Elementary Sorting

1 Selection Sort

```
for i in [0,...n):
    find index of minimum
    swap(i, i_min)
```

Uses $(n-1)+(n-2)+\ldots+1+0 \approx \frac{N^2}{2}$ comparisons and N exchanges, giving $O(n^2)$ running time no matter the size/sort order of input array.

2 Insertion Sort

```
for i in [1,...n):
    j = i
    while (a[j-1] > a[j]):
        swap(j, --j)
```

To sort a random array with distinct keys, insertion sort uses $\approx \frac{1}{4}N^2$ comparisons and $\approx \frac{1}{4}N^2$ exchanges on average, because given a random data set, we can expect each entry to move halfway back.

If the array is sorted, insertion sort makes $N-1$ comparisons and 0 exchanges. If the array is reverse-sorted, it makes $\approx \frac{1}{2}N^2$ comparisons and $\approx \frac{1}{2}N^2$ exchanges.

In a **partially sorted** array with $\leq cN$ inversions, insertion sort runs in linear time and *the number of exchanges is equal to the number of inversions*, and the number of comparisons is $exchanges + (N-1)$.

3 Shellsort

Idea: insertion sort that moves entries more than one position at a time by *h-sorting* the array. An h-sorted array is h interleaved, sorted subsequences - every pair separated by a distance of h is sorted.

Shellsort h-sorts an array for a decreasing sequence of values h. Use insertion sort because large increments at the beginning means small subarrays, and when the increment grows smaller, the array is very nearly sorted. The mathematical fact that makes shellsort possible is that a g-sorted array is still g-sorted after being h-sorted.

Knuth recommends using the sequence $x_n = 3x_{n-1}+1$ for shellsort sizes, and Sedgewick recommends a merge of $(9\times 4!)-(9\times 2^i)+1$ and $4^i-(3\times 2^i)+1$ that goes $1, 5, 19, 41, 109, 209, 505, 929, 2161, 3905\ldots$.

The worst-case number of comparisons using the $3x+1$ model is $O(n^{3/2})$, and nobody has determined an accurate model for the average-case running time of the algorithm yet.

4 Applications of Sorting

4.1 Shuffling

Shuffle sort: generate a random real number for each array entry, then sort. Provided no duplicate values, this randomly permutes the array. However, this is lower-bounded by $O(n \log n)$ due to the limit on comparison-based sorting.

The Knuth shuffle is an algorithm that shuffles an array to a uniformly random permutation in linear time.

```
for i in [0,...n):
    swap(i, rand(0, i))
```

4.2 Convex Hull

The **convex hull** of a set of N points is the smallest perimeter convex "fence" enclosing the points. Equivalent definitions include the smallest convex set containing all points, smallest area convex polygon enclosing the points, or the convex polygon enclosing the points whose vertices are points in the set.

Important properties of the convex hull:

1. Can traverse by making only CCW turns
2. The vertices appear in increasing order of polar angle with respect to point p with the lowest y-coordinate.

One way to find the convex hull is the **Graham scan**:

1. Choose point p with smallest y-coordinate
2. Sort points by polar angle with p
3. Consider points in order: if one point leads to a clockwise turn with any point after it in order, discard it and connect the vertex before it with the connected vertex. This is very naturally implemented with a stack.