algorithm that outputs the closest pair of points.

- Brute-force algorithm: Consider all pairs and pick closest.
  - Running time:

Problem

You are given n points on a two dimensional plane. Each point i is defined by a pair $(x(i), y(i))$ of coordinates. Design an algorithm that outputs the closest pair of points.

- Brute-force algorithm: Consider all pairs and pick closest.
  - Running time: $O(n^2)$

Problem

You are given n points on a two dimensional plane. Each point i is defined by a pair $(x(i), y(i))$ of coordinates. Design an algorithm that outputs the closest pair of points.

- Divide and Conquer: *(Divide based on X-axis)*
  - Consider the *left-half* points P_L and *right-half* points P_R.

Problem

You are given n points on a two dimensional plane. Each point i is defined by a pair $(x(i), y(i))$ of coordinates. Design an algorithm that outputs the closest pair of points.

- Divide and Conquer: *(Divide based on X-axis)*
  - Consider the *left-half* points P_L and *right-half* points P_R.
  - Recursively find the closest pair of points (i_l, j_L) in P_L, and (i_R, j_R) in P_R.

Problem

You are given n points on a two dimensional plane. Each point i is defined by a pair $(x(i), y(i))$ of coordinates. Design an algorithm that outputs the closest pair of points.

- Divide and Conquer: *(Divide based on X-axis)*
  - Consider the *left-half* points P_L and *right-half* points P_R.
  - Recursively find the closest pair of points (i_l, j_L) in P_L, and (i_R, j_R) in P_R.
  - Consider all pair of points (p, q) such that p belongs to P_L and q belongs to P_R.

Problem

You are given n points on a two dimensional plane. Each point i is defined by a pair $(x(i), y(i))$ of coordinates. Design an algorithm that outputs the closest pair of points.

- Divide and Conquer: *(Divide based on X-axis)*
  - Consider the *left-half* points P_L and *right-half* points P_R.
  - Recursively find the closest pair of points (i_l, j_L) in P_L, and (i_R, j_R) in P_R.
  - Consider all pair of points (p, q) such that p belongs to P_L and q belongs to P_R.
  - Pick the closest pair among $(i_L, j_L), (i_R, j_R)$, and (p, q).

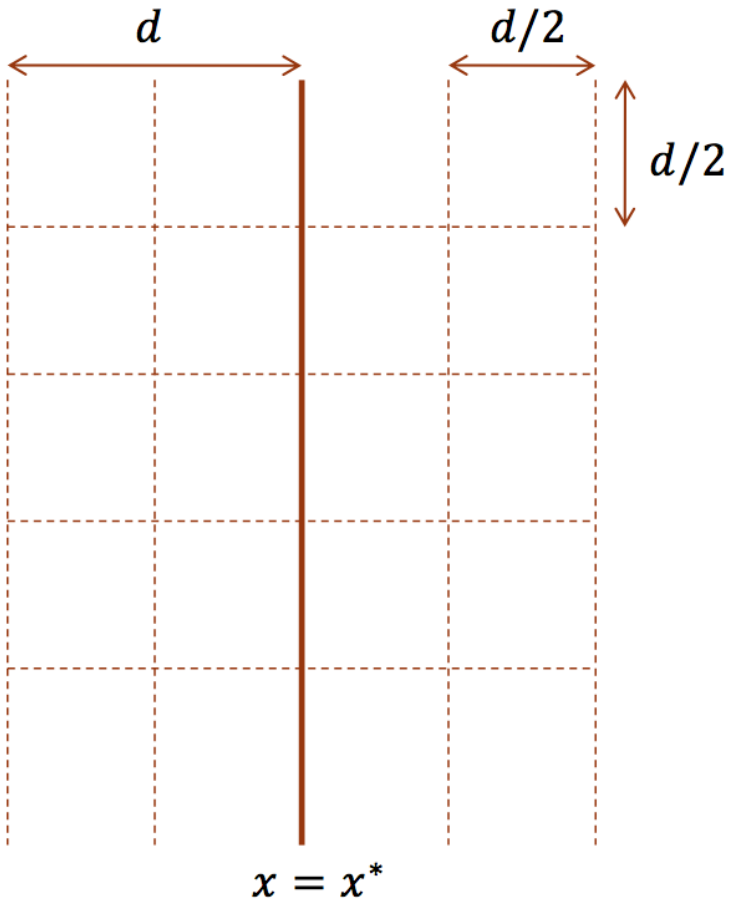
Problem

You are given n points on a two dimensional plane. Each point i is defined by a pair $(x(i), y(i))$ of coordinates. Design an algorithm that outputs the closest pair of points.

- Divide and Conquer: *(Divide based on X-axis)*
  - Consider the *left-half* points P_L and *right-half* points P_R.
  - Recursively find the closest pair of points (i_l, j_L) in P_L, and (i_R, j_R) in P_R.
  - Consider all pair of points (p, q) such that p belongs to P_L and q belongs to P_R.
  - Pick the closest pair among $(i_L, j_L), (i_R, j_R)$, and (p, q).
- What is the running time of the above algorithm?

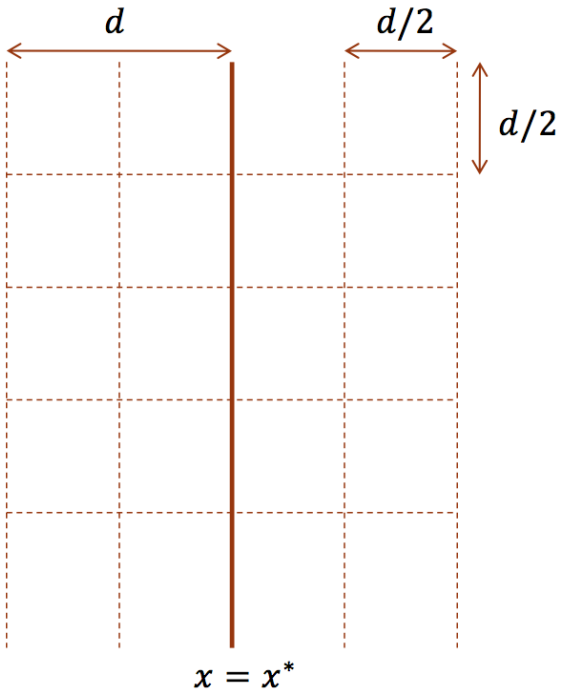
- Divide and Conquer: *(Divide based on X-axis)*
  - Consider the *left-half* points P_L and *right-half* points P_R.
  - Recursively find the closest pair of points (i_l, j_L) in P_L, and (i_R, j_R) in P_R.
  - Consider all pair of points (p, q) such that p belongs to P_L and q belongs to P_R.
  - Pick the closest pair among $(i_L, j_L), (i_R, j_R)$, and (p, q).
- Let $x = x^*$ be a line along the Y-axis dividing the points into P_L and P_R.
- Let d be the distance between the closest pair of points in P_L and P_R.
- Claim 1: For any pair of points (p, q) such that $x(p) < x^* - d$ and $x(q) \geq x^*$, the distance between p and q is $\geq d$.
- Claim 2: For any pair of points (p, q) such that $x(p) \leq x^*$ and $x(q) > x^* + d$, the distance between p and q is $\geq d$.

- Divide and Conquer: *(Divide based on X-axis)*
  - Consider the *left-half* points P_L and *right-half* points P_R.
  - Recursively find the closest pair of points (i_l, j_L) in P_L, and (i_R, j_R) in P_R.
  - Consider all pair of points (p, q) such that p belongs to P_L and q belongs to P_R.
  - Pick the closest pair among $(i_L, j_L), (i_R, j_R)$, and (p, q).
- Let $x = x^*$ be a line along the Y-axis dividing the points into P_L and P_R.
- Let d be the distance between the closest pair of points in P_L and P_R.
- Claim 1: For any pair of points (p, q) such that $x(p) < x^* - d$ and $x(q) \geq x^*$, the distance between p and q is $\geq d$.
- Claim 2: For any pair of points (p, q) such that $x(p) \leq x^*$ and $x(q) > x^* + d$, the distance between p and q is $\geq d$.
- This means that for pairs of points across the line $x = x^*$, we can throw any point in P_L that has small X-coordinate and any point in P_R that has large X-coordinate.
- Do these claims help in improving the running time?



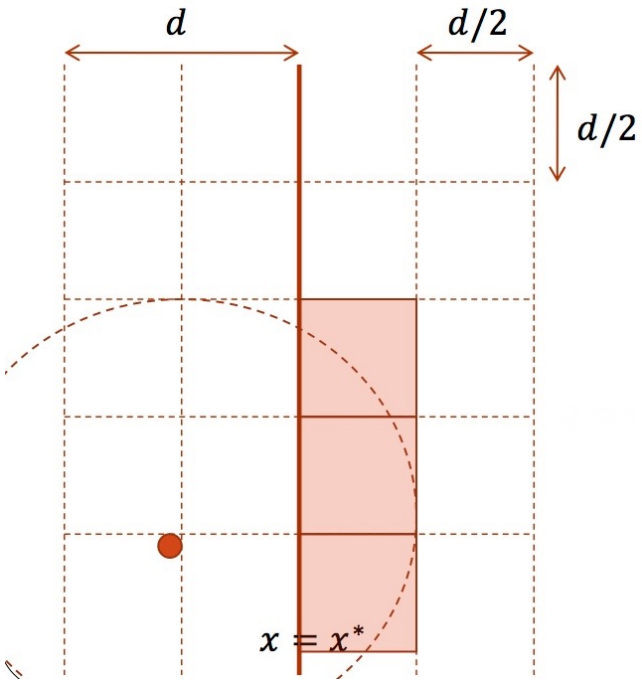
- How many points are there in each "box"?

- Claim 3: Let P be all the points that have X-coordinate between $(x^* - d)$ and $(x^* + d)$. Let S be the sorted list of points in P sorted in increasing order of their Y-coordinates. Consider any pair of points (p, q) such that p belongs to P_L and q belongs to P_R and the distance between p and q is at most d. Then there cannot be more than 10 points between p and q in the sorted list S.



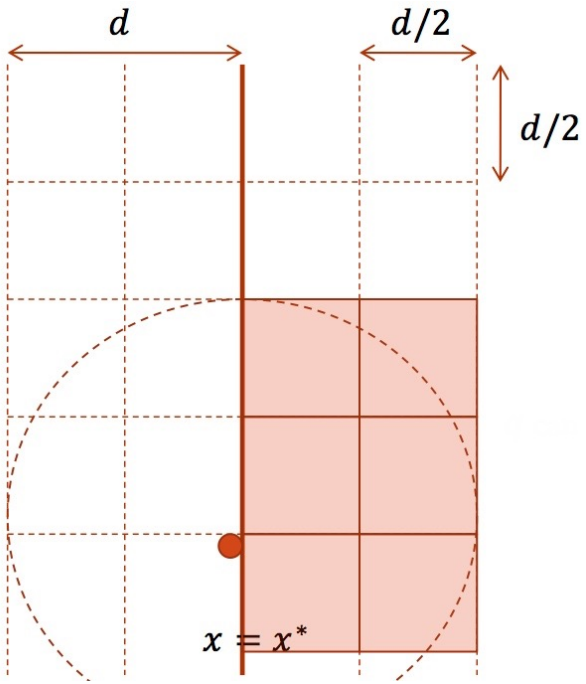
- Consider a pair (p, q) such that p belongs to P_L and q belongs to P_R and distance between p and q is at most d.
- Let $y(p) \leq y(q)$. The case $y(q) < y(p)$ will be symmetric.

- Claim 3: Let P be all the points that have X-coordinate between $(x^* - d)$ and $(x^* + d)$. Let S be the sorted list of points in P sorted in increasing order of their Y-coordinates. Consider any pair of points (p, q) such that p belongs to P_L and q belongs to P_R and the distance between p and q is at most d. Then there cannot be more than 10 points between p and q in the sorted list S.



- Consider a pair (p, q) such that p belongs to P_L and q belongs to P_R and distance between p and q is at most d.
- Let $y(p) \leq y(q)$. The case $y(q) < y(p)$ will be symmetric.
- q can only belong to one of the shaded boxes.

- Claim 3: Let P be all the points that have X-coordinate between $(x^* - d)$ and $(x^* + d)$. Let S be the sorted list of points in P sorted in increasing order of their Y-coordinates. Consider any pair of points (p, q) such that p belongs to P_L and q belongs to P_R and the distance between p and q is at most d. Then there cannot be more than 10 points between p and q in the sorted list S.



- Consider a pair (p, q) such that p belongs to P_L and q belongs to P_R and distance between p and q is at most d.
- Let $y(p) \leq y(q)$. The case $y(q) < y(p)$ will be symmetric.
- q can only belong to one of the shaded boxes.

Algorithm

`ClosestPair(`P`)`

- ... *//Base cases*
- Sort the points in increasing order of X-coordinates and pick the median point (x^*, y)
- Partition P into P_L *(all point p with $x(p) < x^*$)* and P_R *(all points with $x(p) \geq x^*$)*
- Let $(p_L, q_L) \leftarrow$ `ClosestPair(`P_L`)`
- Let $(p_R, q_R) \leftarrow$ `ClosestPair(`P_R`)`
- Let (p, q) be the pair (among (p_L, q_L) and (p_R, q_R)) with the smaller distance and let d be this distance
- Let S be the sorted list of points with X-coordinate between $(x^* - d)$ and $(x^* + d)$
- For $i = 1$ to $|S|$
  - For $j = 1$ to 11
    - If $distance(S[i], S[i + j]) < d$
      - $(p, q) \leftarrow (S[i], S[i + j])$
      - $d \leftarrow distance(S[i], S[i + j])$
- Output(p, q)

- What is the running time of the above algorithm?

Algorithm

`ClosestPair(`P`)`

- ... *//Base cases*
- Sort the points in increasing order of X-coordinates and pick the median point (x^*, y)
- Partition P into P_L *(all point p with $x(p) < x^*$)* and P_R *(all points with $x(p) \geq x^*$)*
- Let $(p_L, q_L) \leftarrow$ `ClosestPair(`P_L`)`
- Let $(p_R, q_R) \leftarrow$ `ClosestPair(`P_R`)`
- Let (p, q) be the pair (among (p_L, q_L) and (p_R, q_R)) with the smaller distance and let d be this distance
- Let S be the sorted list of points with X-coordinate between $(x^* - d)$ and $(x^* + d)$
- For $i = 1$ to $|S|$
  - For $j = 1$ to 11
    - If $distance(S[i], S[i + j]) < d$
      - $(p, q) \leftarrow (S[i], S[i + j])$
      - $d \leftarrow distance(S[i], S[i + j])$
- Output(p, q)

- What is the running time of the above algorithm?
  - $T(n) = 2 \cdot T(n/2) + O(n \log n)$; $T(1) = O(1)$; $T(2) = O(1)$

Algorithm

`ClosestPair(`P`)`

- ... *//Base cases*
- Sort the points in increasing order of X-coordinates and pick the median point (x^*, y)
- Partition P into P_L *(all point p with $x(p) < x^*$)* and P_R *(all points with $x(p) \geq x^*$)*
- Let $(p_L, q_L) \leftarrow$ `ClosestPair(`P_L`)`
- Let $(p_R, q_R) \leftarrow$ `ClosestPair(`P_R`)`
- Let (p, q) be the pair (among (p_L, q_L) and (p_R, q_R)) with the smaller distance and let d be this distance
- Let S be the sorted list of points with X-coordinate between $(x^* - d)$ and $(x^* + d)$
- For $i = 1$ to $|S|$
  - For $j = 1$ to 11
    - If $distance(S[i], S[i + j]) < d$
      - $(p, q) \leftarrow (S[i], S[i + j])$
      - $d \leftarrow distance(S[i], S[i + j])$
- Output(p, q)

- What is the running time of the above algorithm?
  - $T(n) = 2 \cdot T(n/2) + O(n \log n)$; $T(1) = O(1)$; $T(2) = O(1)$
  - $T(n) = O(n \log^2 n)$

Algorithm

`ClosestPair(`P`)`

- ... *//Base cases*
- Sort the points in increasing order of X-coordinates and pick the median point (x^*, y)
- Partition P into P_L *(all point p with $x(p) < x^*$)* and P_R *(all points with $x(p) \geq x^*$)*
- Let $(p_L, q_L) \leftarrow$ `ClosestPair(`P_L`)`
- Let $(p_R, q_R) \leftarrow$ `ClosestPair(`P_R`)`
- Let (p, q) be the pair (among (p_L, q_L) and (p_R, q_R)) with the smaller distance and let d be this distance
- Let S be the sorted list of points with X-coordinate between $(x^* - d)$ and $(x^* + d)$
- For $i = 1$ to $|S|$
  - For $j = 1$ to 11
    - If $distance(S[i], S[i + j]) < d$
      - $(p, q) \leftarrow (S[i], S[i + j])$
      - $d \leftarrow distance(S[i], S[i + j])$
- Output(p, q)

- What is the running time of the above algorithm? $O(n \log^2 n)$
- Can we take the sorting out of the recursive program?
- What is the running time we get in doing so?

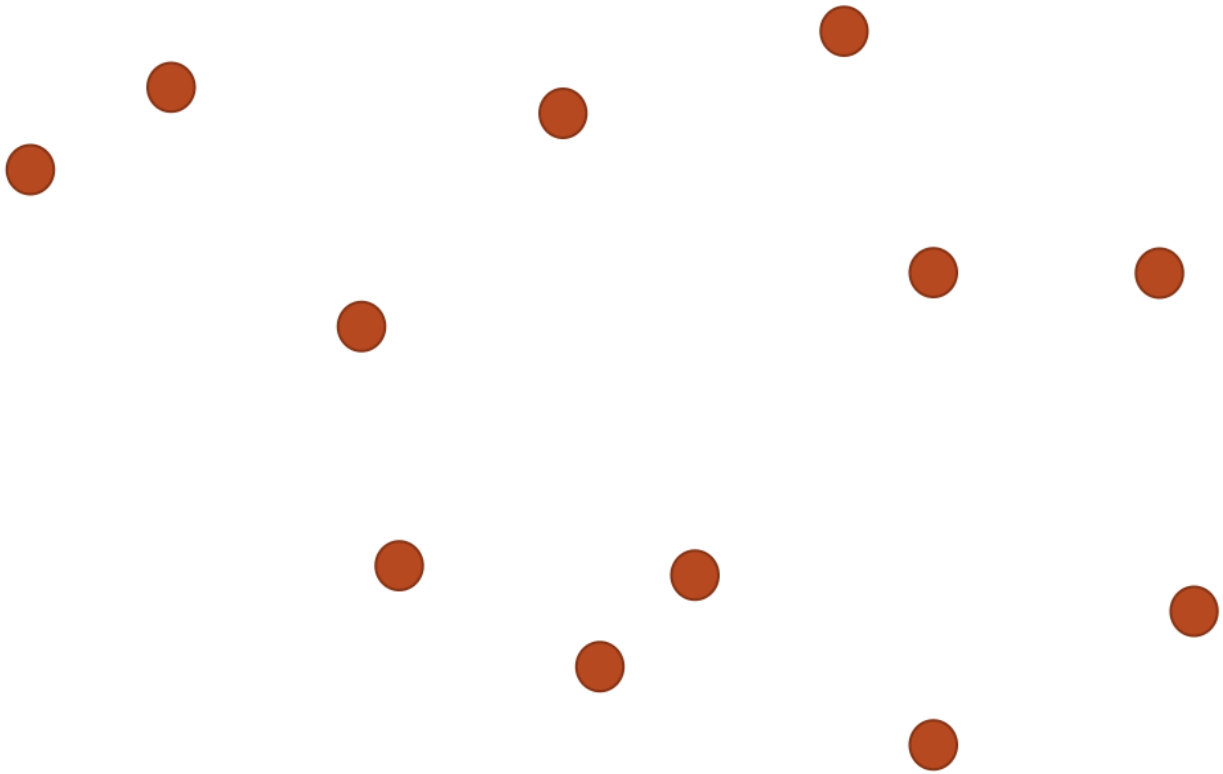
Algorithm

`ClosestPair(`P`)`

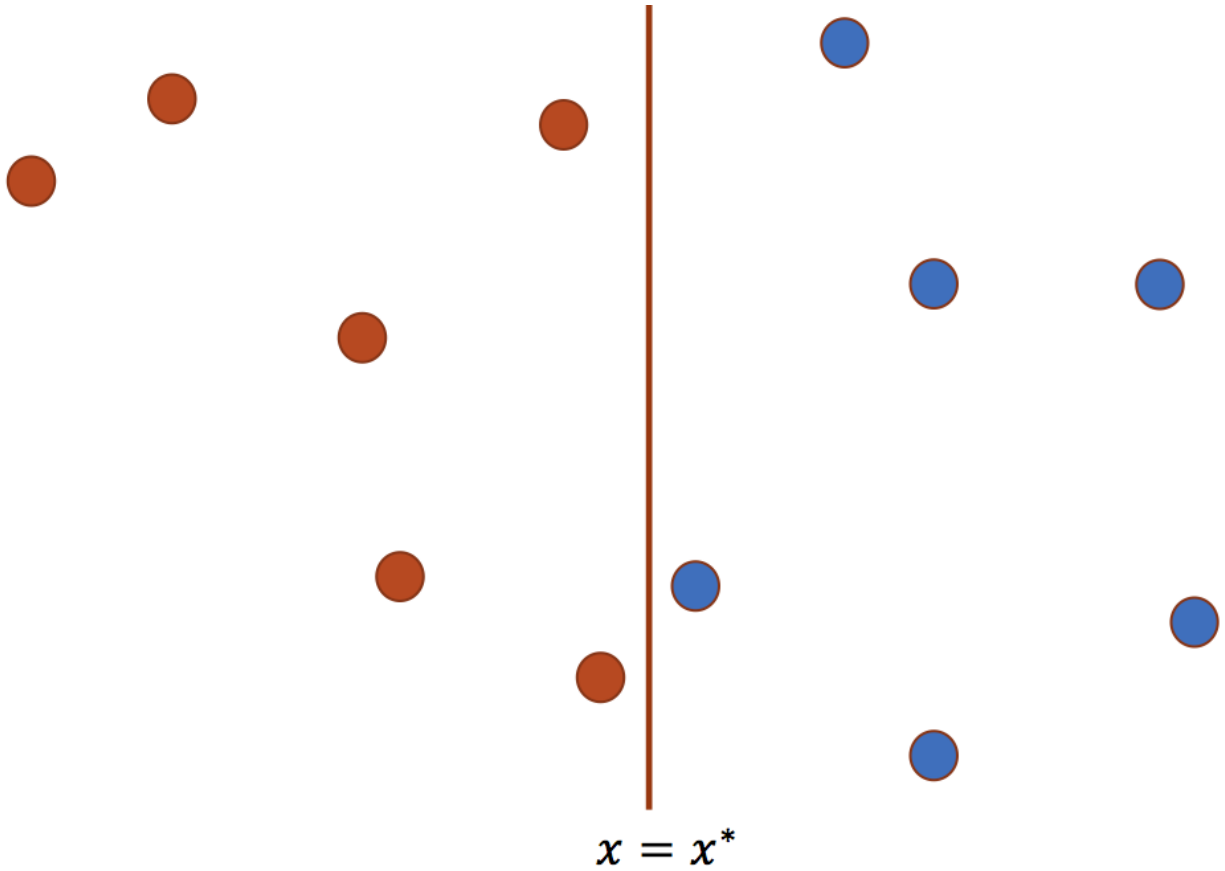
- ... *//Base cases*
- Sort the points in increasing order of X-coordinates and pick the median point (x^*, y)
- Partition P into P_L *(all point p with $x(p) < x^*$)* and P_R *(all points with $x(p) \geq x^*$)*
- Let $(p_L, q_L) \leftarrow$ `ClosestPair(`P_L`)`
- Let $(p_R, q_R) \leftarrow$ `ClosestPair(`P_R`)`
- Let (p, q) be the pair (among (p_L, q_L) and (p_R, q_R)) with the smaller distance and let d be this distance
- Let S be the sorted list of points with X-coordinate between $(x^* - d)$ and $(x^* + d)$
- For $i = 1$ to $|S|$
  - For $j = 1$ to 11
    - If $distance(S[i], S[i + j]) < d$
      - $(p, q) \leftarrow (S[i], S[i + j])$
      - $d \leftarrow distance(S[i], S[i + j])$
- Output(p, q)

- What is the running time of the above algorithm? $O(n \log^2 n)$
- Can we take the sorting out of the recursive program?
- What is the running time we get in doing so? $O(n \log n)$

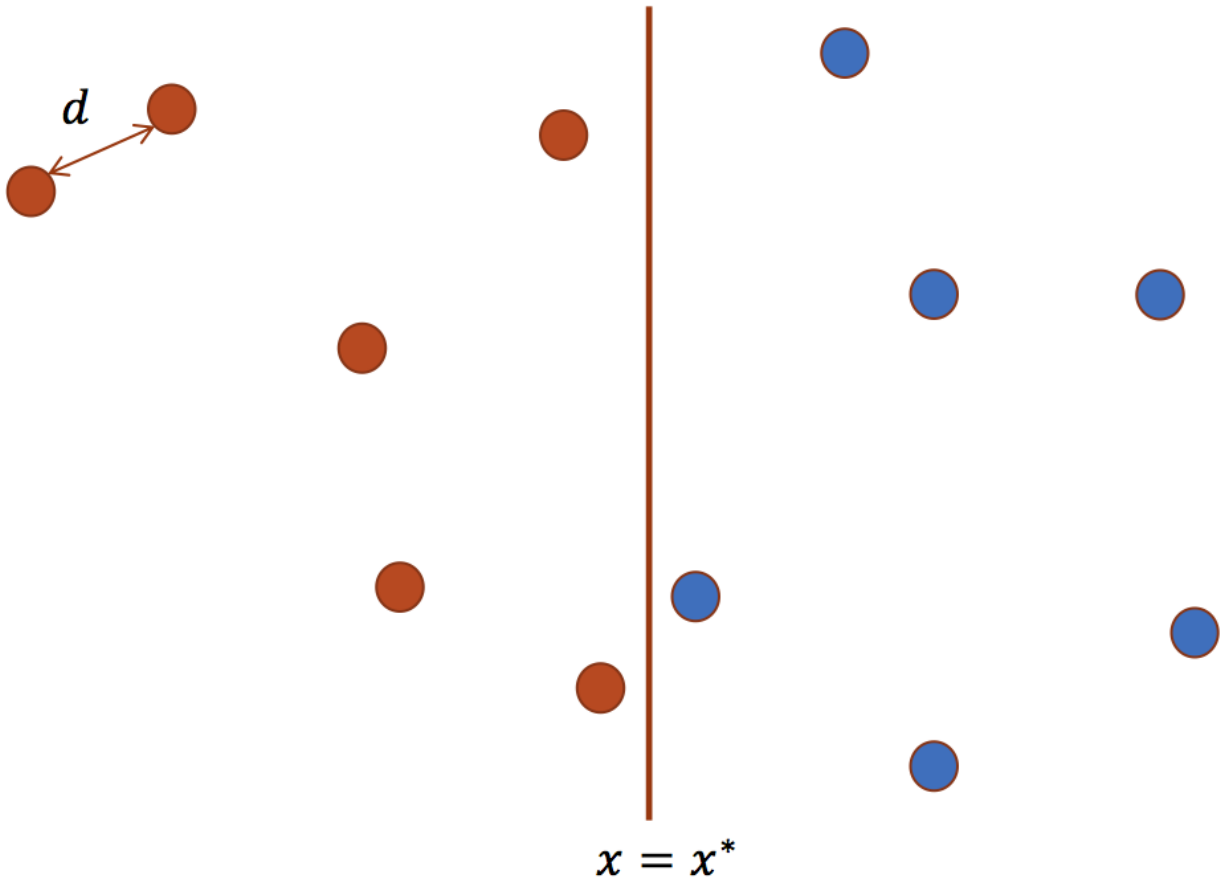
Divide and Conquer
Closest pair of points on a plane

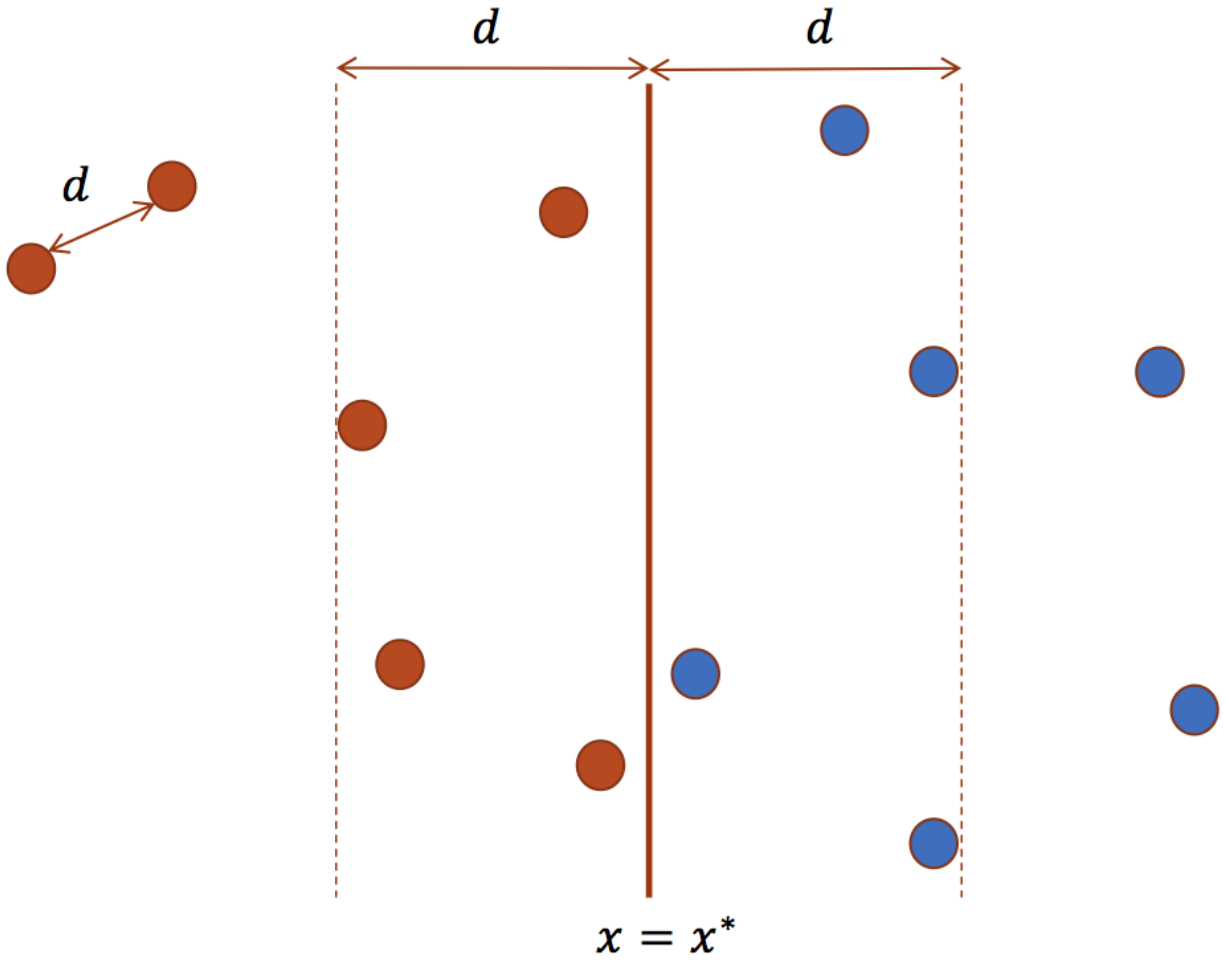


Divide and Conquer
Closest pair of points on a plane

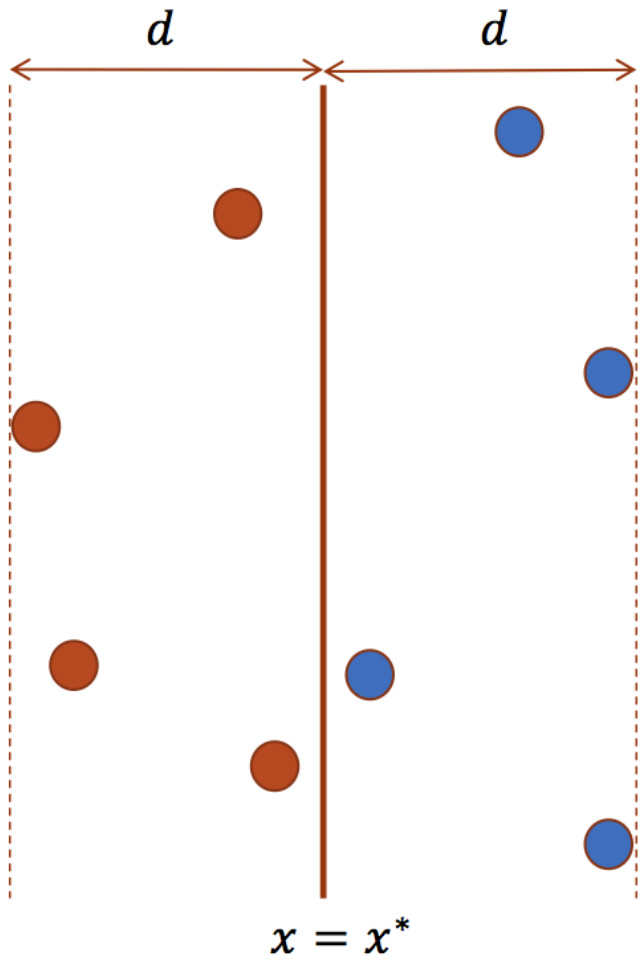


Divide and Conquer
Closest pair of points on a plane

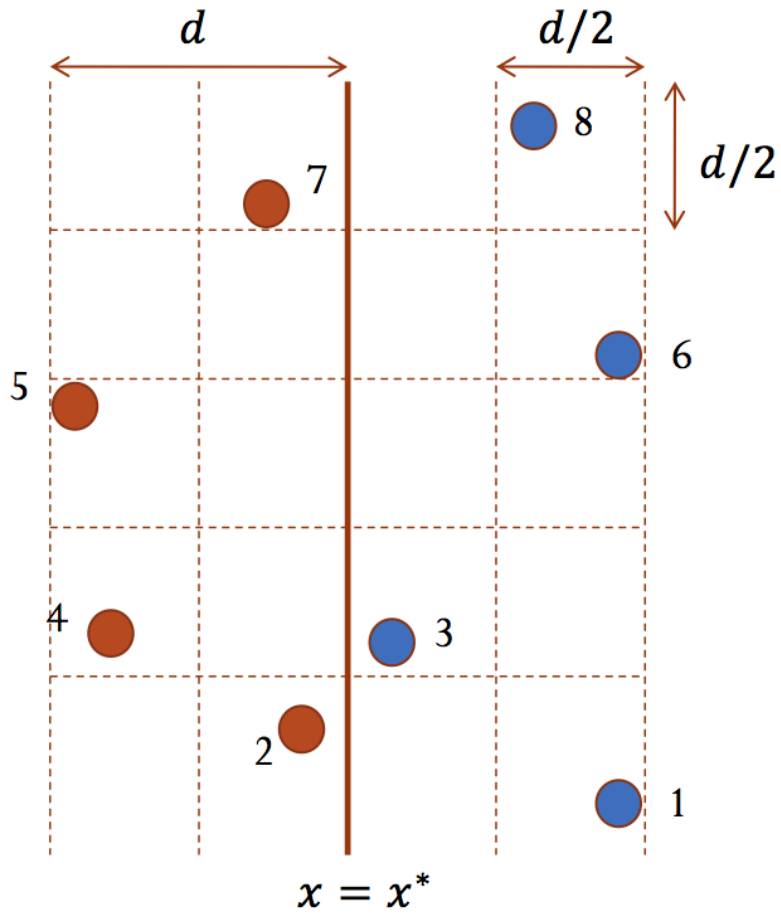




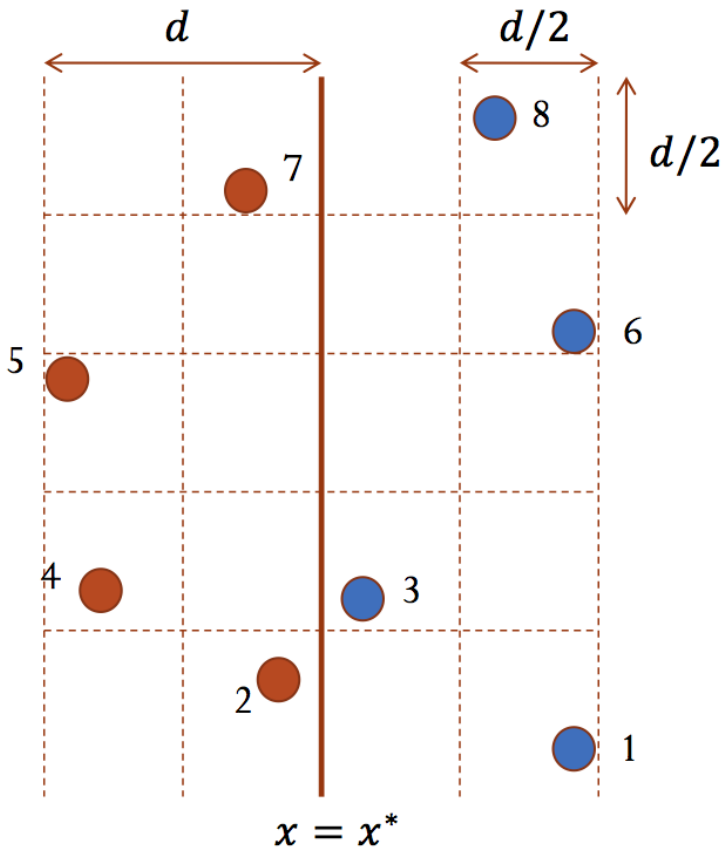
- Throw away points beyond $(x^* - d)$ and $(x^* + d)$.



- Throw away points beyond $(x^* - d)$ and $(x^* + d)$.



- Consider the list of points sorted based on Y-coordinate.



- Consider the list of points sorted based on Y-coordinate.
- Check the distance of a point in the list with the next 11 elements.

Problem

Median Finding: Given an array of unsorted numbers and an integer k, design an algorithm that finds the k^{th} smallest number in the array. Assume that A contains distinct numbers.

Problem

Median Finding: Given an array of unsorted numbers and an integer k, design an algorithm that finds the k^{th} smallest number in the array. Assume that A contains distinct numbers.

Algorithm

`Kth-smallest-incomplete`(A, k)

- Pick a number p as *pivot*
- Partition the numbers in A into A_L (all numbers $< p$) and A_R (all numbers $> p$)
- If $(|A_L| = k - 1)$, then return(p)
- If $(|A_L| > k - 1)$, then return(`Kth-smallest-incomplete`(A_L, k))
- If $(|A_L| < k - 1)$, then return(`Kth-smallest-incomplete`$(A_R, k - |A_L| - 1)$)

Problem

Median Finding: Given an array of unsorted numbers and an integer k, design an algorithm that finds the k^{th} smallest number in the array. Assume that A contains distinct numbers.

Algorithm

`Kth-smallest-incomplete`(A, k)

- Pick a number p as *pivot*
- Partition the numbers in A into A_L (all numbers $< p$) and A_R (all numbers $> p$)
- If $(|A_L| = k-1)$, then return(p)
- If $(|A_L| > k-1)$, then return(`Kth-smallest-incomplete`(A_L, k))
- If $(|A_L| < k-1)$, then return(`Kth-smallest-incomplete`$(A_R, k-|A_L|-1)$)

- What is the running time of this algorithm if the pivot is picked arbitrarily?

Problem

Median Finding: Given an array of unsorted numbers and an integer k, design an algorithm that finds the k^{th} smallest number in the array. Assume that A contains distinct numbers.

Algorithm

`Kth-smallest-incomplete`(A, k)

- Pick a number p as *pivot*
- Partition the numbers in A into A_L (all numbers $< p$) and A_R (all numbers $> p$)
- If $(|A_L| = k-1)$, then return(p)
- If $(|A_L| > k-1)$, then return(`Kth-smallest-incomplete`(A_L, k))
- If $(|A_L| < k-1)$, then return(`Kth-smallest-incomplete`$(A_R, k-|A_L|-1)$)

- What is the running time of this algorithm if the pivot is picked arbitrarily? $O(n^2)$

Algorithm

`Kth-smallest-incomplete`(A, k)

- Pick a number p as *pivot*
- Partition the numbers in A into A_L (all numbers $< p$) and A_R (all numbers $> p$)
- If $(|A_L| = k-1)$, then return(p)
- If $(|A_L| > k-1)$, then return(`Kth-smallest-incomplete`(A_L, k))
- If $(|A_L| < k-1)$, then return(`Kth-smallest-incomplete`$(A_R, k-|A_L|-1)$)

- How do we pick a good pivot number?
  - Randomly: We will look at this later.
  - Deterministically:

- How do we pick a good pivot number?
  - Randomly: We will look at this later.
  - Deterministically:

40	13	70	62	30	19	64	67	24	94	47	83	77	3	78	50	4	46	49	6	85	25	29	11	60

40	13	70	62	30
19	64	67	24	94
49	83	77	3	78
50	4	46	49	6
85	25	29	11	60

• Consider groups of 5 elements.

- How do we pick a good pivot number?
  - Randomly: We will look at this later.
  - Deterministically:

40	13	70	62	30	19	64	67	24	94	47	83	77	3	78	50	4	46	49	6	85	25	29	11	60

13	30	40	62	70
19	24	64	67	94
3	49	77	78	83
4	6	46	49	50
11	25	29	60	85

- Consider groups of 5 elements.
- Sort Individual groups.

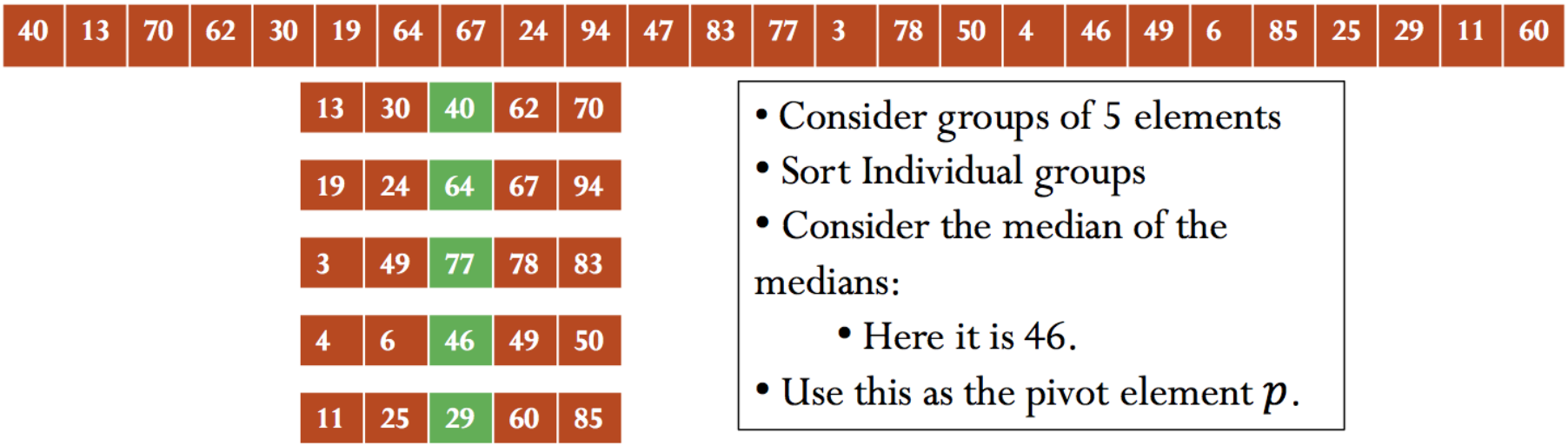
- How do we pick a good pivot number?
  - Randomly: We will look at this later.
  - Deterministically:

40	13	70	62	30	19	64	67	24	94	47	83	77	3	78	50	4	46	49	6	85	25	29	11	60

13	30	40	62	70
19	24	64	67	94
3	49	77	78	83
4	6	46	49	50
11	25	29	60	85

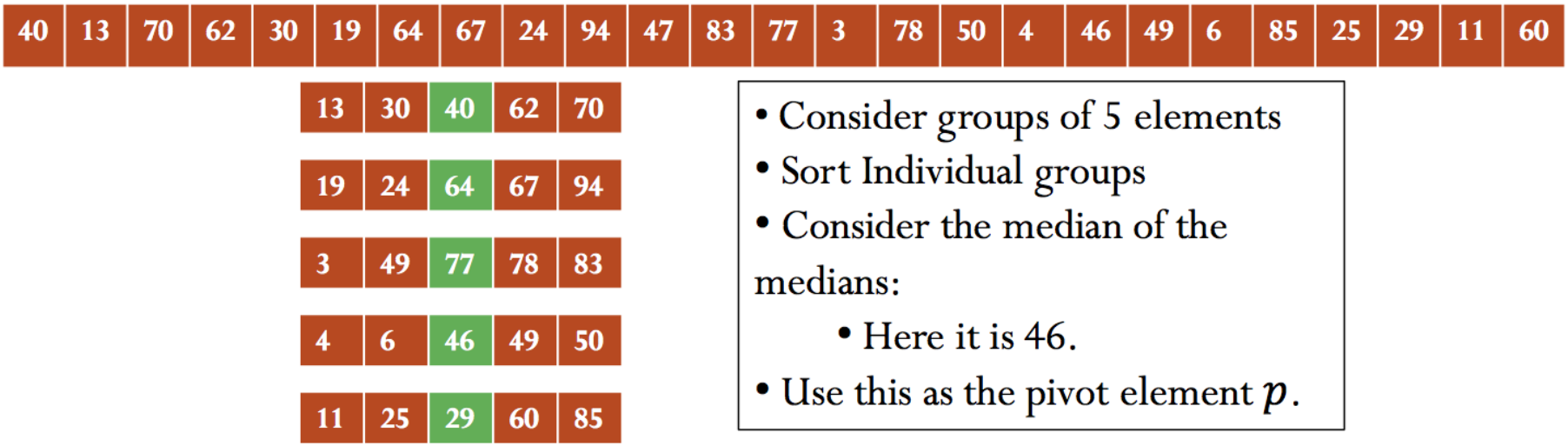
- Consider groups of 5 elements
- Sort Individual groups
- Consider the median of the medians:
  - Here it is 46.
- Use this as the pivot element.

- How do we pick a good pivot number?
  - Randomly: We will look at this later.
  - Deterministically:



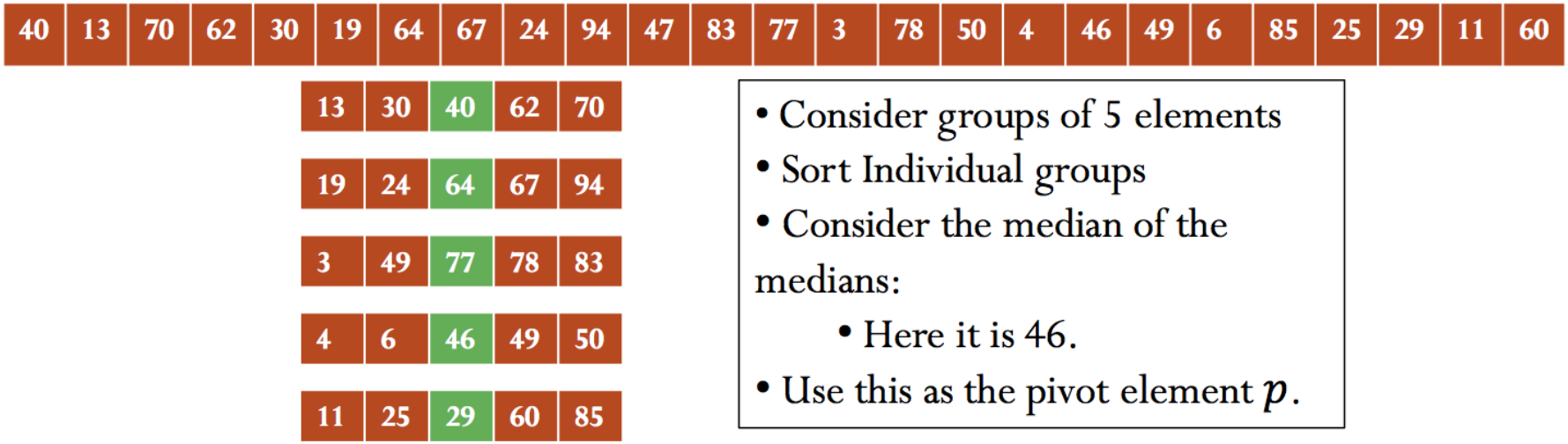
- How many elements in A are larger than p?

- How do we pick a good pivot number?
  - Randomly: We will look at this later.
  - Deterministically:



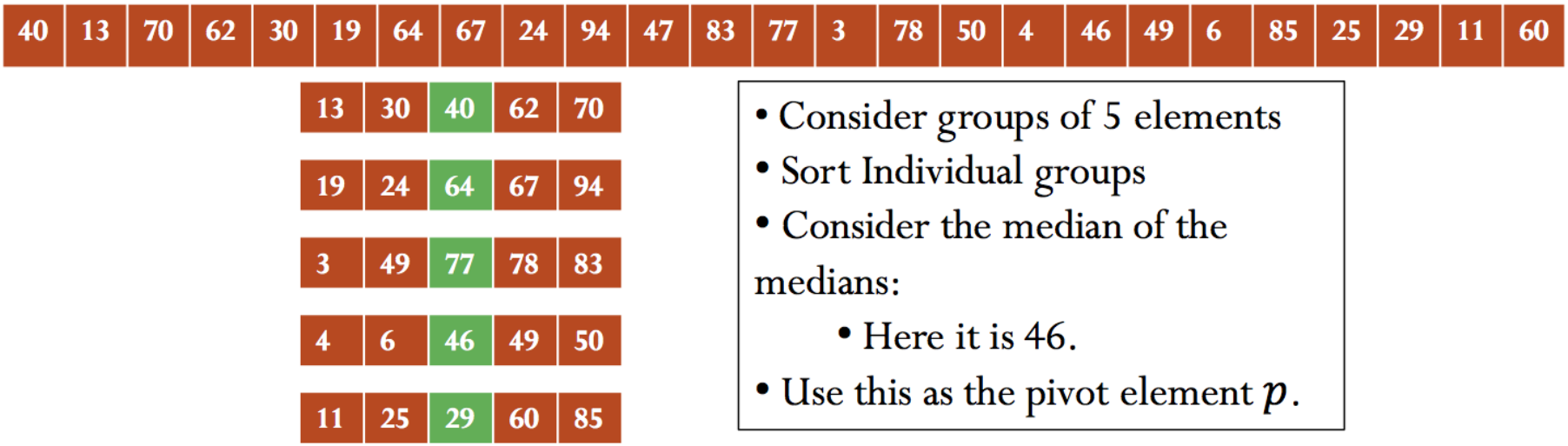
- How many elements in A are larger than p?
  - Claim 1: There are at least $(3n/10 - 6)$ numbers in A that are larger than p.

- How do we pick a good pivot number?
  - Randomly: We will look at this later.
  - Deterministically:



- How many elements in A are larger than p? at least $(3n/10 - 6)$
- How many elements in A are smaller than p?
  - Claim 2: There are at least $(3n/10 - 6)$ numbers in A that are smaller than p.

- How do we pick a good pivot number?
  - Randomly: We will look at this later.
  - Deterministically:



- How many elements in A are larger than p? at least $(3n/10 - 6)$
- How many elements in A are smaller than p? at least $(3n/10 - 6)$

Algorithm

`Find-Kth-smallest`(A, k)

- ... *//Base cases*
- Consider groups of 5 numbers, sort each group and create another array B containing the median number from each group
- $p \leftarrow$ `Find-Kth-smallest`$(B, \lfloor \frac{|B|}{2} \rfloor)$
- Partition the array A into A_L and A_R using p as the pivot
- If $(|A_L| = k-1)$, then return(p)
- If $(|A_L| > k-1)$, then return(`Find-Kth-smallest`(A_L, k))
- If $(|A_L| < k-1)$, then return(`Find-Kth-smallest`$(A_R, k-|A_L|-1)$)

- What is the running time of the above algorithm?

Algorithm

`Find-Kth-smallest`(A, k)

- ... *//Base cases*
- Consider groups of 5 numbers, sort each group and create another array B containing the median number from each group
- $p \leftarrow$ `Find-Kth-smallest`$(B, \lfloor \frac{|B|}{2} \rfloor)$
- Partition the array A into A_L and A_R using p as the pivot
- If $(|A_L| = k-1)$, then return(p)
- If $(|A_L| > k-1)$, then return(`Find-Kth-smallest`(A_L, k))
- If $(|A_L| < k-1)$, then return(`Find-Kth-smallest`$(A_R, k-|A_L|-1)$)

- What is the running time of the above algorithm?
  - $T(n) \leq T(\lceil n/5 \rceil) + T(7n/10+6) + O(n)$; $T(1) = O(1)$
  - What is $T(n)$?

End