Link to github repo: https://github.com/lukerlars/sortingalgorithms/

Task 1 counting the steps

Implement the Four algorithms listed in Table 1 in python using Jupyter notebook or design your own python library. Then vary the size of the input and record the number of steps. Plot the
number of steps as a function the input size (n) to confirm that the plotted functions match the
asymptotic running time shown on the Table 1.
Hint: see the demo from the lecture

Algorithm	Worst-case running time	Average-case/expected running time
Insertion sort	$\Theta(n^2)$	$O(n^2)$
Merge sort	$\Theta(n \log n)$	$\Theta(n \log n)$
Heapsort	$O(n \log n)$	-
Quicksort	$\Theta(n^2)$	$\Theta(n \log n)$ (expected)

Table 1: Comparison of 4 sorting algorithms

Answer:

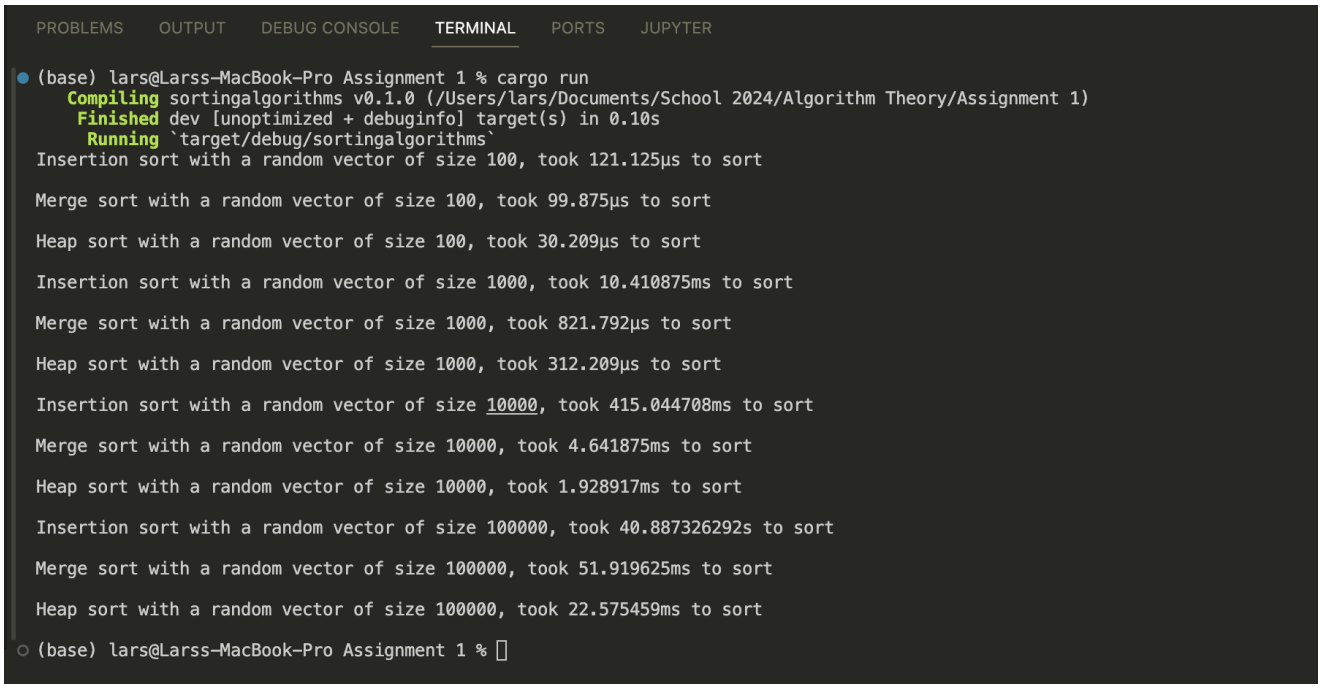
See jupyter notebook in repo.
Found that the Insertion sort significantly increases in time with input size.
Heapsort also start growing, not as quick as insert, but more than heap and merge, signifying its properties as a worst case $\Theta(n^2)$ and average case $\Theta(n \log(n))$

Task 2

Choose one or several of the listed algorithm and implement them in two different programming
languages. For example python and C or Go or C# or whatever language that you like. Run a test
by varying the input size and measuring the execution time on your computer. Comment the differences.

Answer

See rust code in repo.
Implemented insertion, heap and merge sort in Rust.
Run tests creating a random integer vector of test size 100, 1 000, 10 000, and 100 000



(originally, i had an error in heap sort, calling "create_max_heap" instead of heapify in the heapsport algorithm. Needless to say, this was quite a bit slower)
Results:

For the given implementations, we can see that for a small vector, Heapsort wins out with $32\mu s$
This is also the case for the 100 000 random integer sorts with a $22ms$ runtime.
For the small array insertion sort performed the worst with $121\mu s$ while merge sort was in-between. At the larger scale we can see that insertion sort begins to beat out heap sort.
This is unexpected as insertion sort is supposed to have $\Theta(n^2)$ running time over merge sort which has $\Theta(nlogn)$ as expected running time.

Task 3

3.1 Show that for any real constants a and b, where $b > 0$, $(n + a)^b = \Theta(n^b)$

Recall the definition for $\Theta(g(x))$:

$$\Theta(g(x)) = \{f(x) | \text{there exists constants } c_1, c_2 \text{ and } x_0 \text{ s.t } 0 \leq c_1 g(x) \leq f(x) \leq c_2 g(x) \text{ for all } x \geq x_0\}$$

For general polynomials: $p^n(x) = ax^n + bx^{n-1} + \ldots + c$
We need to show that $p^n(x) = \Theta(x^n)$
To check the upper bound, say that we have:

$$\sum_{k=0}^{n} c_k x^k \leq \alpha x^n$$

Then, we check for $n + 1$:

$$\sum_{k=0}^{n+1} c_k x^k = \sum_{k=0}^{n} c_k x^k + c_{n+1} x^{n+1}$$

We must then verify:

$$\sum_{k=0}^{n} c_k x^k + c_{n+1} x^{n+1} \leq \alpha x^{n+1}$$

Since

$$\sum_{k=0}^{n} c_k x^k \leq \alpha x^n$$

We can set

$$\alpha x^n + c_{n+1} x^{n+1} \leq \alpha x^{n+1}$$

$$c_{n+1} x^{n+1} \leq \alpha x^{n+1} - \alpha x^n = \alpha x^n (x-1)$$

$$\alpha \geq c_{n+1} \frac{x}{x-1}$$

For large enough values $\frac{x}{x-1} \approx 1$

For some $n_0, \text{ and } \alpha \geq c_{n+1}$
We can always choose an α such that the upper bound holds. Likewise one can also vary c_{n+1} to that the lower bound holds.
Further since:

$$f(x) = x^b + c_1 x^{b-1} + c_2 x^{b-1} + \ldots + c_b = \Theta(x^b)$$

Apply binomial theorem:

$$(n+a)^b = \sum_{k=0}^{b} \binom{b}{k} n^{b-k} a^k$$

$$= n^b + c_1 \cdot n^{b-1} + c_2 \cdot n^{b-2} + \ldots + c_b$$

Here we use: $c_k = \binom{b}{k} a^k$
Then we have

$$\rightarrow (n+a)^b = \Theta(n^b)$$

3.2) Show that $\frac{n^2}{lg(n)} = o(n^2)$

Recall the limit definition of $o(n)$, is the set of functions $f(n)$ satisfying:

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

Hence we have the limit

$$\lim_{n\to\infty} \frac{\frac{n^2}{lg(n)}}{n^2} = \lim_{n\to\infty} \frac{1}{lg(n)} = 0$$

3.3) Show that $n^2 \neq o(n^2)$

By the same logic:

$$\lim_{n\to\infty} \frac{n^2}{n^2} = 1 \neq 0$$

Task 4 Divide and Conquer Analysis

Using the Master theorem and the recursion tree method, find the running time (Big-O). Time taken
for the Karatsuba's multiplication algorithm is given by the following recurrence equation:
$T(n) = 3T(n/2) + \Theta(n)$

Answer

Given

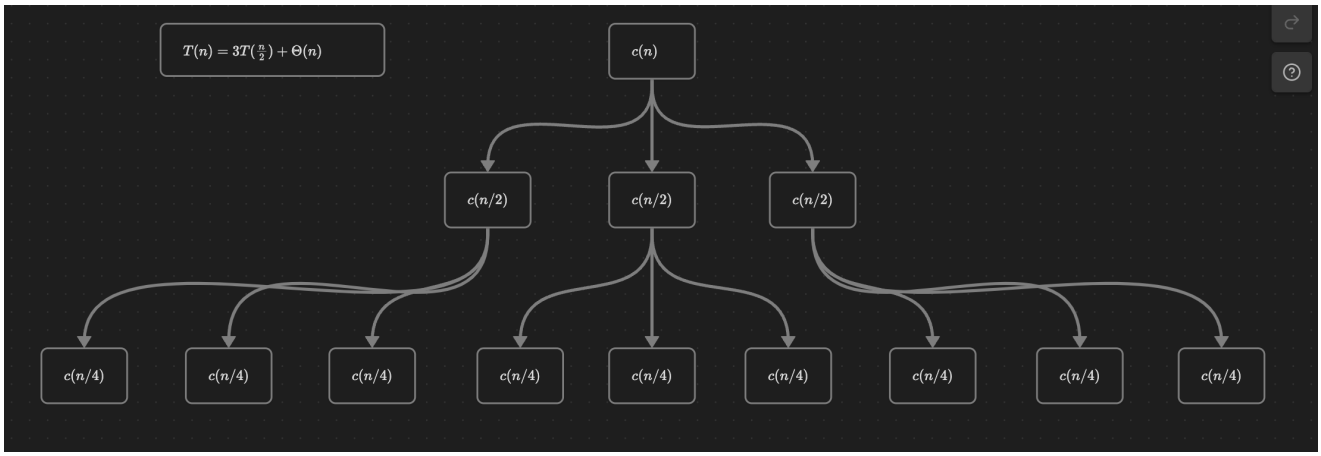
$$T(n) = 3T(\frac{n}{2}) + \Theta(n)$$

In this case for the master theorem, we have that $f(n) = \Theta(n)$
Recall case 1.
if. $f(n) = O(n^{log_b a - \epsilon})$ for some $\epsilon > 0$ then $T(n) = \Theta(n^{log_b a} lg(n))$

Since $f(n) = \Theta(n) \rightarrow f(n) = O(n)$
We then have that $f(n) = O(n^{log_2 3 - 1}) = O(n)$

Hence, $T(n) = \Theta(n^{lg(3)} lg(n))$

Recursion tree method:



Analysis:
There are $lg(n)$ levels

We begin by taking the implied function $\Theta(n) = cn$
The time cost for each level k is then $3^k c(\frac{n}{2^k})$

Summing up for each level:

$$T(n) = \sum_{k=0}^{lg(n)} 3^k \frac{cn}{2^k}$$

The root has cost cn
Level 1 has cost $3c(\frac{n}{2})$
level 2 has cost $9c(\frac{n}{4})$
For each layer, we have cost $3^{lg(k)} = k^{lg(3)}$
And we have $lg(n)$ layers.
Hence

$$T(n) = \Theta(n^{lg(3)} lg(n))$$