Instance Optimal Geometric Algorithms

Notes on Tim Roughgarden's *Beyond Worst-Case Analysis*, Lecture 2

Kyle Clarkson

UBC Algorithms Reading Group - May 19th, 2020

Presentation Outline

- 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:

Presentation Outline

- 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),

Presentation Outline

- 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

Presentation Outline

- 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.

Presentation Outline

- 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.

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.

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.
- Another approach: if $cost(X, Z)$ 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,

$$cost(A, Z) \leq cost(B, Z)$$

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 $cost(A) \leq c \cdot cost(B)$ for constant c.
- Another approach: if $cost(X, Z)$ 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,

$$cost(A, Z) \leq cost(B, Z)$$

- Problem - too strong: consider *BogoSort* and *BubbleSort* for Z being sorted.

What is instance optimally?

- *Instance Optimality:* Let $\mathcal{C}$ 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 $\mathcal{C}$, if for all $B \in \mathcal{C}$ and problem instances z,

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

where c is independent of $\mathcal{C}$ and Z.

What is instance optimally?

- *Instance Optimality:* Let $\mathcal{C}$ 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 $\mathcal{C}$, if for all $B \in \mathcal{C}$ and problem instances z,

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

where c is independent of $\mathcal{C}$ 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. $cost(A, Z) \leq x$), and
  2. A **matching lower bound**, up to some constant, for all $B \in \mathcal{C}$ and Z. (i.e. $x \leq c \cdot cost(B, Z)$).

Showing instance optimality

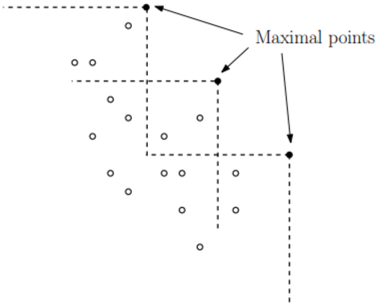
- To show A is instance optimal, we need to show two things:
  1. An **upper bound** on A for all instances Z (i.e. $cost(A, Z) \leq x$), and
  2. A **matching lower bound**, up to some constant, for all $B \in \mathcal{C}$ and Z. (i.e. $x \leq c \cdot cost(B, Z)$).
- **Note:** The matching bound needs 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. $cost(A) \leq c \cdot cost(B)$ for $n \geq n_0$.)

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.)

The 2DMaxima Problem

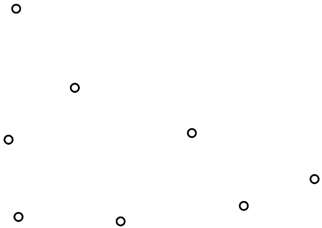
- Let p and q be points in the plane. *p is dominated by q* if q is bigger than p in both coordinates (along x and y axes.)
- A *maximal point* is a point not dominate by any others.
- *2DMaxima Problem:* Given point set S, find all maximal points of S.



The Kirkpatrick-Seidel (KS) Algorithm for 2DMaxima

Input: A point set Q
Output: Maximal point set S

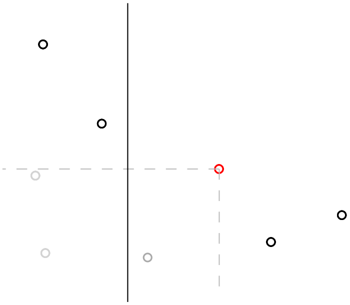
1. If $|Q| \leq 1$ add Q to S, return.
2. Compute median x-coordinate among points in Q; partition Q into left and right halves Q_l and Q_r.
3. Let q be the point with max. y-coord. in Q_r. Add q to output set S.
4. Remove q and all points that it dominates (in both Q_l, Q_r.)
5. Recurse on remaining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

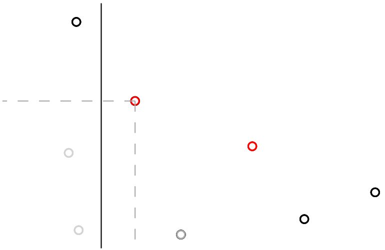


The Kirkpatrick-Seidel (KS) Algorithm for 2DMaxima

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.

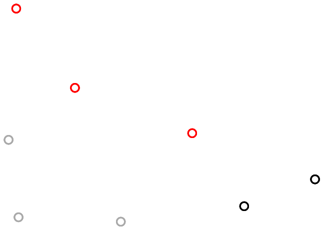


The Kirkpatrick-Seidel (KS) Algorithm for 2DMaxima

Input: A point set Q

Output: Maximal point set S

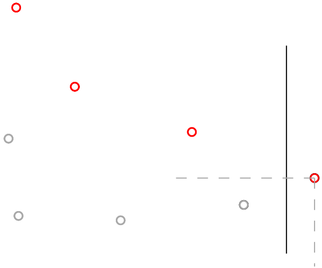
1. If $|Q| \leq 1$ add Q to S, return.
2. Compute median x-coordinate among points in Q; partition Q into left and right halves Q_l and Q_r.
3. Let q be the point with max. y-coord. in Q_r. Add q to output set S.
4. Remove q and all points that it dominates (in both Q_l, Q_r.)
5. Recurse on remaining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

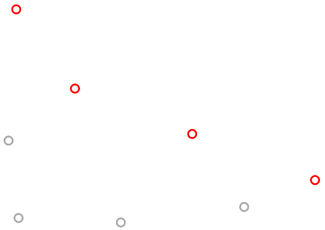


The Kirkpatrick-Seidel (KS) Algorithm for 2DMaxima

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.



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:

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

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:

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

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

Runtime of KS

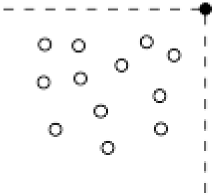
- Classic Divide-and-Conquer algorithm. For n points $O(n)$ operations are needed to compute median (Blum et al. 1973). Two recursive calls are made. Thus the recurrence is:

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

- The KS algorithm is also $\Omega(nlogn)$ in the worst-case under a comparison/decision tree model. Starting with n points, we need to make $\Theta(nlogn)$ comparisons.
- Thus $T(n) \in \Theta(nlogn)$ - are we not done??

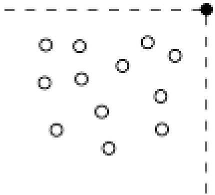
Output-Sensitive Analysis

- Some instances are *easier* than other instances:



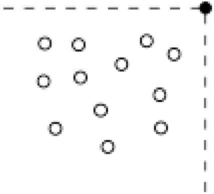
Output-Sensitive Analysis

- 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.



Output-Sensitive Analysis

- 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 $O(nlogh)$ 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.

KS is $O(nlogh)$ 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 $O(nlogh)$. 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,

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

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

Proof Continued

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

Proof Continued

$$\begin{aligned}
T(n,h) &\le \max_{h_l+h_r=h}\{T(\frac{n}{2},h_l)+T(\frac{n}{2},h_r)\}+cn\\
&\le \max_{h_l+h_r=h}\{c\frac{n}{2}log(h_l)+c\frac{n}{2}log(h_r)\}+cn
\end{aligned}$$

Proof Continued

$$
\begin{aligned}
T(n,h) &\leq \max_{h_l+h_r=h}\{T(\frac{n}{2},h_l)+T(\frac{n}{2},h_r)\}+cn \\
&\leq \max_{h_l+h_r=h}\{c\frac{n}{2}log(h_l)+c\frac{n}{2}log(h_r)\}+cn \\
&\leq cn+\frac{1}{2}cn\max_{h_l+h_r=h}\{log(h_lh_r)\}
\end{aligned}
$$

Proof Continued

$$
\begin{aligned}
T(n,h) &\le \max_{h_l+h_r=h}\{T(\frac{n}{2},h_l)+T(\frac{n}{2},h_r)\}+cn \\
&\le \max_{h_l+h_r=h}\{c\frac{n}{2}log(h_l)+c\frac{n}{2}log(h_r)\}+cn \\
&\le cn+\frac{1}{2}cn\max_{h_l+h_r=h}\{log(h_lh_r)\} \\
&\le cn+\frac{1}{2}cn\Big(log\Big(\frac{h}{2}\Big)^2\Big)
\end{aligned}
$$

Proof Continued

$$
\begin{aligned}
T(n,h) &\leq \max_{h_l+h_r=h} \{T(\frac{n}{2}, h_l) + T(\frac{n}{2}, h_r)\} + cn \\
&\leq \max_{h_l+h_r=h} \{c\frac{n}{2}log(h_l) + c\frac{n}{2}log(h_r)\} + cn \\
&\leq cn + \frac{1}{2}cn \max_{h_l+h_r=h} \{log(h_l h_r)\} \\
&\leq cn + \frac{1}{2}cn\left(log\left(\frac{h}{2}\right)^2\right) \\
&\leq cnlog(h)
\end{aligned}
$$

A more Fine-Grained Analysis

- For $h \in O(1)$, the runtime of KS is linear. For $h \in O(n)$, the runtime is $nlog(n)$.
- But even for many h in between the algorithm preforms quite-well!

A more Fine-Grained Analysis

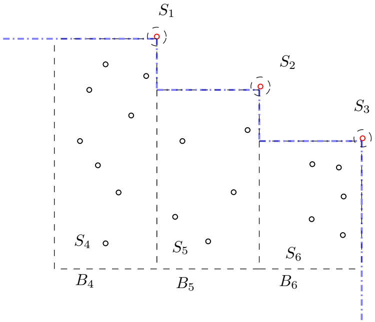
- For $h \in O(1)$, the runtime of KS is linear. For $h \in O(n)$, the runtime is $nlog(n)$.
- But even for many h in between the algorithm preforms quite-well!
- Why? Many points are dominated by q and removed, resulting in fewer points for recursive calls.
- To explore this more, we need to parameterize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

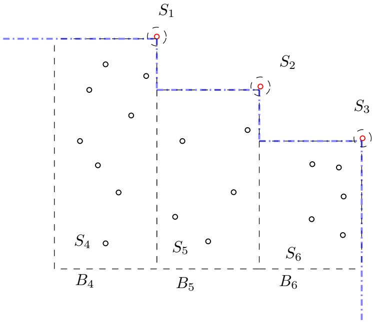
Legal Partitions

- 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.



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.



Instance Optimal Upper Bound on KS Algorithm

- 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.

Instance Optimal Upper Bound on KS Algorithm Proof

- 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.

Instance Optimal Upper Bound on KS Algorithm Proof

- 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:

Instance Optimal Upper Bound on KS Algorithm Proof

- 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:

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

Instance Optimal Upper Bound on KS Algorithm Proof

- 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:

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

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

Proof Continued

$$\leq |S_i|\Big(log(n/|S_i|) + 3\Big)$$

- 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.

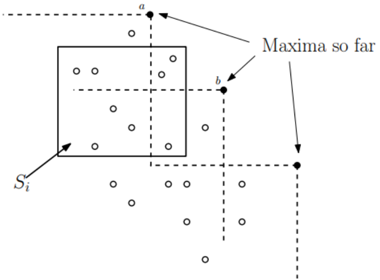
Proof of Claim

- **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:

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:

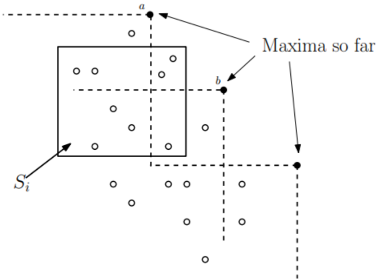
- Recall that S_i is contained in a box B_i. **Any points of S_i not yet removed must be contained in between two previously identified maximal points (along x-axis)**



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:

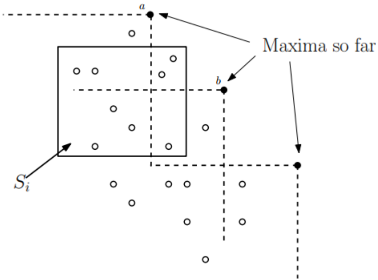
- Recall that S_i is contained in a box B_i. **Any points of S_i not yet removed must be contained in between two previously identified maximal points (along x-axis)**
- All points in S_i have x-coord. less than b's x-coord as b is maximal.



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.

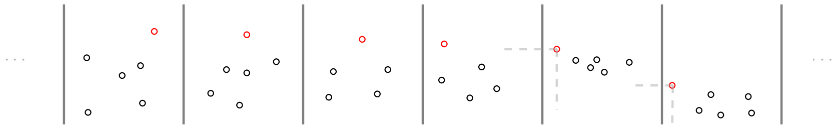


Proof of Claim Continued

- **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.**

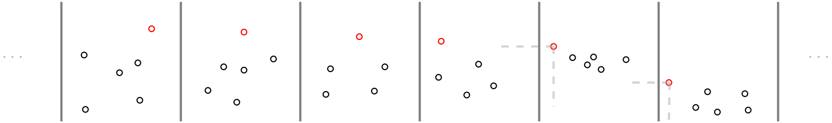
Proof of Claim Continued

- **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.



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,

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

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,

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

- **Note:** When each S_i is a singleton, we have the $O(nlogn)$ 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 $O(nlogh)$ bound.

A Matching Lower Bound for Instance Optimality?

- 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?

A Matching Lower Bound for Instance Optimality?

- 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.

A Matching Lower Bound for Instance Optimality?

- 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.
- 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 $KS(Q)$.

A Matching Lower Bound for Instance Optimality?

- 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.
- 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.

A Matching Lower Bound for Instance Optimality

- 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 π.

A Matching Lower Bound for Instance Optimality

- 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 π.
- Then for every point set S and every algorithm B,

$$Cost(B, S) \in \Omega\Big(\min_{legal\{S_i\}} \sum_{i=1}^{k} |S_i| log \frac{n}{|S_i|}\Big) \tag{2}$$

A Matching Lower Bound for Instance Optimality

- 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 π.
- Then for every point set S and every algorithm B,

$$Cost(B, S) \in \Omega\Big(\min_{legal\{S_i\}} \sum_{i=1}^{k} |S_i| log \frac{n}{|S_i|}\Big) \tag{2}$$

- **Proof outline:** For any correct algorithm A with input S, there exists a permutation of S on which at least $\Omega\Big(\min_{legal\{S_i\}} \sum_{i=1}^{k} |S_i| log \frac{n}{|S_i|}\Big)$ comparisons are made.

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.

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.

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.

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.
- 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 $\min\limits_{legal\{S_i\}} \sum_{i=1}^{k} |S_i| log \frac{n}{|S_i|}$.

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.

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?

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 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 optimaility 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*