Activity 1. Time measurements for sorting algorithms.

In order to know whether a certain sample of executions follows its expected time complexity, we must set a rule of 3 where:

- Execution time 1, T1, is proportional to workload 1, n1
- And execution time 2, T2, is proportional to workload 2, n2

In our very case we know both workloads and one of the execution time. We will compute the next value to a given case, that is, computing T2. By applying the rule of 3, we can know the value of T2 as:

$$T2 = T1 * f(n2) / f(n1)$$

$$\text{Where } n2 / n1 = k^c,$$

Being k a constant and m the time complexity we want to compare against theoretical values.

This procedure will be the same for every case, just modifying the values of k and m. As so, it will be omitted for the sake of simplicity and just the values and explanations will be shown below.

Insertion algorithm

N	sorted(ms)	inverse(ms)	random(ms)
10000	0	69	54
20000	0	195	155
40000	1	526	517
80000	1	2253	2065
160000	1	15670	8581
320000	0	67239	36863
640000	1	379357	138745

- Sorted:
  - T2 (n=80 000) = 1 * 80 000 / 40 000 = 2ms
  - T2 (n=160 000) = 1 * 160 000 / 80 000 = 8ms

  - They do not not seem to follow the expected values(probably because of their very short execution time)
- Inversely sorted:
  - T2 (n=80 000) = 526* (40 000 / 20 000)^2 = 2104
  - T2 (n=160 000) = 15670 *(160 000 / 80 000)^2 = 62680ms
  - They seem to follow the expected values but not very closely
- Random:
  - T2 (n=40 000) = 517* (40 000 / 20 000)^2 = 2068ms
  - T2 (n=160 000) = 8581 *(160 000 / 80 000)^2 = 34324ms
  - They seem to follow the expected values closer than before.

Selection algorithm

N	sorted(ms)	inverse(ms)	random(ms)
10000	17	52	23
20000	41	197	73
40000	233	384	362
80000	490	1254	1124
160000	1751	4654	4913
320000	7564	23142	17969
640000	37737	95625	73150

- Sorted:
  - T2 (n=160 000) = 490 * (160 000 / 80 000)^2 = 1960ms
  - T2 (n=320 000) = 1751 * (320 000 / 160 000)^2 = 7004ms
  - They seem to follow the expected values but not very closely
- Inversely sorted:
  - T2 (n=160 000) = 1254 * (160 000 / 80 000)^2 = 5016ms
  - T2 (n=320 000) = 4654 * (320 000 / 160 000)^2 = 18616ms
  - They seem to follow the expected values but not very closely
- Random:
  - T2 (n=160 000) = 1124 * (160 000 / 80 000)^2 = 4496ms
  - T2 (n=320 000) =4913 * (320 000 / 160 000)^2 = 19652ms

- They seem to follow the expected values closer than before but still not that tightly.

Bubble algorithm

n	sorted(ms)	inverse(ms)	random(ms)
10000	113	69	90
20000	398	368	380
40000	302	1576	2057
80000	5998	7308	10990
160000	24193	30358	72403
320000	95575	153204	204486
640000	256968	577100	804514

- Sorted:
  - T2 (n=160 000) = 5998 * (160 000 / 80 000)^2 = 23992ms
  - T2 (n=320 000) = 24193 * (320 000 / 160 000)^2 = 96772ms
  - They seem to follow the expected values way tighter than any previous case.
- Inversely sorted:
  - T2 (n=160 000) = 7308 * (160 000 / 80 000)^2 = 29232ms
  - T2 (n=320 000) = 30358 * (320 000 / 160 000)^2 = 121432ms
  - They seem to follow the expected values relatively close.
- Random:
  - T2 (n=160 000) = 10990 * (160 000 / 80 000)^2 = 43960ms
  - T2 (n=320 000) = 72403 * (320 000 / 160 000)^2 = 289612ms
  - They do not seem to follow the expected values.

Quicksort algorithm by central position

n	sorted(ms)	inverse(ms)	random(ms)
10000	4	2	4
20000	2	5	6
40000	0	8	12
80000	9	11	10

160000	27	16	20
320000	4	34	32
640000	22	69	110
1280000	53	170	205

- Sorted:
    - T2 (n=160 000) = 27 * 2 * (log 2 + log 80 000) / log80 000 = 53.31ms
    - T2 (n=320 000) = 4 * 2 * (log 2 + log 160 000) / log 160 000 = 5.46ms
    - They do not seem to follow the expected complexity
- Inversely sorted:
    - T2 (n=160 000) = 11 * (160 000 / 80 000)^2 = 44ms
    - T2 (n=320 000) = 16 * (320 000 / 160 000)^2 = 64ms
    - They do not seem to follow the expected complexity
- Random:
    - T2 (n=160 000) = 20 * 2 * (log 2 + log 80 000) / log80 000 = 42.45 ms
    - T2 (n=320 000) = 32 * 2 * (log 2 + log 160 000) / log 160 000 = 67.7 ms
    - Again, the values are not that close the expected ones for their complexity.

Activity 2. QuicksortFateful

We are taking the first element in the array as the pivot, which leads to a permanent $O(n^2)$ complexity as every case would be identical; while another pivot selection strategy would make the average complexity to be O(n log n).

This means that fateful selection will be quite similar with very small workloads but way larger on the opposite case, which would be the only case for using this version. However, median of three or random are still more desirable options even on this very specific situation due to their average complexity over great workloads and similar one with not so big workloads.