Typically we analyze an algorithm where its inputs are parameterized only by their sizes.

By parameterizing the input in more ways, the analysis of the algorithm can be more informative.
We will discuss the 2D Maxima problem, which is closely related to the 2D Convex Hull problem. We analyze the Kirkpatrick-Seidel (KS) algorithm in three ways to give the upper bounds:

- $O(n \log n)$ where $n$ is the number of inputs (i.e. points in the plane),
- $O(n \log h)$ where $h$ is the number of outputs (i.e. maximal points of input), and
- $O(\min_{S_1, \ldots, S_k} \sum_{i=1}^{k} |S_i| \log n)$ where $S_1, \ldots, S_k$ is a legal partition of the input set.

We also mention matching lower bounds for each analysis.
We will discuss the 2D Maxima problem, which is closely related to the 2D Convex Hull problem. We analyze the Kirkpatrick-Seidel (KS) algorithm in three ways to give the upper bounds:

- \( O(n \log n) \) where \( n \) is the number of inputs (i.e. points in the plane),
We will discuss the 2D Maxima problem, which is closely related to the 2D Convex Hull problem. We analyze the Kirkpatrick-Seidel (KS) algorithm in three ways to give the upper bounds:

- $O(n \log n)$ where $n$ is the number of inputs (i.e. points in the plane),
- $O(n \log h)$ where $h$ is the number of outputs (i.e. maximal points of input), and

- $O(\min_{S_1, \ldots, S_k} \sum_{i=1}^{k} |S_i| \log n)$ where $S_1, \ldots, S_k$ is a legal partition of the input set.
We will discuss the 2D Maxima problem, which is closely related to the 2D Convex Hull problem. We analyze the Kirkpatrick-Seidel (KS) algorithm in three ways to give the upper bounds:

- \( O(n \log n) \) where \( n \) is the number of inputs (i.e. points in the plane),
- \( O(n \log h) \) where \( h \) is the number of outputs (i.e. maximal points of input), and
- \( O(\min_{S_1, \ldots, S_k} \{ \sum_{i=1}^k |S_i| \log \frac{n}{|S_i|} \}) \) where \( S_1, \ldots, S_k \) is a legal partition of the input set.

We also mention matching lower bounds for each analysis.
We will discuss the 2D Maxima problem, which is closely related to the 2D Convex Hull problem. We analyze the Kirkpatrick-Seidel (KS) algorithm in three ways to give the upper bounds:

- $O(n \log n)$ where $n$ is the number of inputs (i.e. points in the plane),
- $O(n \log h)$ where $h$ is the number of outputs (i.e. maximal points of input), and
- $O\left( \min_{S_1, \ldots, S_k} \left\{ \sum_{i=1}^{k} |S_i| \log \frac{n}{|S_i|} \right\} \right)$ where $S_1, \ldots, S_k$ is a legal partition of the input set.

We also mention matching lower bounds for each analysis.
What is instance optimally?

Suppose algorithms A and B solve the same problem - in what ways can we say A is better than B? In what way can we say A is better than any other algorithm that solves the problem?
What is instance optimally?

- Suppose algorithms A and B solve the same problem - in what ways can we say A is better than B? In what way can we say A is better than any other algorithm that solves the problem?

- Typical Approach: For sufficiently large input sizes $n$, $A$ is better than $B$ if $cost(A) \leq c \cdot cost(B)$ for constant $c$.

- Problem - too strong: consider BogoSort and BubbleSort for $Z$ being sorted.
What is instance optimally?

- Suppose algorithms A and B solve the same problem - in what ways can we say A is better than B? In what way can we say A is better than any other algorithm that solves the problem?
- Typical Approach: For sufficiently large input sizes \( n \), \textit{A is better than B} if \( \text{cost}(A) \leq c \cdot \text{cost}(B) \) for constant \( c \).
- Another approach: if \( \text{cost}(X, Z) \) denotes is a measure of how long algorithm \( X \) takes to solve problem instance \( Z \) then \textit{A is dominates B} if for all instances \( Z \),

\[
\text{cost}(A, Z) \leq \text{cost}(B, Z)
\]
What is instance optimally?

- Suppose algorithms A and B solve the same problem - in what ways can we say A is better than B? In what way can we say A is better than any other algorithm that solves the problem?

- Typical Approach: For sufficiently large input sizes $n$, A is better than B if $\text{cost}(A) \leq c \cdot \text{cost}(B)$ for constant $c$.

- Another approach: if $\text{cost}(X, Z)$ denotes is a measure of how long algorithm $X$ takes to solve problem instance $Z$ then A is dominates B if for all instances $Z$,

$$\text{cost}(A, Z) \leq \text{cost}(B, Z)$$

- Problem - too strong: consider BogoSort and BubbleSort for $Z$ being sorted.
Instance Optimality: Let $C$ be a set of algorithms we are interested in comparing algorithm $A$ against. Then we say that $A$ is instance optimal, wrt. approximation-constant $c \geq 1$ and set $C$, if for all $B \in C$ and problem instances $z$,

$$\text{cost}(A, Z) \leq c \cdot \text{cost}(B, Z),$$

where $c$ is independent of $C$ and $Z$. 

If $A$ is instance optimal, then there is no reason to use any other algorithm for the problem!
Instance Optimality: Let $C$ be a set of algorithms we are interested in comparing algorithm $A$ against. Then we say that $A$ is instance optimal, wrt. approximation-constant $c \geq 1$ and set $C$, if for all $B \in C$ and problem instances $z$,

$$\text{cost}(A, Z) \leq c \cdot \text{cost}(B, Z),$$

where $c$ is independent of $C$ and $Z$.

- If $A$ is instance optimal, then there is no reason to use any other algorithm for the problem!
Showing instance optimality

To show $A$ is instance optimal, we need to show two things:

1. An upper bound on $A$ for all instances $Z$ (i.e. $\text{cost}(A, Z) \leq x$), and
2. A matching lower bound, up to some constant, for all $B \in C$ and $Z$. (i.e. $x \leq c \cdot \text{cost}(B, Z)$).

Note: The matching bound needs to hold for all instances $Z$. This differs from worst-case analysis, where the bound only needs to match for sufficiently large inputs (i.e. $\text{cost}(A) \leq c \cdot \text{cost}(B)$ for $n \geq n_0$).
To show $A$ is instance optimal, we need to show two things:

1. An **upper bound** on $A$ for all instances $Z$ (i.e. $\text{cost}(A, Z) \leq x$), and
2. A **matching lower bound**, up to some constant, for all $B \in C$ and $Z$. (i.e. $x \leq c \cdot \text{cost}(B, Z)$).

**Note:** The matching bound needs to hold for all instances $Z$. This differs from worst-case analysis, where the bound only needs to match for sufficiently large inputs (i.e. $\text{cost}(A) \leq c \cdot \text{cost}(B)$ for $n \geq n_0$.)
Let $p$ and $q$ be points in the plane. $p$ is dominated by $q$ if $q$ is bigger than $p$ in both coordinates (along $x$ and $y$ axes.)
The 2DMaxima Problem

- Let \( p \) and \( q \) be points in the plane. \( p \) is dominated by \( q \) if \( q \) is bigger than \( p \) in both coordinates (along x and y axes.)

- A maximal point is a point not dominate by any others.

- 2DMaxima Problem: Given point set \( S \), find all maximal points of \( S \).
The Kirkpatrick-Seidel (KS) Algorithm for 2DMaxima

**Input:** A point set $Q$

**Output:** Maximal point set $S$

1. If $|Q| \leq 1$ add $Q$ to $S$, return.
2. Compute median x-coordinate among points in $Q$; partition $Q$ into left and right halves $Q_l$ and $Q_r$.
3. Let $q$ be the point with max. y-coord. in $Q_r$. Add $q$ to output set $S$.
4. Remove $q$ and all points that it dominates (in both $Q_l$, $Q_r$.)
5. Recurse on remaining $Q_l$, $Q_r$. 
The Kirkpatrick-Seidel (KS) Algorithm for 2DMaxima

Input: A point set \( Q \)

Output: Maximal point set \( S \)

1. If \( |Q| \leq 1 \) add \( Q \) to \( S \), return.
2. Compute median x-coordinate among points in \( Q \); partition \( Q \) into left and right halves \( Q_l \) and \( Q_r \).
3. Let \( q \) be the point with max. y-coord. in \( Q_r \). Add \( q \) to output set \( S \).
4. Remove \( q \) and all points that it dominates (in both \( Q_l, Q_r \).)
5. Recurse on remaining \( Q_l, Q_r \).
The Kirkpatrick-Seidel (KS) Algorithm for 2DMaxima

**Input:** A point set $Q$

**Output:** Maximal point set $S$

1. If $|Q| \leq 1$ add $Q$ to $S$, return.
2. Compute median x-coordinate among points in $Q$; partition $Q$ into left and right halves $Q_l$ and $Q_r$.
3. Let $q$ be the point with max. y-coord. in $Q_r$. Add $q$ to output set $S$.
4. Remove $q$ and all points that it dominates (in both $Q_l$, $Q_r$.)
5. Recurse on remaining $Q_l$, $Q_r$. 
The Kirkpatrick-Seidel (KS) Algorithm for 2D Maxima

**Input:** A point set \( Q \)

**Output:** Maximal point set \( S \)

1. If \( |Q| \leq 1 \) add \( Q \) to \( S \), return.
2. Compute median x-coordinate among points in \( Q \); partition \( Q \) into left and right halves \( Q_l \) and \( Q_r \).
3. Let \( q \) be the point with max. y-coord. in \( Q_r \). Add \( q \) to output set \( S \).
4. Remove \( q \) and all points that it dominates (in both \( Q_l \), \( Q_r \)).
5. Recurse on remaining \( Q_l \), \( Q_r \).
The Kirkpatrick-Seidel (KS) Algorithm for 2DMaxima

**Input:** A point set $Q$

**Output:** Maximal point set $S$

1. If $|Q| \leq 1$ add $Q$ to $S$, return.

2. Compute median x-coordinate among points in $Q$; partition $Q$ into left and right halves $Q_l$ and $Q_r$.

3. Let $q$ be the point with max. y-coord. in $Q_r$. Add $q$ to output set $S$.

4. Remove $q$ and all points that it dominates (in both $Q_l$, $Q_r$).

5. Recurse on remaining $Q_l$, $Q_r$. 
The Kirkpatrick-Seidel (KS) Algorithm for 2DMaxima

**Input:** A point set \( Q \)

**Output:** Maximal point set \( S \)

1. If \( |Q| \leq 1 \) add \( Q \) to \( S \), return.

2. Compute median \( x \)-coordinate among points in \( Q \); partition \( Q \) into left and right halves \( Q_l \) and \( Q_r \).

3. Let \( q \) be the point with max. \( y \)-coord. in \( Q_r \). Add \( q \) to output set \( S \).

4. Remove \( q \) and all points that it dominates (in both \( Q_l \), \( Q_r \).)

5. Recurse on remaining \( Q_l \), \( Q_r \).
Correctness of KS

- Point $q$ is maximal in the input $Q$: its x-coord. is larger than all points in $Q_l$ and its y-coord. is larger than all points in $Q_r$. Clearly, removal of any points dominated by $q$ is correct as well.

- Issue: During the execution, can a point that is not maximal in $Q$ become maximal by the removable of previous recursive calls?
Correctness of KS

- Point \( q \) is maximal in the input \( Q \): its x-coord. is larger than all points in \( Q_l \) and its y-coord. is larger than all points in \( Q_r \). Clearly, removal of any points dominated by \( q \) is correct as well.

- Issue: During the execution, can a point that is not maximal in \( Q \) become maximal by the removable of previous recursive calls?

- No, consider that maximal points from the recursive call on \( Q_r \) are maximal points of \( Q \) - let \( p \) be such a point added from the recursive call.
  - \( p \) is not dominated by \( q \) or any point in \( Q_l \) (\( p \) has larger x-coord.)
Correctness of KS

- Point \( q \) is maximal in the input \( Q \): its x-coord. is larger than all points in \( Q_l \) and its y-coord. is larger than all points in \( Q_r \). Clearly, removal of any points dominated by \( q \) is correct as well.

- Issue: During the execution, can a point that is not maximal in \( Q \) become maximal by the removable of previous recursive calls?

- No, consider that maximal points from the recursive call on \( Q_r \) are maximal points of \( Q \) - let \( p \) be such a point added from the recursive call.
  - \( p \) is not dominated by \( q \) or any point in \( Q_l \) (\( p \) has larger x-coord.)

- For maximal points from the recursive call on \( Q_l \), note that after pruning, all points that remain in \( Q_l \) must have larger y-coord. than \( q \) (i.e. these points cannot be dominated by \( q \).)
Runtime of KS

- Classic Divide-and-Conquer algorithm. For \( n \) points \( O(n) \) operations are needed to compute median (Blum et al. 1973). Two recursive calls are made. Thus the recurrence is:

\[
T(n) \leq 2T(n/2) + cn \implies T(n) \in O(n\log n)
\]
Runtime of KS

- Classic Divide-and-Conquer algorithm. For $n$ points $O(n)$ operations are needed to compute median (Blum et al. 1973). Two recursive calls are made. Thus the recurrence is:

$$T(n) \leq 2T(n/2) + cn \implies T(n) \in O(n \log n)$$

- The KS algorithm is also $\Omega(n \log n)$ in the worst-case under a comparison/decision tree model. Starting with $n$ points, we need to make $\Theta(n \log n)$ comparisons.
Runtime of KS

- Classic Divide-and-Conquer algorithm. For $n$ points $O(n)$ operations are needed to compute median (Blum et al. 1973). Two recursive calls are made. Thus the recurrence is:

$$T(n) \leq 2T(n/2) + cn \implies T(n) \in O(n\log n)$$

- The KS algorithm is also $\Omega(n\log n)$ in the worst-case under a comparison/decision tree model. Starting with $n$ points, we need to make $\Theta(n\log n)$ comparisons.

- Thus $T(n) \in \Theta(n\log n)$ - are we not done??
Some instances are easier than other instances:

- $O(n)$ to find median, $O(n)$ comparisons and deletions.
- $Q_l$ and $Q_r$ are now empty. Thus the algorithm (on this instance) has linear runtime.
Some instances are *easier* than other instances:

- $O(n)$ to find median, $O(n)$ comparisons and deletions. $Q_l$ and $Q_r$ are now empty. Thus the algorithm (on this instance) has linear runtime.
Some instances are *easier* than other instances:

- $O(n)$ to find median, $O(n)$ comparisons and deletions. $Q_l$ and $Q_r$ are now empty. Thus the algorithm (on this instance) has linear runtime.

- What makes this instance easy?
KS is $O(n\log h)$ Proof

Input size does not cut it alone! Let’s parameterize the input by both number of points $n$ and number of maximal points (i.e. output size) $h$. 

Define our recurrence as $T(n,h)$. Let $h_l$ and $h_r$ denote the number of maximal points in the left and right partitions (before removal). Thus we have,

$$T(n,h) \leq \max\{ h_l + h_r = h \{ T(n/2, h_l) + T(n/2, h_r) \} + cn$$

where $h_l, h_r < h$. We proceed by induction.
KS is $O(n \log h)$ Proof

- Input size does not cut it alone! Let’s parameterize the input by both number of points $n$ and number of maximal points (i.e. output size) $h$.

- Claim: The KS algorithm runs in $O(n \log h)$. Proof:
  - Define our recurrence as $T(n, h)$. Let $h_l$ and $h_r$ denote the number of maximal points in the left and right partitions (before removal). Thus we have,

  $$T(n, h) \leq \max_{h_l + h_r = h} \left\{ T\left(\frac{n}{2}, h_l\right) + T\left(\frac{n}{2}, h_r\right) \right\} + cn$$

  where $h_l, h_r < h$. We proceed by induction.
Proof Continued

\[ T(n, h) \leq \max_{h_l + h_r = h} \left\{ T\left(\frac{n}{2}, h_l\right) + T\left(\frac{n}{2}, h_r\right) \right\} + cn \]
Proof Continued

\[
T(n, h) \leq \max_{h_l + h_r = h} \left\{ T\left(\frac{n}{2}, h_l \right) + T\left(\frac{n}{2}, h_r \right) \right\} + cn
\]

\[
\leq \max_{h_l + h_r = h} \left\{ c \frac{n}{2} \log(h_l) + c \frac{n}{2} \log(h_r) \right\} + cn
\]
Proof Continued

\[ T(n, h) \leq \max_{h_l + h_r = h} \left\{ T\left(\frac{n}{2}, h_l\right) + T\left(\frac{n}{2}, h_r\right) \right\} + cn \]
\[ \leq \max_{h_l + h_r = h} \left\{ \frac{cn}{2} \log(h_l) + \frac{cn}{2} \log(h_r) \right\} + cn \]
\[ \leq cn + \frac{1}{2} cn \max_{h_l + h_r = h} \{ \log(h_l h_r) \} \]
Proof Continued

\[ T(n, h) \leq \max_{h_l + h_r = h} \{ T\left(\frac{n}{2}, h_l\right) + T\left(\frac{n}{2}, h_r\right) \} + cn \]

\[ \leq \max_{h_l + h_r = h} \left\{ \frac{c}{2}n \log(h_l) + \frac{c}{2}n \log(h_r) \right\} + cn \]

\[ \leq cn + \frac{1}{2}cn \max_{h_l + h_r = h} \left\{ \log(h_l h_r) \right\} \]

\[ \leq cn + \frac{1}{2}cn \left( \log\left(\frac{h}{2}\right)^2 \right) \]
Proof Continued

\[
T(n, h) \leq \max_{h_l + h_r = h} \left\{ T\left(\frac{n}{2}, h_l \right) + T\left(\frac{n}{2}, h_r \right) \right\} + cn
\]
\[
\leq \max_{h_l + h_r = h} \left\{ c\frac{n}{2} \log(h_l) + c\frac{n}{2} \log(h_r) \right\} + cn
\]
\[
\leq cn + \frac{1}{2}cn \max_{h_l + h_r = h} \{ \log(h_l h_r) \}
\]
\[
\leq cn + \frac{1}{2}cn \left( \log\left(\frac{h}{2}\right)^2 \right)
\]
\[
\leq cn \log(h)
\]
For $h \in O(1)$, the runtime of KS is linear. For $h \in O(n)$, the runtime is $n\log(n)$.

But even for many $h$ in between the algorithm preforms quite-well!
A more Fine-Grained Analysis

- For $h \in O(1)$, the runtime of KS is linear. For $h \in O(n)$, the runtime is $n\log(n)$.
- But even for many $h$ in between the algorithm preforms quite-well!
- Why? Many points are dominated by $q$ and removed, resulting in fewer points for recursive calls.
- To explore this more, we need to parameterize the input even further.
Legal Partitions

To show the instance optimality of the KS algorithm, we use the following parameterization. For partition $S_1, \cdots, S_k$ of input set $S$, the partition $\{S_i\}$ is a **legal partition/set** if:

1. $S_i$ contain a single point, or
2. $S_i$ is contained in the **interior** of an axis-aligned box $B_i$ and is located below the **staircase** of $S$. 

Intuition: For case 2) if the top-right corner of $B_i$ is a point of the set; choosing this point in KS will remove the entirety of $S_i$. 
Legal Partitions

To show the instance optimality of the KS algorithm, we use the following parameterization. For partition $S_1, \ldots, S_k$ of input set $S$, the partition $\{S_i\}$ is a legal partition/set if:

1. $S_i$ contain a single point, or
2. $S_i$ is contained in the interior of an axis-aligned box $B_i$ and is located below the staircase of $S$. 

Intuition: For case 2) if the top-right corner of $B_i$ is a point of the set; choosing this point in KS will remove the entirety of $S_i$. 

$S_1 \quad S_2 \quad S_3 \quad S_4 \quad S_5 \quad S_6 \quad B_4 \quad B_5 \quad B_6$
Legal Partitions

- To show the instance optimality of the KS algorithm, we use the following parameterization. For partition $S_1, \ldots, S_k$ of input set $S$, the partition $\{S_i\}$ is a **legal partition/set** if:
  1. $S_i$ contain a single point, or
  2. $S_i$ is contained in the **interior** of an axis-aligned box $B_i$ and is located below the **staircase** of $S$.

- **Intuition**: For case 2) if the top-right corner of $B_i$ is a point of the set; choosing this point in KS will remove the entirety of $S_i$. 

![Diagram of legal partitions and KS algorithm](image-url)
For a point set $S$ partitioned into $k$ legal sets, the runtime of the KS algorithm is:

$$O\left(\sum_{i=1}^{k} |S_i| \log \frac{n}{|S_i|}\right)$$

What this says: there is a relationship between legal partitions and the rate at which points are removed.
Proof: Analyze the recurrence tree. The amount of work done at each level is linear in the number of points remaining at that level. We will bound how much \( S_i \) contributes to the number of points remaining at level \( j \).
Proof: Analyze the recurrence tree. The amount of work done at each level is linear in the number of points remaining at that level. We will bound how much $S_i$ contributes to the number of points remaining at level $j$.

Claim: The number of points in $S_i$ not yet removed at level $j$ is at most $\min\{|S_i|, 2n/2^j\}$. Proceeding with the claim, across all levels, $S_i$ contributes:
Proof: Analyze the recurrence tree. The amount of work done at each level is linear in the number of points remaining at that level. We will bound how much $S_i$ contributes to the number of points remaining at level $j$.

Claim: The number of points in $S_i$ not yet removed at level $j$ is at most $\min\{|S_i|, 2n/2^j\}$. Proceeding with the claim, across all levels, $S_i$ contributes:

$$\leq \sum_{j=0}^{\lfloor \log_2(n) \rfloor} \min\{|S_i|, 2n/2^j\}$$
Proof: Analyze the recurrence tree. The amount of work done at each level is linear in the number of points remaining at that level. We will bound how much $S_i$ contributes to the number of points remaining at level $j$.

Claim: The number of points in $S_i$ not yet removed at level $j$ is at most $\min\{|S_i|, 2n/2^j\}$. Proceeding with the claim, across all levels, $S_i$ contributes:

$$\leq \sum_{j=0}^{\left\lceil \log_2(n) \right\rceil} \min\{|S_i|, 2n/2^j\}$$

$$\leq \left( |S_i| + \cdots + |S_i| + \frac{|S_i|}{1} + \frac{|S_i|}{2^1} + \frac{|S_i|}{2^2} + \cdots \right)$$
\[ \leq |S_i| \left( \log(n/|S_i|) + 3 \right) \]

- which is in \(O(|S_i| \log(n/|S_i|))\). As each \(S_i\) is a partition of the input set, each of the \(k\) partitions contributes \(O\left(\sum_{i=1}^{k} |S_i| \log \frac{n}{|S_i|}\right)\) to the algorithm.
Claim: The number of points in \( S_i \) not yet removed at level \( j \) is at most \( \min\{|S_i|, 2n/2^j\} \). Consider recursion level \( j \):
Claim: The number of points in $S_i$ not yet removed at level $j$ is at most $\min\{|S_i|, 2n/2^j\}$. Consider recursion level $j$:

Recall that $S_i$ is contained in a box $B_i$. Any points of $S_i$ not yet removed must be contained in between two previously identified maximal points (along x-axis).
Proof of Claim

**Claim:** The number of points in $S_i$ not yet removed at level $j$ is at most $\min\{|S_i|, 2n/2^j\}$. Consider recursion level $j$:

- Recall that $S_i$ is contained in a box $B_i$. Any points of $S_i$ not yet removed must be contained in between two previously identified maximal points (along x-axis)
- All points in $S_i$ have x-coord. less than b’s x-coord as $b$ is maximal.
Proof of Claim

**Claim:** The number of points in $S_i$ not yet removed at level $j$ is at most $\min\{|S_i|, 2n/2^j\}$. Consider recursion level $j$:

- Recall that $S_i$ is contained in a box $B_i$. **Any points of $S_i$ not yet removed must be contained in between two previously identified maximal points (along x-axis)**.

- All points in $S_i$ have x-coord. less than b’s x-coord as $b$ is maximal.

- $B_i$ (and thus $S_i$) is below the staircase of $S$ - as $a$ is maximal, all points of $S_i$ have y-coord less than $a$. 
Claim: The number of points in $S_i$ not yet removed at level $j$ is at most $\min\{|S_i|, 2n/2^j\}$. Consider recursion level $j$:

For every pair of adjacent maxima found so far (along x-axis) at level $j$, there are at most $2n/2^j$ remaining points in between them.
Claim: The number of points in $S_i$ not yet removed at level $j$ is at most $\min\{|S_i|, 2n/2^j\}$. Consider recursion level $j$:

For every pair of adjacent maxima found so far (along x-axis) at level $j$, there are at most $2n/2^j$ remaining points in between them.

At level $j$ we’ve partitioned the point set into at most $2^j$ (non-empty) buckets. In each bucket, there are at most $n/2^j$ points.
Proof of Claim Continued

- **Claim:** The number of points in $S_i$ not yet removed at level $j$ is at most $\min\{|S_i|, 2n/2^j\}$. Consider recursion level $j$:
  - For every pair of adjacent maxima found so far (along x-axis) at level $j$, there are at most $2n/2^j$ remaining points in between them.
  - At level $j$ we’ve partitioned the point set into at most $2^j$ (non-empty) buckets. In each bucket, there are at most $n/2^j$ points.
  - Each recursive call identifies a maximal point. Once identified and removed, at most $2n/2^j$ points can remain between consecutive buckets.
But is there more to this?

- The proof of the claim only required the sets $S_i$ to be legal sets.
But is there more to this?

- The proof of the claim only required the sets $S_i$ to be legal sets.
- Thus the overall upper bound will hold for all legal partitions! That is,

$$O\left( \min_{\text{legal}\{S_i\}} \sum_{i=1}^{k} |S_i| \log \frac{n}{|S_i|} \right)$$

Note: When each $S_i$ is a singleton, we have the $O(n \log n)$ bound, and, when each maximal point is a singleton, and non-maximal points are in sets below and left of each maximal point, we have the $O(n \log h)$ bound.
But is there more to this?

- The proof of the claim only required the sets $S_i$ to be legal sets.
- Thus the overall upper bound will hold for all legal partitions! That is,

$$O\left( \min_{\text{legal}\{S_i\}} \sum_{i=1}^{k} |S_i| \log \frac{n}{|S_i|} \right)$$  \hspace{1cm} (1)

- **Note:** When each $S_i$ is a singleton, we have the $O(n \log n)$ bound, and,

- when each maximal point is a singleton, and non-maximal points are in sets below and left of each maximal point, we have the $O(n \log h)$ bound.
We’ve given a good upper bound on the runtime of the KS algorithm in (1), but for all problem instances, will KS preform better than any other algorithm?
We’ve given a good upper bound on the runtime of the KS algorithm in (1), but for all problem instances, will KS preform better than any other algorithm?

No, because of silly algorithms.

But, “annoying counterexamples are not a good reason to abandon the quest for an interesting theorem” - Tim Roughgarden.
We’ve given a good upper bound on the runtime of the KS algorithm in (1), but for all problem instances, will KS perform better than any other algorithm?

No, because of silly algorithms.

Consider, the **KS with Extra Steps** algorithm:

**Input:** A point set $Q$

**Output:** Maximal point set $S$

1. Check if $Q$ is instance $Z$
2. If so, output hard-coded maximal point set of $Z$
3. If not, output $KS(Q)$.

"annoying counterexamples are not a good reason to abandon the quest for an interesting theorem" - Tim Roughgarden.
We’ve given a good upper bound on the runtime of the KS algorithm in (1), but for all problem instances, will KS perform better than any other algorithm?

No, because of silly algorithms.

Consider, the **KS with Extra Steps** algorithm:

**Input:** A point set \( Q \)

**Output:** Maximal point set \( S \)

1. Check if \( Q \) is instance \( Z \)
2. If so, output hard-coded maximal point set of \( Z \)
3. If not, output \( KS(Q) \).

But, "**annoying counterexamples are not a good reason to abandon the quest for an interesting theorem**" - Tim Roughgarden.
What to do?

▶ There are two approaches:

1. Restrict algorithms $B$ to be *order-oblivious*; the input set $Q$ must first be sorted to compare it against hard coded $Z$. 
What to do?

- There are two approaches:
  1. Restrict algorithms $B$ to be *order-oblivious*; the input set $Q$ must first be sorted to compare it against hard coded $Z$.
  2. Redefine $cost(B, Z)$; compare the $KS$ algorithm against the performance of $B$ on permutations of $Z$ - take the max or average of this cost.
Let $Cost(B, Z) = \max_{\pi} \{cost(B, \pi(Z))\}$ where $\pi(Z)$ denotes the ordering the point set $Z$ is presented to $B$, according to an ordering $\pi$. Then for every point set $S$ and every algorithm $B$, $Cost(B, S) \in \Omega(\min_{\text{legal}} \sum_{i=1}^{k} |S_i| \log n |S_i|)$. 

Proof outline: For any correct algorithm $A$ with input $S$, there exists a permutation of $S$ on which at least $\Omega(\min_{\text{legal}} \sum_{i=1}^{k} |S_i| \log n |S_i|)$ comparisons are made.
Let $Cost(B, Z) = \max_{\pi} \{ cost(B, \pi(Z)) \}$ where $\pi(Z)$ denotes the ordering the point set $Z$ is presented to $B$, according to an ordering $\pi$.

Then for every point set $S$ and every algorithm $B$,

$$Cost(B, S) \in \Omega \left( \min_{\text{legal}\{S_i\}} \sum_{i=1}^{k} |S_i| \log \frac{n}{|S_i|} \right)$$

(2)
Let \( \text{Cost}(B, Z) = \max_{\pi} \{ \text{cost}(B, \pi(Z)) \} \) where \( \pi(Z) \) denotes the ordering the point set \( Z \) is presented to \( B \), according to an ordering \( \pi \).

Then for every point set \( S \) and every algorithm \( B \),

\[
\text{Cost}(B, S) \in \Omega\left( \min_{\text{legal}\{S_i\}} \sum_{i=1}^{k} |S_i| \log \frac{n}{|S_i|} \right)
\]  \hspace{2cm} (2)

**Proof outline:** For any correct algorithm \( A \) with input \( S \), there exists a permutation of \( S \) on which at least \( \Omega\left( \min_{\text{legal}\{S_i\}} \sum_{i=1}^{k} |S_i| \log \frac{n}{|S_i|} \right) \) comparisons are made.
A k-d tree ($k = 2$) of axis-aligned boxes is generated - the root is the entire plane, internal nodes are regions (boxes) that are across the staircase of $S$, and leaf nodes are boxes strictly below the staircase or singletons.

Maintain a node (box) $B_p$ for each point $p$ - only when $p$ is a leaf node is the algorithm certain of $p$'s exact position within $B_p$.

An adversary can simulate running $A$ on $S$ and see how permuting the order in which points of $S$ are considered will hide maximal points, requiring more comparisons.

Let $D$ be the sum of the depths of boxes $B_p$ for each $p \in S$ and $T$ be the number of comparisons made by $A$. It is shown that $T \in \Omega(D)$, and that $D$ is of order $\min legal \{ S_i \} \sum_{k=1}^{m} |S_i| \log n |S_i|$. 
A k-d tree \((k = 2)\) of axis-aligned boxes is generated - the root is the entire plane, internal nodes are regions (boxes) that are across the staircase of \(S\), and leaf nodes are boxes strictly below the staircase or singletons.

Maintain a node (box) \(B_p\) for each point \(p\) - only when \(p\) is a leaf node is the algorithm certain of \(p\)'s exact position within \(B_p\).
Proof Outline Continued

▶ A k-d tree \((k = 2)\) of axis-aligned boxes is generated - the root is the entire plane, internal nodes are regions (boxes) that are across the staircase of \(S\), and leaf nodes are boxes strictly below the staircase or singletons.

▶ Maintain a node (box) \(B_p\) for each point \(p\) - only when \(p\) is a leaf node is the algorithm certain of \(p\)’s exact position within \(B_p\).

▶ An adversary can simulate running \(A\) on \(S\) and see how permuting the order in which points of \(S\) are considered will hide maximal points, requiring more comparisons.
A k-d tree \((k = 2)\) of axis-aligned boxes is generated - the root is the entire plane, internal nodes are regions (boxes) that are across the staircase of \(S\), and leaf nodes are boxes strictly below the staircase or singletons.

Maintain a node (box) \(B_p\) for each point \(p\) - only when \(p\) is a leaf node is the algorithm certain of \(p\)'s exact position within \(B_p\).

An adversary can simulate running \(A\) on \(S\) and see how permuting the order in which points of \(S\) are considered will hide maximal points, requiring more comparisons.

Let \(D\) be the sum of the depths of boxes \(B_p\) for each \(p \in S\) and \(T\) be the number of comparisons made by \(A\). It is shown that \(T \in \Omega(D)\), and that \(D\) is of order \(\min_{\text{legal}\{S_i\}} \sum_{i=1}^{k} |S_i| \log \frac{n}{|S_i|}\).
Conclusions and Take-Away Points

▶ By parameterizing our input in terms of more than just the input size, we can give a more descriptive upper bound on runtimes.

▶ For KS algorithm on the 2DMaxima problem, we saw how describing the input in terms of the both the output set size $h$ and legal partitions of the input gave more descriptive runtime performances than when only considering input size.

▶ This was the first result of the paper Afshani, Barbay, and Chan, which can be extended to the 3DMaxima problem and 2D and 3D Convex Hull problem.

▶ The KS algorithm for 2DMaxima is instance optimal when compared against algorithms that do not "memorize" solution for some inputs.
Conclusions and Take-Away Points

▶ By parameterizing our input in terms of more than just the input size, we can give a more descriptive upper bound on runtimes.

▶ For KS algorithm on the 2DMaxima problem, we saw how describing the input in terms of the both the output set size $h$ and legal partitions of the input gave more descriptive runtime performances than when only considering input size.

▶ This was the first result of the paper Afshani, Barbay, and Chan, which can be extended to the 3DMaxima problem and 2D and 3D Convex Hull problem.
Conclusions and Take-Away Points

- By parameterizing our input in terms of more than just the input size, we can give a more descriptive upper bound on runtimes.

- For KS algorithm on the 2DMaxima problem, we saw how describing the input in terms of the both the output set size $h$ and legal partitions of the input gave more descriptive runtime performances than when only considering input size.
  - This was the first result of the paper Afshani, Barbay, and Chan, which can be extended to the 3DMaxima problem and 2D and 3D Convex Hull problem.

- The KS algorithm for 2DMaxima is instance optimal when compared against algorithms that do not "memorize" solution for some inputs.
Conclusions and Take-Away Points

- Instance optimality is *very strong*, and seems problem specific.
  - For 2DMaxima, this *entropy-like* measure of the point set was the bound reached by the KS algorithm - what is the corresponding measure for other problems?

- Instance optimality may not exist for all problems - the best algorithm may rely on the input domain.

- Even if instance optimality may exist, a matching lower bound needs to be shown on an input-by-input basis. If lower bound proof techniques in the computational model used are not well known, it is difficult to prove such results.
Conclusions and Take-Away Points

- Instance optimality is very strong, and seems problem specific.
  - For 2DMaxima, this entropy-like measure of the point set was the bound reached by the KS algorithm - what is the corresponding measure for other problems?

- Instance optimality may not exist for all problems - the best algorithm may rely on the input domain.
Conclusions and Take-Away Points

- Instance optimality is very strong, and seems problem specific.
  - For 2D Maxima, this entropy-like measure of the point set was the bound reached by the KS algorithm - what is the corresponding measure for other problems?

- Instance optimality may not exist for all problems - the best algorithm may rely on the input domain.

- Even if instance optimality may exist, a matching lower bound needs to be shown on an input-by-input basis. If lower bound proof techniques in the computational model used are not well known, it is difficult to prove such results.
Conclusions and Take-Away Points

▶ Thank you for listening!
▶ Next week: *Online Paging and Resource Augmentation*