Searching / Sorting. Thomas Li

Project 30.1

1. Selection Sort:

consider 40, 63, 64, 2, 87, 62, 45, 66, 99, 30, 31, 57.

(Pass Through Array) (determine min in unsorted part... move to sorted.)

→ Base: 40, 63, 64, 2, 87, 62, 45, 66, 99, 30, 31, 57

→ 1: 2, 63, 64, 40, 87, 62, 45, 66, 99, 30, 31, 57

→ 2: 2, 30, 64, 40, 87, 62, 45, 66, 99, 63, 31, 57

→ 3: 2, 30, 31, 40, 87, 62, 45, 66, 99, 63, 64, 57

→ 4: 2, 30, 31, 40, 87, 62, 45, 66, 99, 63, 64, 57

→ 5: 2, 30, 31, 40, 45, 62, 87, 66, 99, 63, 64, 57

→ 6: 2, 30, 31, 40, 45, 57, 87, 66, 99, 63, 64, 62

→ 7: 2, 30, 31, 40, 45, 57, 62, 66, 99, 63, 64, 87

→ 8: 2, 30, 31, 40, 45, 57, 62, 63, 99, 66, 64, 87 i = size −3

→ 9: 2, 30, 31, 40, 45, 57, 62, 63, 64, 66, 99, 87 ↗

→ 10: 2, 30, 31, 40, 45, 57, 62, 63, 64, 66, 99, 87 i = size −2

→ 11: 2, 30, 31, 40, 45, 57, 62, 63, 64, 66, 87, 99 ↗

Time: $O(n^2)$

Note, here, I've just followed the code on sort.cpp, davetcales.com lesson 24. iterates $0 \le i \le size-2$.

this algorithm is derived by the number of comparisons: $1+2+\dots+(n-3)+(n-2)+(n-1)$

$$= \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} \quad \text{Thus } \frac{n(n-1)}{2} = O(n^2)$$

because $\left|\frac{n(n-1)}{2}\right| \le \frac{n^2}{2} + \left|\frac{n}{2}\right|$

$\le n^2$ for $n \ge 0$.

2. Bubble sort (no swap flag)
continuously compare pairs and float max to top of unsorted list.

Pass:
→ Base: 40, 63, 64, 2, 87, 62, 45, 66, 99, 30, 31, 57
→ 1: 40, 63, 2, 64, 62, 45, 66, 87, 30, 31, 57, 99
→ 2: 40, 2, 63, 62, 45, 64, 66, 30, 31, 57, 87, 99
→ 3: 2, 40, 62, 45, 63, 64, 30, 31, 57, 66, 87, 99
→ 4: 2, 40, 45, 62, 63, 30, 31, 57, 64, 66, 87, 99
→ 5: 2, 40, 45, 62, 30, 31, 57, 63, 64, 66, 87, 99
→ 6: 2, 40, 45, 30, 31, 57, 62, 63, 64, 66, 87, 99
→ 7: 2, 40, 30, 31, 45, 57, 62, 63, 64, 66, 87, 99
→ 8: 2, 30, 31, 40, 45, 57, 62, 63, 64, 66, 87, 99
→ 9: 2, 30, 31, 40, 45, 57, 62, 63, 64, 66, 87, 99
→ 10: 2, 30, 31, 40, 45, 57, 62, 63, 64, 66, 87, 99
→ 11: 2, 30, 31, 40, 45, 57, 62, 63, 64, 66, 87, 99
→ 12: 2, 30, 31, 40, 45, 57, 62, 63, 64, 66, 87, 99

---

3. Insertion sort
Time: $O(n^2)$.

Pass:
→ Base: 40, 63, 64, 2, 87, 62, 45, 66, 99, 30, 31, 57
→ 1: 40, 63, 64, 2, 87, 62, 45, 66, 99, 30, 31, 57
→ 2: 40, 63, 64, 2, 87, 62, 45, 66, 99, 30, 31, 57
→ 3: 40, 63, 64, 2, 87, 62, 45, 66, 99, 30, 31, 57
→ 4: 2, 40, 63, 64, 87, 62, 45, 66, 99, 30, 31, 57
→ 5: 2, 40, 63, 64, 87, 62, 45, 66, 99, 30, 31, 57
→ 6: 2, 40, 62, 63, 64, 87, 45, 66, 99, 30, 31, 57
→ 7: 2, 40, 45, 62, 63, 64, 87, 66, 99, 30, 31, 57
→ 8: 2, 40, 45, 66, 62, 63, 64, 87, 99, 30, 31, 57
→ 9: 2, 40, 45, 66, 62, 63, 64, 87, 99, 30, 31, 57
→ 10: 2, 30, 40, 45, 66, 62, 63, 64, 87, 99, 31, 57
→ 11: 2, 30, 31, 40, 45, 66, 62, 63, 64, 87, 99, 57
→ 12: 2, 30, 31, 40, 45, 57, 66, 62, 63, 64, 87, 99

Time: $O(n^2)$

4. Merge Sort: (0) (11)

→ Base: 40, 63, 64, 2, 87, 62, 45, 66, 99, 30, 31, 57

(0+11/2 = 5)

[40,63,64,2,87,62] [45,66,99,30,31,57]

(0+5/2 = 2) (6+11/2 = 8)

[40,63,64] [2,87,62] [45,66,99] [30,31,57]

(0+3/2 = 1) (3+5/2 = 4) (6+8/2 = 7) (9+11/2 = 10)

[40,63] [64] [2,87] [62] [45,66] [99] [30,31] [57]

[40] [63] [2] [87] [45] [66] [30] [31]

dividing the list - - - - - - - comparisons + putting together.

[40,63] [40,63,64] [2,87] [2,62,87] [45,66] [45,66,99] [30,31] [30,31,57]

[40,63,64] and [2,62,87]

we assume both sorted so to get the next sorted merged list

[40 63 64]
[2 62 87]

[2 40 62 63 64 87]
2vs40 40vs62 62vs63 63vs87 64vs87

How merge sort compares sorted list from start and onwards.

[2, 40, 62, 63, 64, 87] ← [2,62,87] [40,63,64]

process: [2,40,62,63,64,87]
[30,31,45,57,66,99]

sorted. by recursive definition, we know these 2 already sorted. Then all have to do is COMPARE SMALLEST NEXT 2 INT from either list.

[2,30,...,64,66] (66 vs. 87)

[2,30,...,66,87,99] (87 vs. 99)

sorted final:

[2,30,31,40,45,57,62,63,64,66,87,99]

[2] (2 vs. 30)
[2, 30] (30 vs. 40) ← next smallest
[2,30,31] (31 vs. 40)
[2,30,31,40] (40 vs. 45)
[2,30,31,40,45] (45 vs. 62)
[2,30,31,40,45,57] (62 vs. 57)
[2,30,31,40,45,57,62] (62 vs. 66
[2,30,31,40,45,57,62,63] (63 vs.
[2,30,31,40,45,57,62,63,64] (64 vs 66)

Time: In each $K_1 \log n$ groups, make $K_2 n$ comparisons.

So:

$O(K_1 K_2 n \log n)$

$= O(n \log n)$

5. Quicksort.

→ Base: 13, 17, 0, 43, 28, 86, 83, 50, 3, 29, 36, 40 (pivot)

→ 1: 0, 3, 13, 17, 36, 28, 29, 40, 43, 83, 86, 50

→ 2: 0, 3, 13, 36, 28, 29, 17, 40, 43, 50, 83, 86

→ 0, 3, 13, 17, 28, 29, 36, 40, 43, 50, 83, 86

→ 0, 3, 13, 17, 28, 29, 36, 40, 43, 50, 83, 86

→ 0, 3, 13, 17, 28, 29, 36, 40, 43, 50, 83, 86

⇒ every list size 1 ... sorted.

The following pages contain complex math, akin to a math paper.

Skip it if you'd like.

If you're interested, read!

Time complexity: $\sim O(n \log n)$ on average.

Source Concrete Mathematics, Graham Knuth [28]:

quicksort satisfies: $C_0 = 0$

$$C_n = n + 1 + \frac{2}{n} \sum_{k=0}^{n-1} C_k, \quad n > 0$$

for the number of comparisons that quicksort makes on average.

(The first moment, or average gives: $EX = \left[\frac{d}{dt} P_n(t) \right]_{t=1}$.

where $P_n(t)$ describes the probability generating function of the # of comparisons quicksort can make).

we can solve this recurrence:

$$C_0 = 0, \quad C_n = n + 1 + \frac{2}{n} \sum_{k=0}^{n-1} C_k, \quad n > 0$$

$$C_n = n + 1 + [n>0] \sum_{k=0}^{n-1} C_k - [n=0], \quad n \geq 0.$$

(This describes our recurrence for $n \geq 0$ now.

$[\,]$ is the famous Iverson notation
... returns 1 if true, return 0 if false).

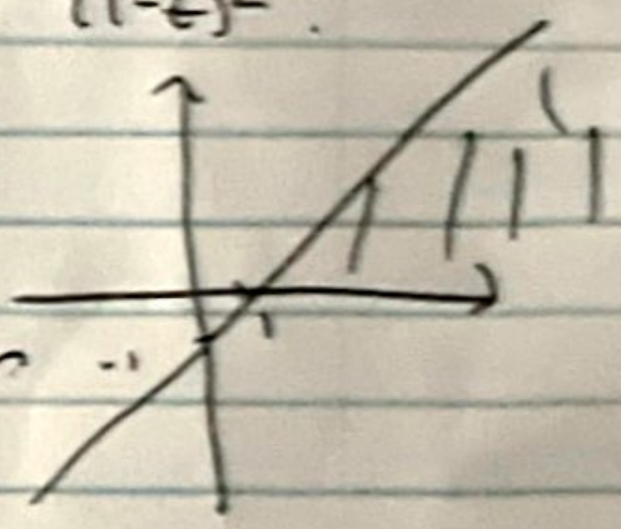
$$\Rightarrow C(z) = \sum_{n\geq 0} C_n z^n = \sum_{n\geq 0} n z^n + \sum_{n\geq 0} z^n$$

$$+ \sum_{n>0} z^n \left(\sum_{k=0}^{n-1} C_k \right) z^n - \sum_{n\geq 0} [n=0] z^n$$

$$\Rightarrow \sum_{n\geq 0} z^n = \frac{1}{1-z}, \quad \sum_{n\geq 0} n z^{n-1} = \frac{d}{dz}\left(\frac{1}{1-z}\right) = \frac{1}{(1-z)^2}$$

$$\Rightarrow \sum_{n\geq 0} n z^n = \frac{z}{(1-z)^2}.$$

$$\sum_{n>0} z^n \left(\sum_{k=0}^{n-1} C_k \right) z^n = \sum_{n>0,\, 0\leq k\leq n-1} z^n C_k z^n \Rightarrow$$



$$= \sum_{k\geq 0} C_k z^k \sum_{n\geq k+1} z^{n-k} \qquad n \geq k+1, \; k \geq 0.$$

ouch. No way forward.

Try again, using more sophisticated techniques.

note: if $G(z) = \sum_{n\geq 0} g_n z^n$, then $\int_0^z G(t)\,dt$

$$= \sum_{n\geq 0} g_n \cdot \frac{z^{n+1}}{n+1} = \sum_{n\geq 1} g_{n-1} \frac{z^n}{n}.$$

aha! $\sum_{n\geq 1} z^n \left(\underbrace{\sum_{k=0}^{n-1} C_k}_{S_{n-1}} \right) z^n = \sum_{n\geq 1} z^n S_{n-1} z^n.$

Then it suffices to find an $S(z) = \sum_{n\geq 0} S_n z^n$.

since $S_{n-1} = \sum_{k=0}^{n-1} c_k$, $S_n = \sum_{k=0}^{n} c_k$.

then $S(z) = \sum_{n\geq 0} \left(\sum_{k=0}^{n} c_k \right) z^n$

$= \left(\sum_{k_1 \geq 0} c_{k_1} z^{k_1} \right) \left(\sum_{k_2 \geq 0} z^{k_2} \right) = \sum_{n} \left(\sum_{k_1+k_2=n} c_{k_1} \right) z^n$

$= \sum_{n} \left(\sum_{k=0}^{n} c_k \right) z^n = S(z)$ ... wow!

so $\frac{1}{1-z} \cdot C(z) = S(z)$.

Therefore, $\sum_{n\geq 1} \frac{1}{n} s_{n-1} z^n = \int_0^z S(t)\, dt$

$= \int_0^z \frac{C(t)}{1-t}\, dt$ $\rightsquigarrow$ $\sum_{n\geq 1} \frac{u}{z} s_{n-1} z^n = 2\int_0^z \frac{C(t)}{1-t} dt$

Thus: $C(z) = \frac{1}{1-z} + \frac{z}{(1-z)^2} + 2\int_0^z \frac{C(t)}{1-t} dt - 1$

$\Rightarrow \frac{dP}{dz}(C(z)) = \frac{dP}{dz}\left(\frac{1}{1-z} + \frac{z}{(1-z)^2}\right) + \frac{2C(z)}{1-z}$

$C'(z) = \frac{z}{(1-z)^3} + \frac{2C(z)}{1-z}$

$\rightarrow \frac{dC}{dz} = \frac{z}{(1-z)^3} + \frac{2C}{1-z}$. This is a linear differential equation (next)

That yields: $C(z) =$

where $C(z) = \sum_{n\geq 0} c_n z^n$. $C(0) = c_0 0^0 + 0 + 0 + \ldots$

$= c_0 = 0$

$\Rightarrow \frac{z}{(1-0)^2} + \frac{k_1}{1} = 0 \Rightarrow k_1 = -2$.

This diff. equation yields (not gonna solve by hand, it's linear ... any good "memorizer" student could solve this)

$$\Rightarrow \quad C(z) = \frac{-2\ln(z-1)}{(1-z)^2} + \frac{K_1}{(1-z)^2} =$$

(Now, $C(0) = 0 = \sum_{n\geq 0} c_n (0)^n = c_0 0^0 + 0 = c_0 = 0$.)

we're going to now need some complex analysis.

$$\ln(z-1) = \ln(-(1-z)) = \underbrace{\ln(-1)}_{i\pi} + \ln(1-z)$$

$$\Rightarrow \quad \ln(z-1) = \pi i + \ln(1-z)$$

Thus: $C(z) = \frac{-2\pi i}{(1-z)^2} - \frac{2\ln(1-z)}{(1-z)^2} + \frac{K_1}{(1-z)^2}$

Do you see it now?

Since $C(0) = 0 \Rightarrow \quad 0 = \frac{-2\pi i}{1} - 0 + \frac{K_1}{1}$

Thus $K_1 = 2\pi i$

and $C(z) = \frac{-2\ln(1-z)}{(1-z)^2}$.

$$\Rightarrow \quad C(z) = \frac{2}{(1-z)^2} \cdot \ln\left(\frac{1}{1-z}\right).$$

Recall Knuth [352] concrete math: $\frac{1}{(1-z)^{m+1}} \ln\left(\frac{1}{1-z}\right)$

$$= \sum_{n\geq 0} (H_{m+n} - H_n) \binom{m+n}{n} z^n.$$

where H_n are the harmonic #'s up to $\frac{1}{n}$.

Ans: $C(z) = 2 \cdot \sum_{n \geq 0} (H_{1+n} - H_1) \binom{n+1}{n} z^n$.

so because $C(z) = \sum_{n \geq 0} c_n z^n$

we have ... $c_n = 2(H_{n+1} - H_1)\binom{n+1}{n}$

$= 2(n+1)(H_{n+1} - 1)$

$= 2(n+1)\left(\frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n+1}\right)$

$= 2(n+1)\left(1 + \frac{1}{2} + \cdots + \frac{1}{n} + \frac{1}{n+1} - 1\right)$

$= 2(n+1)H_n + 2 - 2n - 2$

$= \boxed{2(n+1)H_n - 2n}$ on the average.

and because $H_n \sim \log n$ by analytical techniques quicksort is $O(n \log n) = O(2(n+1)H_n - 2n)$.
average

remark: because $EX = 2(n+1)H_n - 2n$,
we can actually set up a diff. equation
to find the pgf of quicksort.
This is already discovered (probably)
in some rigorous mathematical paper
... or maybe I'll write one.

references: Concrete Matematics : Knuth, Graham
[335], [29], [28].

6. 40, 63, 64, 2, 87, 62, 45, 66, 99, 30, 31, 57

search for 62: using linear search:

40 ⇒ 40 ≠ 62

63 ⇒ 63 ≠ 62

64 ⇒ 64 ≠ 62

2 ⇒ 2 ≠ 62

87 ⇒ 87 ≠ 62

62 ⇒ 62 = 62 → end.

Time: O(n).

7. Binary search for 99:

02, 30, 31, 40, 45, 57, 62, 63, 64, 66, 87, 99
0 1 2 3 4 5 6 7 8 9 10 11

⇒ $\frac{0+11}{2} = 5$ ⇒ search: 57 ≠ 99, 57 < 99
... search upper half.

⇒ 62, 63, 64, 66, 87, 99
6 7 8 9 10 11

⇒ $\frac{6+11}{2} = 8$ ... search: 64 ≠ 99, 64 < 99
... search upper half

⇒ 66, 87, 99
9 10 11
... search: 87 ≠ 99, 87 < 99

$\frac{9+11}{2} = 10$ ... search upper half

⇒ 99
11
$\frac{11+11}{2} = 11$ ... search: 99 = 99
... end ✓.

8. search for 52:

02, 30, 31, 41, 45, 57, 62, 63, 64, 66, 87, 99
0 1 2 3 4 5 6 7 8 9 10 11

$\frac{0+11}{2} = 5$ ⇒ search: 57 ≠ 52 . 57 > 52
... search lower half

02, 30, 31, 41, 45
0 1 2 3 4

$\frac{0+4}{2} = 2$ ⇒ search: 31 ≠ 52 . 31 < 52
... search upper half

41, 45
3 4

⇒ search: 41 ≠ 52. 41 < 52
... search upper half

$\frac{3+4}{2} = 3$

45 → search: 45 ≠ 52. 45 < 52
4 ... search upper

$\frac{4+4}{2} = 4$

↗ (5)
⇒ lower is now larger than high → (4).
Nothing found.
return −1.

9. binary search run time
⇒ n elements → number of steps to reduce search space to 1 element
⇒ given by $\log_2(n)$

So we have $O(\log_2(n))$.