I) Sorting problems

- sort any type of data
- sort() function calls back the object's compareTo() method as needed
- Comparable<Item> interface
  - public int 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 => 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[min]
  - * algorithm: scans from left to right
  - * invariants: entries to the left (and including) of the i are smaller

Fixed and ASC

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

Insertion sort

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

* mathematical analysis

~ 1/4 N^2 compares

~ 1/4 N^2 exchanges on average

- **proof**: expect each entry to move halfway back

(approx. twice as fast as selection sort) ← on average

- **best case**: array already sorted

$N-1$ compares, 0 exchanges

- **worst case**: array is in descending order

(no duplicates) ~ 1/2 N^2 compares

~ 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.

Shellsort

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

The array

h-sorted array = h interleaved sorted subsequences

Shellsort h-sort array for decreasing sequence of values of h

h-sort = insertion sort, with stride 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 $\Theta(N^{3/2})$
- 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

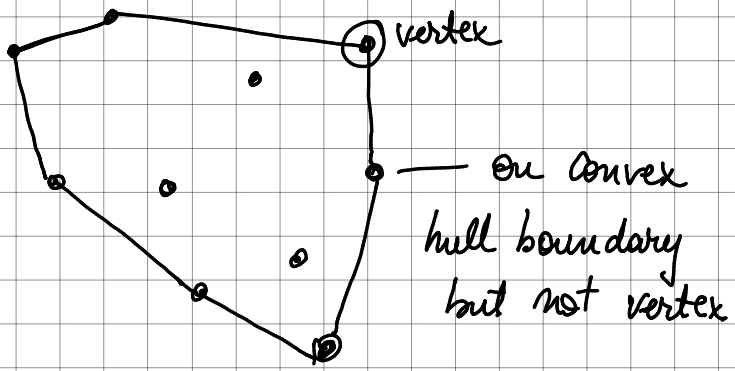
Knuth Shuffle

- 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 counterclockwise order