day 2 1/14/98

1 Sorting vs. Convex hull

1.1 sorting algorithms

A lower bound on the time complexity of sorting algorithms given only the use of a compare primitive, *compare(a,b)*, which returns (a $<$ b) is $\Omega(\log_2 n!) \approx \Omega(n \log_2 n)$. You can see this by noting that there are $n!$ possible outcomes to sorting n elements which implies that $\log_2 n!$ choices must be made.

1.1.1 Insertion sort

Insert each element into a (balanced) binary tree, then traverse the tree in order to find the sort

1.1.2 heap sort

Place all the elements into a heap and pull them out of the heap in order

1.1.3 merge sort

Split the elements into 2 equal size subsequences, sort each subsequence, then merge the two sorted subsequences.

1.1.4 median sort

deterministic version let Sort(P) =
m = Median(P)
return $Sort(\{P_i : P_i < m\})$:m: $Sort(\{P_i : P_i > m\})$

This differs from merge sort, becuase the work at a particular level occurs before recursing so only a concatentation is necessary on return.

randomized version Otherwise known as quicksort.
let QSort(P) =
m = Random(P)
return $Sort(\{P_i : P_i < m\})$:m: $Sort(\{P_i : P_i > m\})$

1.2 Convex hull algorithms

1.2.1 The primitive operations of convex hull algorithms

- testing if 2 line segments intersect, *bool Intersect(Line_ segment_ 1,Line_ segment_ 2)*
- testing if a point lies to the left or to the right the line derived from a line segment, *LR_ test(Line_ segment,Point)* $= Sign(Det(\begin{bmatrix} LS_{11} & LS_{12} & 1 \\ LS_{21} & LS_{22} & 1 \\ P_{31} & P_{32} & 1 \end{bmatrix}))$

where LS_{1j} = the components of the starting point of the line segment and LS_{2j} = the components of the ending point of the line segment. If the output is positive, *Point* lies to the left of the line defined by the line segment

1.2.2 definitions

- $A \subseteq R^2$ is a set in the plane
- $CH(A)$ = simplest convex set conatining A
- $A \subseteq R^2$ is convex if forall $\forall p_1, p_2 \in A \Rightarrow LineSegment(P_1, P_2) \subseteq A$

1.2.3 Convex hull heap sort analog = Graham scan

Graham Scan($P_1, .., P_n$) =

1. pick $P = P_i$ with minimum y value.
2. compute $\theta_j = slope(P, P_j) \forall j \neq i$
3. sort slopes $Sort(\theta_1, ..., \theta_{n-1})$
4. push P_1, P_2 onto stack C
5. while $size(C) \geq 2$ let $C_1 = Pop(C), C_2 = Pop(C)$ if *LR_test(Line_segment(*C_1, C_2*),*P_i*)=R then* $Pop(C)$ *else* $Push(P_i); i := i + 1$
6. push P and return C

Heuristically, this algorithm winds counterclockwise around the points, unwinding to remove points not in the convex hull every time it discovers (by *LR_test*) that a point is not in the convex hull. The time complexity of this algorithm is dominated by the sort, $O(n \log n)$.