Results and Observations (lecture3 lab for COMS 5250)

Results:

```
=== Performance benchmarks ===

Base N0=20000, repeats=3, steps=4, pop_fraction=0.5, copy_batch=50


Average time (seconds)

N        pop_many   pop0_many      L[:]    list(L)   reverse()    [::-1]   x2(pop)  x2(pop0)

------------------------------------------------------------------------------------------------

20000      0.000375   0.017067   0.000028   0.000028   0.000034   0.000028     -      -

40000      0.000693   0.067681   0.000057   0.000054   0.000067   0.000055     1.85x
3.97x

80000      0.001403   0.295779   0.000112   0.000109   0.000139   0.000115     2.02x
4.37x

160000     0.002972   1.341805   0.000236   0.000236   0.000284   0.000235     2.12x
4.54x
```

Observations:

We ran benchmarks for N = 20000, 40000, 80000, 160000. For each N we first generated a base list of length N.

pop_many means: copy the base list into A, then remove k = N/2 elements using A.pop() (removing from the end) inside a loop, and time the whole operation.

pop0_many means: copy the base list into A, then remove k = N/2 elements using A.pop(0) (removing from the front) inside a loop, and time the whole operation. L[:] means: time making a copy of the list using slicing (base[:]); list(L) means: time making a copy using the constructor (list(base)); reverse() means: copy the list into A and time A.reverse() which reverses in place; [::-1] means: time base[::-1] which creates a new reversed list...

The measured times for pop_many were 0.000375 s, 0.000693 s, 0.001403 s, 0.002972 s as N doubled from 20000 to 160000, giving about 1.85x, 2.02x, and 2.12x growth, which likely is close to linear scaling because popping from the end is O(1) per operation and we perform k = N/2 removals.

The pop0_many times were 0.017067 s, 0.067681 s, 0.295779 s, 1.341805 s, giving about 3.97x, 4.37x, and 4.54x growth when N doubled. This can probably considered as consistent with quadratic scaling because each pop(0) forces shifting of the remaining elements, and doing that repeatedly accumulates about O(N^2) work.

For copying, L[:] took 0.000028 s, 0.000057 s, 0.000112 s, 0.000236 s and list(L) took 0.000028 s, 0.000054 s, 0.000109 s, 0.000236 s, which both scale roughly linearly because both copy N references into a new list.

For reversing, reverse() took 0.000034 s, 0.000067 s, 0.000139 s, 0.000284 s and [::-1] took 0.000028 s, 0.000055 s, 0.000115 s, 0.000235 s. Seems like both are roughly linear because both touch all N elements, but reverse() modifies in place while [::-1] allocates a new reversed list.