What is *output sensitivity*? Design & analyze algorithms for the convex-hull problem

Output sensitivity (informal). An algorithm is *output sensitive* when its running time depends not only on the input size n but also on the size h of the output. For convex hull in the plane, n is the number of input points and h the number of points on the hull. An output-sensitive convex-hull algorithm runs in time T(n,h) that is smaller when h is small.

Algorithms

1. **Graham scan**
    - Idea: sort points by polar angle about a pivot, then scan to build hull using a stack.
    - Time: O(nlogn) dominated by sorting. Not output sensitive.
2. **Jarvis march (Gift wrapping)**
    - Idea: start at extreme point, repeatedly find the next hull point by scanning all points for the most counterclockwise point.
    - Time: O(nh). Output sensitive — good when h is small (e.g., constant hull size).
3. **Chan's algorithm (combines Jarvis + Graham)**
    - Idea: partition points into groups of size m, compute hulls of groups (Graham), then do a Jarvis-type wrap using tangents to group hulls; choose m and iterate doubling until full hull found.
    - Time:O(nlogh). This is output sensitive and optimal for comparison-based models up to constant factors.
4. **Kirkpatrick–Seidel ("ultimate planar convex hull")**
    - Time: O(nlogh) as well (divide-and-conquer using a "marriage-before-conquest" step).

Design sketch for Chan's algorithm (practical output-sensitive method):

- Pick parameter m (e.g. try m=2,4,8,…).
- Partition input into $\lceil n/m \rceil$ groups of at most m points.
- Compute each group's convex hull by Graham scan in O(mlogm) per group → total $O((n/m)\cdot m\log m)=O(n\log m)$.
- Use a Jarvis-step that finds the next hull vertex among all group hulls by computing tangent points to each group hull in O(logm) per group; each step costs O((n/m)logm).
- If actual $h \le m$, Jarvis completes in h steps. Choose m doubling until it succeeds. Overall cost gives O(nlogh).

Example (Convex Hull)

Input points:
A(1,1), B(2,5), C(5,4), D(6,1), E(3,3), F(4,2)

Visual hull points = A → B → C → D (4 points).

Output-sensitive runtimes:

- n = 6, h = 4
- **Jarvis march:** O(nh) = 6×4 = 24 ops
- **Graham scan:** O(n log n) = 6 log 6 ≈ 15 ops
- **Chan's:** O(n log h) = 6 log 4 ≈ 12 ops

If h were only 2 or 3, Jarvis becomes much faster — this shows output sensitivity.

Define NP-complete and the Vertex-Cover problem. Is Vertex-Cover NP-complete? Justify.

Definitions.

- **NP:** class of decision problems for which a "yes" instance has a certificate (witness) verifiable in polynomial time by a deterministic Turing machine.
- **NP-hard:** a problem L is NP-hard if every problem in NP can be polynomial-time reduced to L. Informally: L is at least as hard as every problem in NP.
- **NP-complete:** a problem is NP-complete if it is in NP *and* NP-hard. These are the hardest problems in NP.

Vertex-Cover (decision form).
Input: Graph **G=(V,E) and integer k.**

Question: Is there a subset $C \subseteq V$ with $|C| \leq k$ such that every edge has at least one endpoint in

C? (Such a C is a vertex cover.)
Vertex-Cover is NP-complete:

1. **Vertex-Cover ∈ NP.** Certificate: a set C of ≤ k vertices. We can check in polynomial time that every edge has an endpoint in C.
2. **Vertex-Cover is NP-hard.** Standard polynomial-time reduction from **Independent Set** (or from CLIQUE) which are known NP-complete:

- Independent Set decision: given G and k, is there an independent set S of size ≥ k (no edges among vertices in S)?
- Reduction: Given ((G,k) for Independent Set, produce (G,k′) for Vertex Cover with $k' = |V| - k$.
- Correctness: S is an independent set of size k iff $V \setminus S$ is a vertex cover of size $|V| - k$. This is a polynomial, simple transformation, so Vertex-Cover is NP-hard.

3. Combined ⇒ Vertex-Cover is NP-complete.

Example

Graph:
Edges: (A,B), (B,C), (C,D), (A,D)

Vertex cover of size 2 exists?

Try C = {B, C}:

- (A,B) covered by B
- (B,C) covered by B or C
- (C,D) covered by C
- (A,D) NOT covered → FAIL

Try C = {A, C}:

- (A,B) covered by A
- (B,C) covered by C
- (C,D) covered by C
- (A,D) covered by A → SUCCESS

Thus vertex cover size ≤ 2 exists.

NP-hardness example using reduction from Independent Set

Graph: A–B–C–D as above.
Independent set of size 2: {A, C}.
Vertex cover size = $|V| - |IS| = 4 - 2 = 2$ → matches above.

This shows the reduction works.

Plane-sweep algorithm for line-segment intersection detection in O(nlogn)

Problem: Given set S of n segments in the plane, determine whether any pair intersects.

Algorithm (sweep line — Bentley–Ottmann style simplified for detection):

Data structures:

- Event queue Q: all segment endpoints, sorted by x-coordinate (ties broken by y). (Use balanced BST or priority queue.) Size 2n.
- Status structure T: ordered set (balanced BST) of segments currently intersecting the vertical sweep line, ordered by intersection point's y-coordinate with the sweep line. Supports insert, delete, predecessor, successor in O(logn).

Procedure:

1. Initialize Q with all endpoints (mark as left or right endpoint for each segment; for vertical segments define ordering consistently).
2. While Q not empty:
    - Extract next event point p (left-to-right). Let the set of segments whose endpoint is p be processed.
    - If p is a left endpoint of segment s: insert s into T; check s's immediate predecessor and successor in T for intersection with s. If any intersection found → return "yes".
    - If p is a right endpoint of segment s: find predecessor u and successor v of s in T. Delete s from T. If both u and v exist, check u vs v for intersection.
    - (If you want to report all intersections, you must also enqueue intersection events when two neighbors swap; for detection-only, neighbor checks suffice.)
3. If finish without finding intersection → return "no".

Correctness sketch: If two segments intersect, at the x-coordinate of their leftmost intersection the sweep line will have them adjacent in T at some event; the predecessor/successor checks find the intersection. The algorithm maintains the invariant that T is the vertical order of segments intersecting the sweep line.

Time complexity:

- Sorting endpoints: O(nlogn).
- Each endpoint causes O(1) BST ops (insert/delete/find neighbors): O(logn) each → total O(nlogn).
- Intersection tests are constant time each; only neighbor pairs are tested O(n) times. Thus overall O(nlogn).

(If you must handle segments that share endpoints or collinear overlaps, handle tie-breaking and special cases carefully; but complexity unchanged.)

Example

Segments:
s1: A(1,1) → B(5,5)
s2: C(1,5) → D(5,1)
s3: E(6,6) → F(7,7)

Sweep line moves left → right.

Events sorted by x:
x = 1: A, C
x = 5: B, D
x = 6: E
x = 7: F

Processing at x=1:
Insert s1 and s2. They are neighbors in the sweep-line tree; test intersection → they cross at (3,3).
Return "YES".

Hence O(n log n) detects intersection.

Malware / logs problem → decision version is NP-complete

Problem restatement (formal).

- Universe U: all websites.
- For each infected machine i we have subset Si ⊆U (sites visited by that machine).
- We seek a smallest collection H⊆U such that H∩Si ≠ ∅ for every infected machine i. This is the *hitting set* problem (equivalently, the *set cover* dual).

Decision version: Given sets **{S1 ,…,Sm }⊆U and integer**

k
k, is there a set H⊆U with ∣H∣≤kthat intersects every Si ?

Show NP-complete:

1. **In NP.** Certificate is the set H of at most k websites; verify in polynomial time that H∩Si ≠ ∅ for all i.
2. **NP-hard.** Standard reduction from Set Cover (or vice versa). For example, Set Cover is NP-complete: given universe U and family F={Fj }, ask if there's ≤ k sets whose union is U. Convert to hitting set by letting infected machines correspond to elements u∈U and for each

u define Su ={j|u∈Fj }(the indices of sets covering u). A hitting set of size ≤ k(choice of indices) corresponds exactly to a set cover of size ≤ k. This polynomial reduction shows hitting set decision is NP-hard. Hence NP-complete.

Therefore the decision version of "smallest collection of websites visited by all infected computers" is NP-complete.

Example

3 infected computers, their visited websites:

PC1: {A, B, C}
PC2: {B, D}
PC3: {A, D}

Goal: smallest set H that touches all.

Try size k=1:

- H={A}: covers PC1 & PC3 but not PC2 (which needs B or D)
- H={B}: covers PC1 & PC2 but not PC3
- H={D}: covers PC2 & PC3 but not PC1
  No solution size 1.

Try size k=2:

- H={A, B}:
  - PC1 visited A/B → OK
  - PC2 visited B → OK
  - PC3 visited A → OK

Thus minimum hitting set = {A, B}, size 2.

This direct maps to a SET COVER instance → NP-complete.

Christofides' algorithm for Metric-TSP and 3/2 guarantee

Instance: Complete graph G=(V,E) with nonnegative edge weights satisfying triangle inequality.

Christofides' algorithm (steps):

1. Compute a Minimum Spanning Tree (MST) T of G.
2. Let OO be the set of vertices with odd degree in T. |O| is even.

3. Compute a minimum-weight perfect matching M on the induced complete subgraph on O (weights are still metric distances).

4. Form multigraph H by taking edges of T∪M. Every vertex now has even degree → H is Eulerian.

5. Find an Eulerian tour of H. Shortcut repeated vertices using triangle inequality to obtain a Hamiltonian cycle (a TSP tour).

Complexity: MST: O(|E|log|V|). Minimum perfect matching on |O| nodes can be done in polynomial time (blossom algorithm) — polynomial but potentially higher; overall polynomial.

Approximation bound proof sketch (≤ 3/2 OPT):

- Let OPT denote length of optimal TSP tour.
- **(MST ≤ OPT)**: Remove an edge from optimal TSP tour gives a spanning tree of cost ≤ OPT, so MST cost c(T)≤OPT.
- **(Matching ≤ OPT/2)**: Consider the optimal tour. Restrict the tour to the vertices in O and take every other edge along the tour to pair up the odd vertices — with triangle inequality this provides a matching whose total cost ≤ OPT/2. Since M is a minimum perfect matching on O, c(M)≤OPT/2.
- Summing: c(T)+c(M)≤OPT+OPT/2=3OPT/2.
- The Eulerian tour of T∪M has cost c(T)+c(M). Shortcutting (removing repeated vertices) cannot increase cost due to triangle inequality. So the final TSP tour cost ≤ 3/2 OPT.

Thus Christofides returns a tour with cost at most 3/2 times optimal.

Example

Complete graph with vertices {A,B,C,D} and metric distances:

	A	**B**	**C**	**D**
A	0	4	6	7
B	4	0	3	5
C	6	3	0	2
D	7	5	2	0

Step 1 — MST

Pick minimum edges:
C–D (2), B–C (3), A–B (4)
MST cost = 2 + 3 + 4 = 9

Odd-degree vertices = {A, D}

Step 2 — Perfect Matching

Match A–D (cost 7)

Total = 9 + 7 = 16

Step 3 — Euler Tour & Shortcut

Euler tour: A → B → C → D → A
No shortcuts needed.

Cost = 16
Optimal TSP = 14
So Christofides = 16 ≤ 1.5 × 14 = 21
(Guarantee holds).

Approximation algorithms for Vertex Cover and Set Cover

Vertex Cover — 2-approximation (simple greedy / matching based)

Algorithm (edge-picking):

- Initialize cover $C=\varnothing$.
- While there is an uncovered edge (u,v):
  - Add both endpoints u and v to C.
  - Remove all edges incident to u or v.
- Return C

Analysis:

- Each iteration picks an edge and forces adding 2 vertices; set of picked edges forms a matching M (no two picked edges share a vertex). Any vertex cover must pick at least one endpoint from each matching edge, so $|OPT|\geq|M|$. Our cover size $|C|=2|M|\leq 2|OPT|$. So 2-approximation.
- Time: O(m) if implemented carefully.

Alternative: LP relaxation + rounding or maximal matching give same 2-approx ratio.

Set Cover — greedy algorithm ($H_n \approx \ln n$ approximation)

Problem: Universe U of n elements, family F of subsets. Pick ≤ k sets to cover U with minimum number of sets.

Greedy algorithm:

- Start with C=∅, uncovered elements R=U.
- While R ≠ ∅: pick the set S∈F that covers the largest number of uncovered elements ∣S∩R∣. Add S to C and remove covered elements from R.
- Return C.

Approximation factor and proof idea:

- Greedy achieves a Hn approximation where Hn =1+21 +31 +⋯+n1 ≤lnn+1.
- Proof (standard): Let OPT be size of optimal cover. At each greedy step, the chosen set covers at least 1/OPTfraction of remaining uncovered elements. This yields a geometric decay in uncovered elements; summing gives ≤ Hn ⋅OPT sets chosen.

Complexity: naive implementation O(nm) or with clever book keeping O(mlogm+total set sizes).

Other methods: Primal-dual or LP rounding give same O(lnn) guarantee; hardness results show Ω(logn)approximation lower bound (unless P=NP), so greedy is essentially optimal.

A. Vertex Cover 2-Approx Example

Graph edges: (A,B), (B,C), (C,D), (D,E)

Greedy algorithm picks an uncovered edge each time:
Pick (A,B) → add A,B
Pick (C,D) → add C,D
Now E is isolated → done

Cover = {A, B, C, D} (size 4)

But optimal = {B, C, D} (size 3)

Ratio = 4/3 ≤ 2 → valid 2-approx.

B. Set Cover Greedy Example

Universe U = {1,2,3,4,5}
Sets:
S1 = {1,2}
S2 = {2,3,4}
S3 = {4,5}
S4 = {1,5}

Greedy picks:

1. S2 (covers 3 elements: 2,3,4)
   Remaining: {1,5}
2. S4 (covers {1,5})

Total sets used: 2
Optimal is also 2, so greedy = optimal in this example.