Reductions

- Informally, we say problem A *reduces to* problem B (A ≤ B) if an instance of A can be solved by
  - Calling a subroutine that can solve B
  - Doing a small amount of additional work before and/or after calling B

```
SolveA(InstanceOfA):
    Use InstanceOfA to build an
        InstanceOfB;
    s = SolveB(InstanceOfB);
    Use s to build solution for
        InstanceOfA;
    return solution;
end.
```

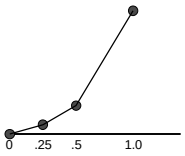

Comparing Problems

If we know A ≤ B then

- Intuitively,
  A is "easier" than B and B is "harder" than A
- Knowing how long B takes give us a bound on how long A takes:
  $T_A \leq T_B$ + translation-time
- If it's known that A is hard (i.e., it requires a certain amount of time) and if the translation is cheap then B must also be hard

Claim: Sorting ≤ ConvexHull

- To show this, we need to build a routine that can sort any set of numbers by making a subroutine call to ConvexHull




```
Sort(X[1..n]):
    for i = 1 to n do
        Point p[i] = (X[i],X[i]^2);
    array ch = ConvexHull(p);
    for i = 1 to n do
        s[i] = ch[i].x;
    return s;
end.
```


Linear Time Reductions

- Technically, the examples we've done so far have all used *linear-time reductions*: in other words, the translation parts have taken just linear time
- We should be using $\leq_L$ to indicate the linear-time
- We'll see other kinds of reductions
  $\leq_P$ for polynomial-time reductions
- The type of reduction will usually be clear from context

Reductions among Geometric Problems

- All Nearest Neighbors
  - Given a set of points in the plane, report a nearest-neighbor for each point
- Closest Pair
  - Given a set of n points in the plane, which two points are the closest?
- Euclidean MST
  - Given n points in the plane, find the Minimum Spanning Tree of the complete (Euclidean) graph
- Triangulation
  - Given n points in the plane, find a triangulation of these points
- Sorting
  - Given a set of n numbers, report the numbers in order
- Element Uniqueness
  - Given a set of n numbers, are the numbers all distinct?

This set of problems is from [Preparata & Shamos]

Element Uniqueness ≤ Closest Pair

- We accept two lower bounds as given
  - Element Uniqueness requires Ω(n log n) time
  - Sorting requires Ω(n log n) time (we proved this)

Claim: Element Uniqueness ≤ Closest Pair

Proof:

- Convert the n numbers into points on the x-axis
- Find the closest pair of points
- The closest pair is distance zero iff the initial elements were not unique

Conclusion: Closest Pair requires Ω(n log n) time

Closest Pair ≤ All Nearest Neighbors

Claim: Closest Pair ≤ All Nearest Neighbors

Proof:

- Given the input set, find all the nearest neighbors
- Examine each point to see which one has the nearest neighbor that is the closest
- Report that pair as the closest pair

Conclusion: All Nearest Neighbors requires $\Omega(n \log n)$ time

Closest Pair ≤ EMST

Claim: Closest Pair ≤ EMST

Proof:

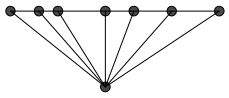
- Given the set of input points, form the EMST
- We know the EMST is the MST of the compete graph on the set of input points and we know from Kruskal's Algorithm that the shortest edge in the graph is a member of the MST
- Thus we can check each edge of the EMST to find the shortest; this will tell us the closest pair

Conclusion: EMST requires $\Omega(n \log n)$ time

Sorting ≤ Triangulation

Claim: Sorting ≤ Triangulation

Proof:

- Convert the initial numbers into points on the x-axis
- Add a new point at (0,-1)
- Form the triangulation of these points
- Find the leftmost point on the x-axis
- We can report the numbers in order by traversing the triangulation graph

Conclusion: Triangulation requires $\Omega(n \log n)$ time