

I) Sorting problems

- sort any type of data
- sort() function calls back the object's compareTo() method as needed
- Comparable<Item> interface
 - public in compareTo(Item that)

(-1) less
(+1) greater
(0) = equals

Total Order

- antisymmetry: if $v \leq w$ and $w \leq v$
 $\Rightarrow v = w$
- transitivity: if $v \leq w$ and $w \leq x$
 $\Rightarrow v \leq x$
- totality: either $v \leq w$ or $w \leq v$
or both

Comparable API

- is a total order
- incompatible types or null \Rightarrow

throw an exception

Sorting abstractions

(Less) is item v less than w ?

(Exchange) swap item in array $a[]$ at index i with the one at index j

Selection sort

- in iteration i , find index min of smallest remaining entry
- swap $a[i]$ with $a[\text{min}]$
- * algorithm: scans from left to right \uparrow
- * invariants: entries to the left (and including) the \uparrow are smaller
fixed and ASC

- no entry to the right of \uparrow is smaller than any entry to the left of the \uparrow
- Selection sort uses $(N-1) + (N-2) + \dots + 1 + 0$
 $\sim \left\lfloor \frac{N^2}{2} \right\rfloor$ compares
- running time = quadratic time, regardless of input (even if input is sorted)
- data movement is minimal \rightarrow linear number of exchanges

Insertion Sort

- in iteration i , swap $a[i]$ with each larger entry to its left
- algorithm: \uparrow scans from left to right
- invariants: entries to the left of \uparrow are in ASC (including \uparrow)
 entries to the right of \uparrow have not yet been seen

* mathematical analysis

$\sim 1/4 N^2$ compares

$\sim 1/4 N^2$ exchanges on average

◦ proof: expect each entry to move halfway back ↗ on average

(approx. twice as fast as selection sort)

◦ best case: array already sorted

$N-1$ compares, 0 exchanges

◦ worst case: array is in descending order

(no duplicates) $\sim 1/2 N^2$ compares
 $\sim 1/2 N^2$ exchanges

→ worst than selection sort

Inversion = pair of keys that are out of order

array = partially sorted if no. of inversions
is $\leq cN$

* for partially sorted arrays, insertion sort runs in linear time.

Shell sort

• idea = move entries more than one position at a time by h -sorting

the array
 h -sorted array = h interleaved sorted
subsequences

(shell sort) h -sort array for decreasing
sequence of values of h

h -sort = insertion sort, with slide
length h

Why insertion sort?

- Big increments, small subarray
- small increments, nearly in order

! a g-sorted array remains g-sorted after h-sorting it

- which increment sequence do we use?

1) powers of two? — no

2) powers of two — 1

3) $3x+1$ easy to compute

Analysis

• worst case is $O(N^{3/2})$

• tiny, fixed code footprint, used in embedded systems

Shuffle sort

• generate a random real number for each array entry

• sort the array based on the random generated numbers

Shuffle sort produces a uniformly random permutation of the input array (no duplicate values)

(goal) rearrange array so that result is a uniformly random permutation in linear time

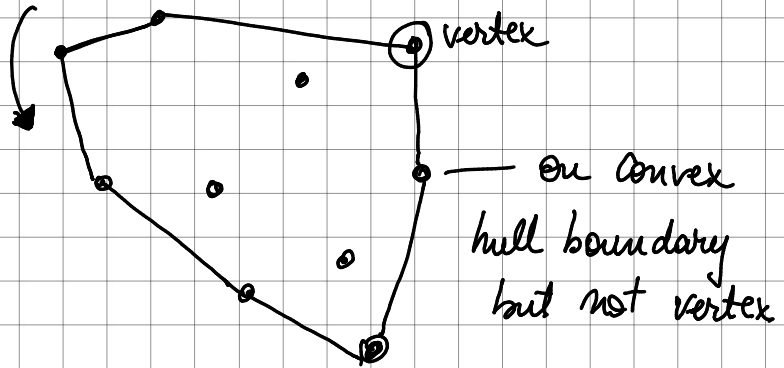
Knuth Shuffle

- in iteration i , pick integer r between 0 and i uniformly at random
- swap $a[i]$ and $a[r]$

! number of permutations of n distinct elements is $(n!)$

Convex Hull

- of a set of n points is the smallest perimeter fence enclosing the points



Convex hull output = sequence of vertices
in counter clockwise order

* Robot motion planning

- shortest path from s to t
 - straight line
 - one of two polygonal chains of convex hull

- (fact i): you can traverse the convex hull by making only counterclockwise turns
- (fact ii): the vertices of convex hull appear in increasing order of polar angle with respect to point p with lowest y coordinate

Graham Scan

- choose point p with smallest y coordinate
- sort points by polar angle with p
- consider points in order, discard unless it creates a counterclockwise turn

→

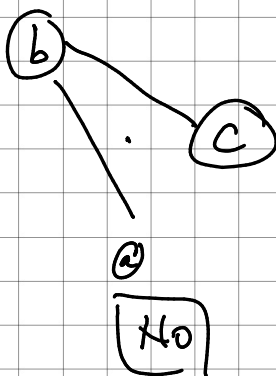
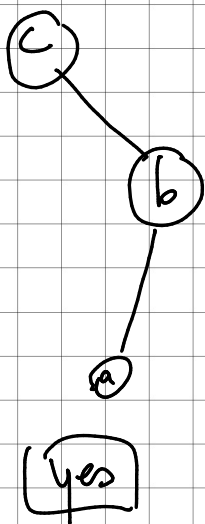
Q: how to find point p with the smallest y coordinate? def. total order, compare y -coordinate

Q: How to sort points by polar angles with respect to p ?

- define a total order for each point p

Q: how to figure out if $p_1 \rightarrow p_2 \rightarrow p_3$ is a counterclockwise turn?

* Given 3 points / $a \rightarrow b \rightarrow c$ is a counterclockwise turn?



is c to the left of \overline{ab} ?

* same line

* ccw - slope

-
- implement Convex hull
 - Dijkstra's two-stack algorithm