

Genome 540 Discussion

Jan. 8th 2026



Discussion Structure

20 – 30 mins discussion, 20 mins office hours

- Lecture questions, thoughts
- Homework advice
- Related technical topics
- Debugging, get unstuck

Homework 1

Lets Do An Example!

sequence1: CTAACGC

sequence2: TGTTAGGC

Homework 1

Handwritten notes on a whiteboard showing a sequence alignment problem. The top row displays the sequence $C T A A C G C$ (positions p0 to p6) and $T G T T A G G C$ (positions p7 to p14). The third row shows the sequence $G C C T A A C A$ (positions p15 to p22).

Below the sequences, three columns of handwritten notes show the alignment of the sequences:

- Column 1 (Left):
 - p22 A
 - p19 AACA
 - p2 AACGC
 - p20 ACA
 - p3 ACGC
 - p11 AGGC
 - p6 C
 - p14 C
 - p21 CA
- Column 2 (Middle):
 - p16 CCTAACA
 - p4 CGC
 - p17 CTAAACA
 - p0 CTAAACGC
 - p5 GC
 - p13 GC
 - p15 GCCTAACA
- Column 3 (Right):
 - p12 GGC
 - p8 GTTAGGC
 - p18 TAACA
 - p1 TAACGC
 - p10 TAGGC
 - p7 GTTAGGC
 - p9 TTAGGC

Python sticking points:

```
def suffix_array(seq1, seq2):
    rev_comp_seq2 = rev_comp(seq2)

    pointers = list(range(0, len(seq1) + 2 * len(seq2)))

    def pointer_to_sequence(pointer):
        s = ""
        if 0 <= pointer < len(seq1):
            s = seq1
        elif len(seq1) <= pointer < len(seq1) + len(seq2):
            s = seq2
            pointer -= len(seq1)
        elif len(seq1) + len(seq2) <= pointer < len(seq1) + 2 * len(seq2):
            s = rev_comp_seq2
            pointer -= len(seq1) + len(seq2)

        return s[pointer:]

    pointers.sort(key = pointer_to_sequence)
    # ....
```

Python sticking points:

In general, slicing a string ! makes a copy ! of that string.

Look up “zero-copy slicing”, which does exist, but only for certain kinds of python objects.

Try ``bytearray`` or ``memoryview``.

Rust sticking points:

Strings

The two most used string types in Rust are `String` and `&str`.

A `String` is stored as a vector of bytes (`Vec<u8>`), but guaranteed to always be a valid UTF-8 sequence. `String` is heap allocated, growable and not null terminated.

`&str` is a slice (`&[u8]`) that always points to a valid UTF-8 sequence, and can be used to view into a `String`, just like `&[T]` is a view into `Vec<T>`.

These tools are built to solve issues that arise due to text encoding, but nothing in the class will require such tooling, and some of the restrictions can get in the way.

I found myself using the `.as_bytes()` conversion a lot, might be better to commit early to working with arrays of `u8 (char)` s.

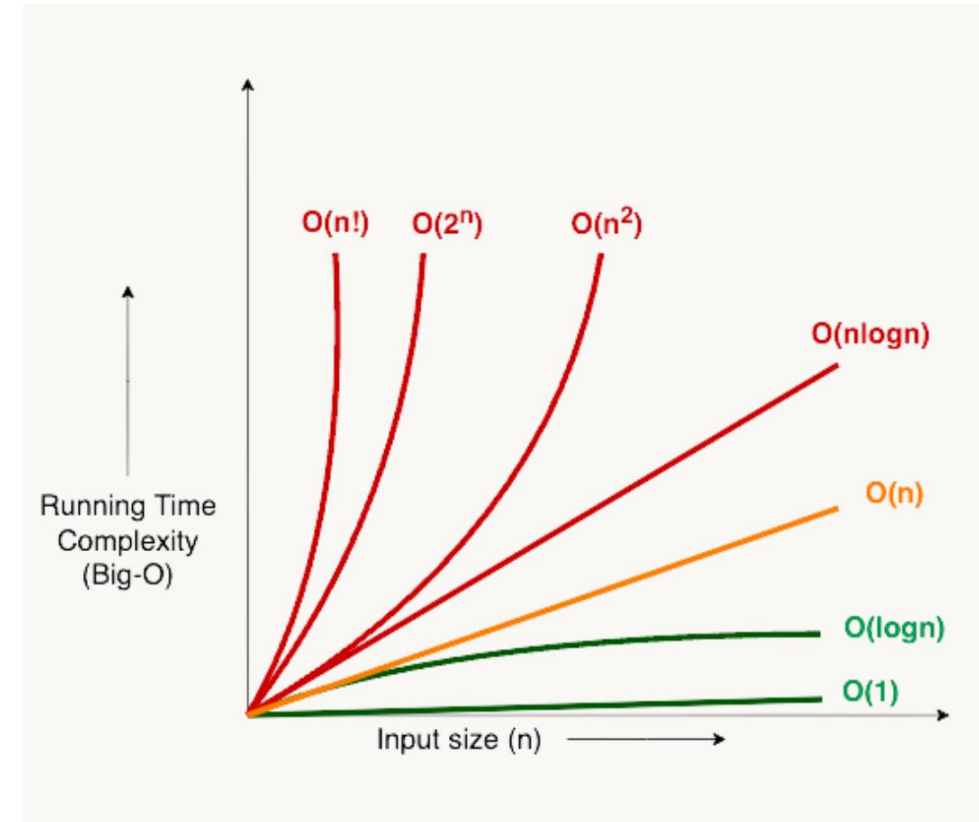
Algorithmic Complexity

$$f(N) = .05 N^3 + 50.7 N^2 + 6.03 N$$

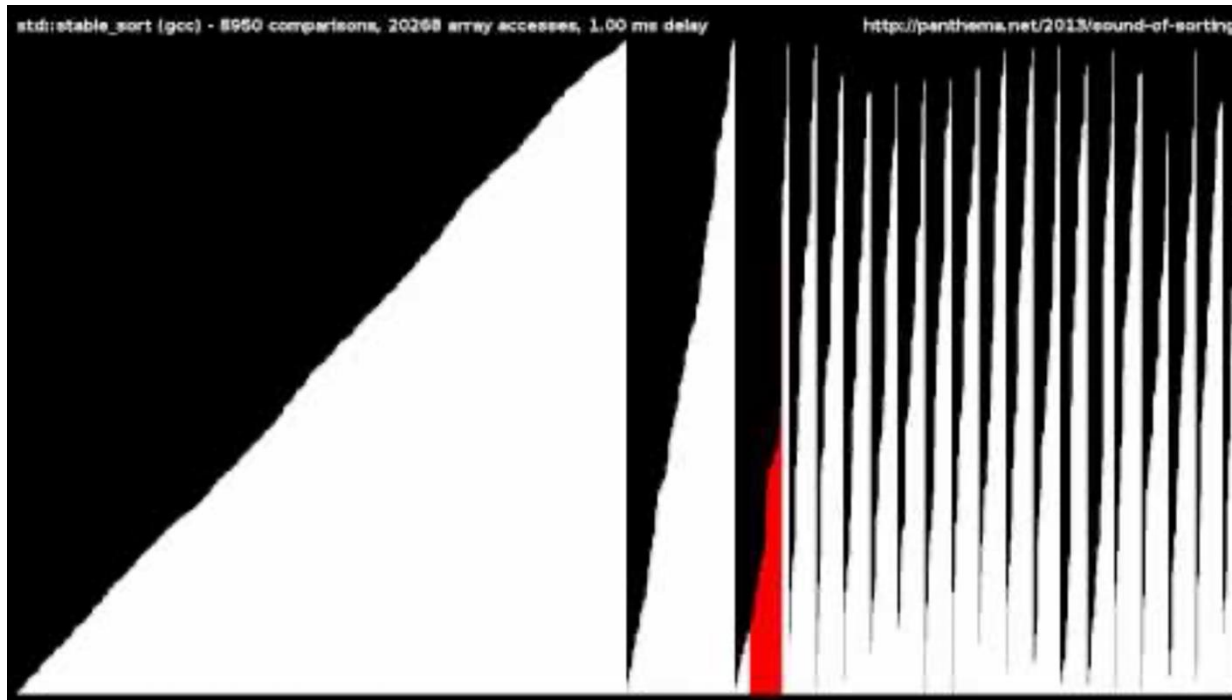
$$C_1 g(N) < f(N) < C_2 g(N)$$

$$\text{take } g(N) = N^3$$

$$O(N^3)$$



Algorithmic Complexity



<https://www.youtube.com/watch?v=kPRA0W1kECg>

Name	Best	Average	Worst	Memory	Stable	In-place	Method	Other notes
Heapsort	$n \log n$	$n \log n$	$n \log n$	1	No	Yes	Selection	An optimized version of selection sort. Performs selection sort by constructing and maintaining a max heap to find the maximum in $O(\log n)$ time.
Introsort	$n \log n$	$n \log n$	$n \log n$	$\log n$	No	Yes	Partitioning & Selection	Used in several STL implementations. Performs a combination of Quicksort, Heapsort, and Insertion sort.
Merge sort	$n \log n$	$n \log n$	$n \log n$	n	Yes	No	Merging	Highly parallelizable (up to $O(\log n)$ using the Three Hungarians' Algorithm). ^[5]
In-Place Merge Sort	$n \log^2 n$	$n \log^2 n$	$n \log^2 n$	$\log n$	Yes	Yes	Merging	Variation of Mergesort which uses an $O(n \log n)$ in-place stable merge algorithm, such as rotate merge or symmerge.
Tournament sort	$n \log n$	$n \log n$	$n \log n$	n	Yes	No	Selection	An optimization of Selection Sort, which uses a tournament tree to select the min/max.
Tree sort	$n \log n$	$n \log n$	$n \log n$ (balanced)	n	Yes	No	Insertion	When using a self-balancing binary search tree.
Block sort	n	$n \log n$	$n \log n$	1	Yes	Yes	Insertion & Merging	Combine a block-based $O(n)$ in-place merge algorithm ^[6] with a bottom-up merge sort.
Smoothsort	n	$n \log n$	$n \log n$	1	No	Yes	Selection	Adaptive variant of heapsort based on the Leonardo sequence instead of a binary heap.
Timsort	n	$n \log n$	$n \log n$	n	Yes	No	Insertion & Merging	Makes $n - 1$ comparisons when the data is already sorted or reverse sorted.
Patience sorting	n	$n \log n$	$n \log n$	n	No	No	Insertion & Selection	Finds all the longest increasing subsequences in $O(n \log n)$.
Cubesort	n	$n \log n$	$n \log n$	n	Yes	No	Insertion	Makes $n - 1$ comparisons when the data is already sorted or reverse sorted.
Quicksort	$n \log n$	$n \log n$	n^2	$\log n$	No	Yes	Partitioning	Quicksort can be done in-place with $O(\log n)$ stack space. ^{[7][8]}
Fluxsort	n	$n \log n$	$n \log n$	n	Yes	No	Partitioning & Merging	An adaptive branchless stable introsort.
Crumsort	n	$n \log n$	$n \log n$	$\log n$	No	Yes	Partitioning & Merging	An in-place, but unstable variant of Fluxsort.
Library sort	$n \log n$	$n \log n$	n^2	n	No	No	Insertion	Similar to a gapped insertion sort.

https://en.wikipedia.org/wiki/Sorting_algorithm

Algorithmic Complexity

Pseudocode [\[edit\]](#)

The following is a description of the randomized algorithm in [pseudocode](#):

```
function bogoSort(deck: List):  
    while deck is not sorted:  
        shuffle(deck)
```

Algorithmic Complexity

Algorithm 1 was found when the author was making up some wrong sorting algorithms and changed the j loop of Algorithm 2 to be from 1 to n , and was shocked to see that it sorts in decreasing order.

While the pseudocode of Algorithm 1 makes it look like an exchange-based sort, in fact the first iteration of the outer loop ($i = 1$) selects the maximum, and then each of the other iterations works like insertion sort. Thus in some way it is a combination of selection-based and insertion-based algorithms.

Algorithm 1 ICan'tBelieveItCanSort($A[1..n]$)

```
for  $i = 1$  to  $n$  do
  for  $j = 1$  to  $n$  do
    if  $A[i] < A[j]$  then
      swap  $A[i]$  and  $A[j]$ 
```

Algorithm 2 ExchangeSort($A[1..n]$)

```
for  $i = 1$  to  $n$  do
  for  $j = i + 1$  to  $n$  do
    if  $A[i] > A[j]$  then
      swap  $A[i]$  and  $A[j]$ 
```
