CS2040S

Recitation 2
AY20/21S2

Question 1

Problem

Input: Given a stack of pancakes with varying sizes

Output: Order it with the smallest on top and the biggest at the bottom

Constraint: You can only flip pancakes from the top

Problem 1.a.

Given a stack of n pancakes, how many flips (in terms of n) does it take to order it?

Guiding question

How many flips does it take to get a specific pancake to the bottom of the stack?

Guiding question

How many flips does it take to get a specific pancake to the bottom of the stack?

Answer: 2:

1. Flip from target pancake to bring it to the top
2. Flip entire stack to bring pancake to the bottom

Guiding question

What is the base case? At worst, how many flips are required to sort it?

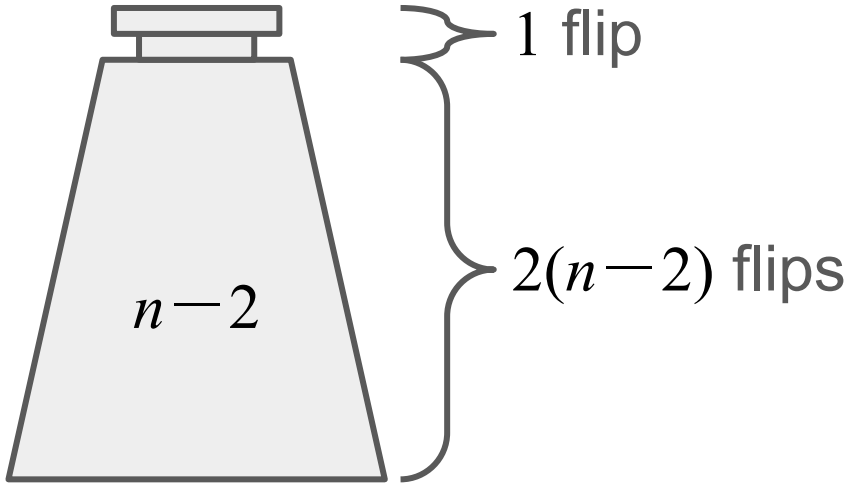
Guiding question

What is the base case? At worst, how many flips are required to sort it?

Answer: 2 pancakes. 1 flip need at worst.

Realise that base cases in pancake problems are special cases that require *less* flips than the usual cases and therefore require special treatment.

Flips needed



Problem 1.b.

From your proposed pancake flipping strategy in the previous part, invariant at each step of the sorting process?

Out of all the sorting algorithms covered in lectures so far, which one of them is the most analogous to your strategy?

Guiding question

What can you say about the problem after $2k$ flips according to our strategy?

Guiding question

What can you say about the problem after $2k$ flips according to our strategy?

Answer: After $2k$ flips, k pancakes are sorted at the bottom of the stack.

Guiding question

Which sorting algorithm(s) most closely resembles this invariance?

Guiding question

Which sorting algorithm(s) most closely resembles this invariance?

Answer: Max-selection sort or bubble-sort. Both these entail a sorted region at the end that grows by 1 with each completion of their subroutine.

Problem 1.c.

Now each pancake has a burnt side

You want to order it also with the burnt side facing down

How many flips do you need now?

Guiding question

Do we need a new strategy or can we built on the previous one? What has changed?

Guiding question

Do we need a new strategy or can we built on the previous one? What has changed?

Answer: We can build on the previous strategy. All we need is one extra step in the worst case: after flipping target package to the top, we need to flip it once before we flip the entire unsorted stack. This is to orientate it such that the burnt side is facing down in its final state.

Guiding question

What is the base case now? How many flips at worst does it need?

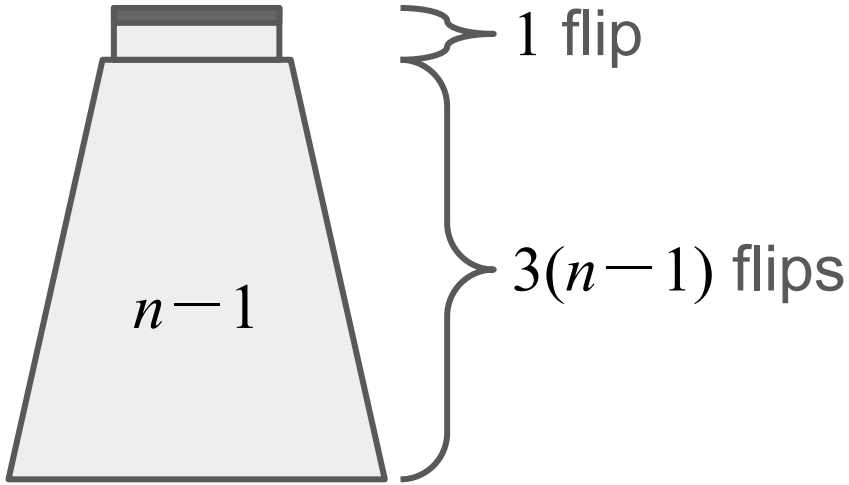
Guiding question

What is the base case now? How many flips at worst does it need?

Answer: Now the base case is just a single pancake which requires 1 flip to orientate its burnt side down.

The second last pancake cannot be ordered in less than 3 flips so it falls under the regular case.

Flips needed



Problem 1.d.

What if there are only 2 sizes of pancakes?

How many flips do you need now?

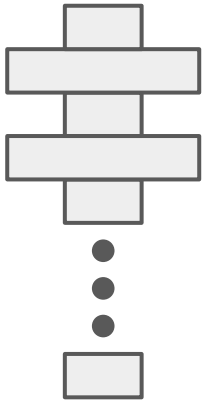
Guiding question

What is a possible worst case if there are only 2 sizes of pancakes?

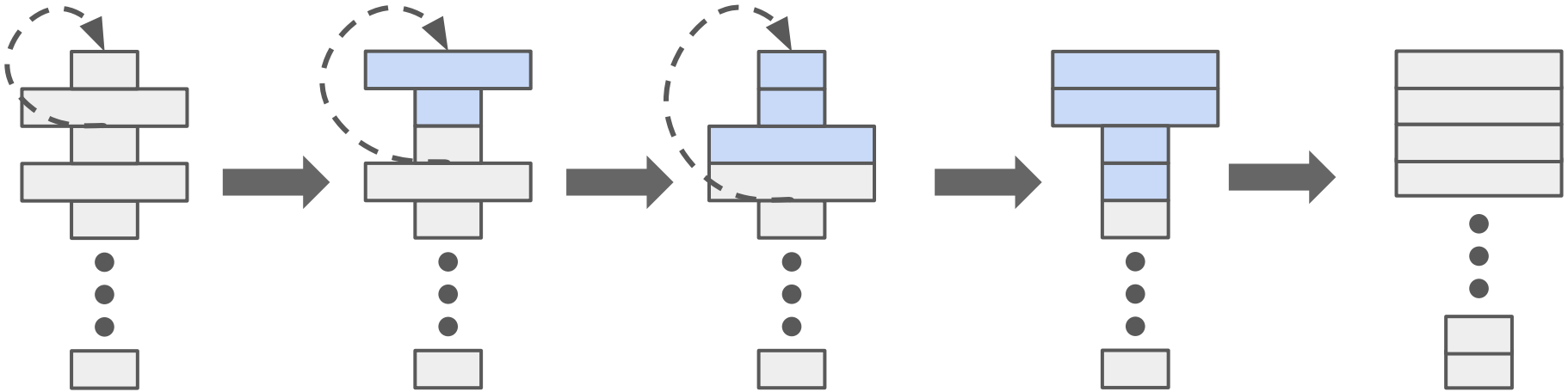
Guiding question

What is a possible worst case if there are only 2 sizes of pancakes?

Answer:



Solution



Realise that for the worst-case to happen, we would also need the smaller pancake to be the bottom-most so that one final flip would be needed in the end.

Flips needed

Starting from the top after the second pancake, we'll have to flip every time we encounter the next pancake (which is of a different size than the previous).

This means we'll have taken $n-2$ flips to reach the last pancake. However don't forget that since the last pancake is a small one, we'll need 1 final flip to orientate the entire stack. So total flips needed is $n-1$.

Find it hard to see? Do up a table and reason inductively.

Problem 1.e.

What if we want to order pancakes according to their skin textures instead of their sizes? I.e., the crunchiest (darkest in colour) on top and the fluffiest (lightest in colour) at the bottom?

Guiding question

Does this change the problem?

Guiding question

Does this change the problem?

Answer: No it doesn’t. Skin texture is just another abstraction of the measurement which we use to sort the pancakes. We simply need to relabel each pancake using their texture measurements.

Test yourself!

Do items in a collection need to be measurable in order for them to be sortable via comparison-based sorting?

Test yourself!

Do items in a collection need to be measurable in order for them to be sortable via comparison-based sorting?

Answer: No, they just need to be *mutually comparable*. E.g. I can sort a bag of apples and oranges if I say that oranges are *greater* than apples, without specifying any amounts attached to them (doesn't matter how much greater). Look at how we implement custom comparator classes in Java.

Problem 1.f.

Show that any pancake sorting algorithm requires at least n

Flips.

What do we call such properties of the problem?

Lower bound

Definition: Complexity of the *best possible* solution we can have on the *worst possible case* input for the problem.

In other words, a ‘theoretical limit’ for a problem by which no solutions can be faster than.

Guiding question

What is one possible worst case input?

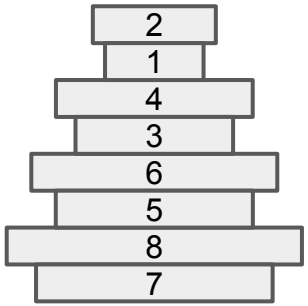
Hint: Consider adversarial inputs for sorting algorithms. The worst case is often ‘very close’ to the best case with a single property that is reversed/wrong.

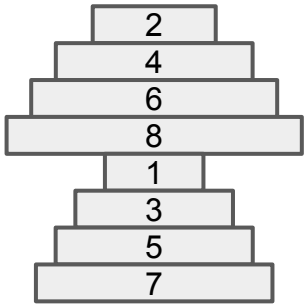
Guiding question

What is one possible worst case input?

Hint: Consider adversarial inputs for sorting algorithms. The worst case is often ‘very close’ to the best case with a single property that is reversed/wrong.

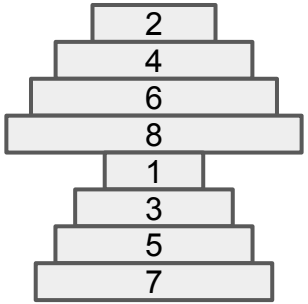
Answer:





And possibly many more forms..

Worst cases



For such worst case problems, we would *minimally* have to separate *every* consecutive pancake by inserting our spatula in between and them and flipping.

Guiding question

How many of such “inbetweens” do we have to flip in a worst case?

Guiding question

How many of such “inbetweens” do we have to flip in a worst case?

Answer: Exactly n. Don’t forget the “inbetween” between the the bottom-most pancake and the table! Trick: you can treat the table as a pancake of infinite size.

Question 2

Problem

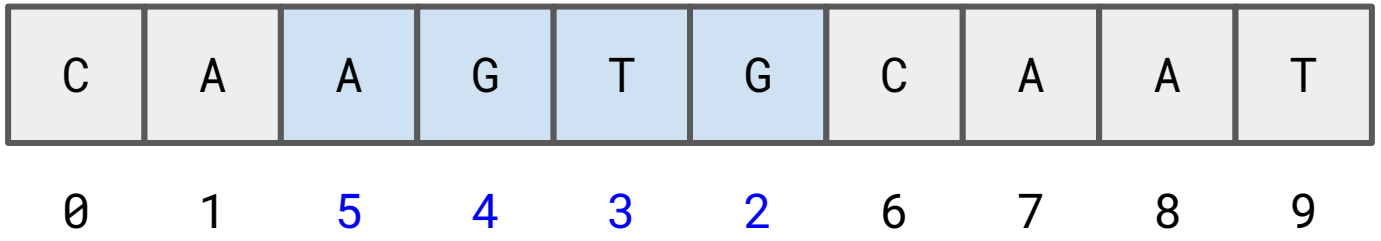
Input: String (sequence of characters) e.g. DNA sequence

Output: Ordered string

Constraint: Can only reverse a subsequence

Example

After reversing [2,5]



Problem 2.a.

First assume your string is binary: only made up of letters ‘A’ and ‘T’.

Devise a divide-and-conquer algorithm for sorting. What is the recurrence? What is the running time?

Caveat: The only legal operations are reversals. You cannot simply count the ‘A’s and ‘T’s and rebuild the string! Inspecting/examining the string is free.

Guiding question

What are the divide and conquer algorithms we have seen so far and which one of them is appropriate?

Guiding question

What are the divide and conquer algorithms we have seen so far and which one of them is appropriate?

Answer: We have seen Merge Sort and Quick Sort. Since we cannot easily implement Quick Sort with the operation constraint of subsequence flipping, Merge Sort is probably a better choice.

Guiding question

What do we achieve by sorting a sequence of only 2 element types?

Guiding question

What do we achieve by sorting a sequence of only 2 element types?

Answer: We achieve a *binary partitioning* where each partition is a sequence of homogeneous type.

Guiding question

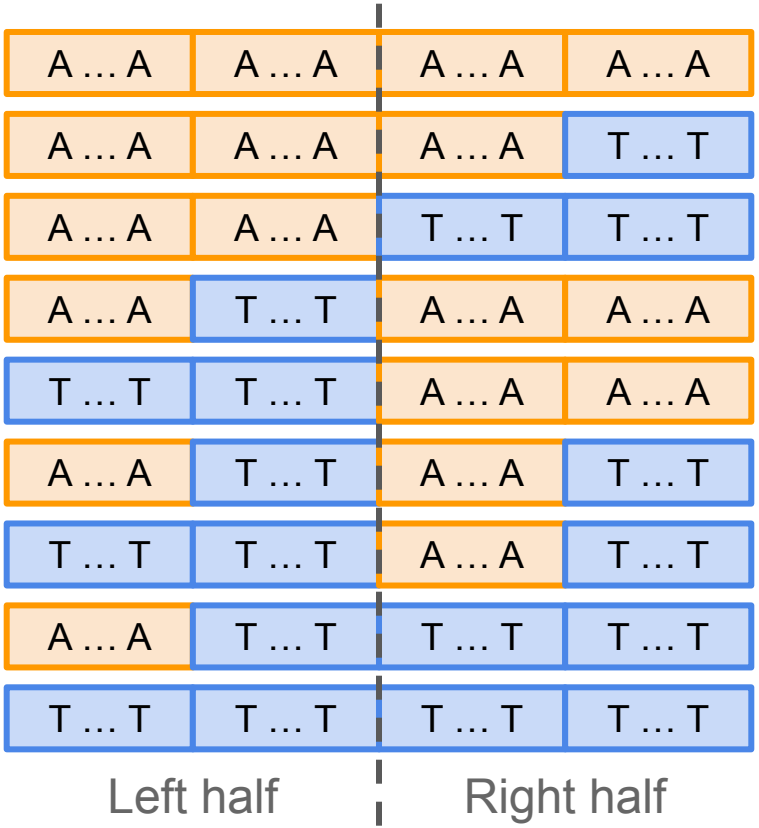
After dividing the string into two halves and sorting each half, what do you have?

Guiding question

After dividing the string into two halves and sorting each half, what do you have?

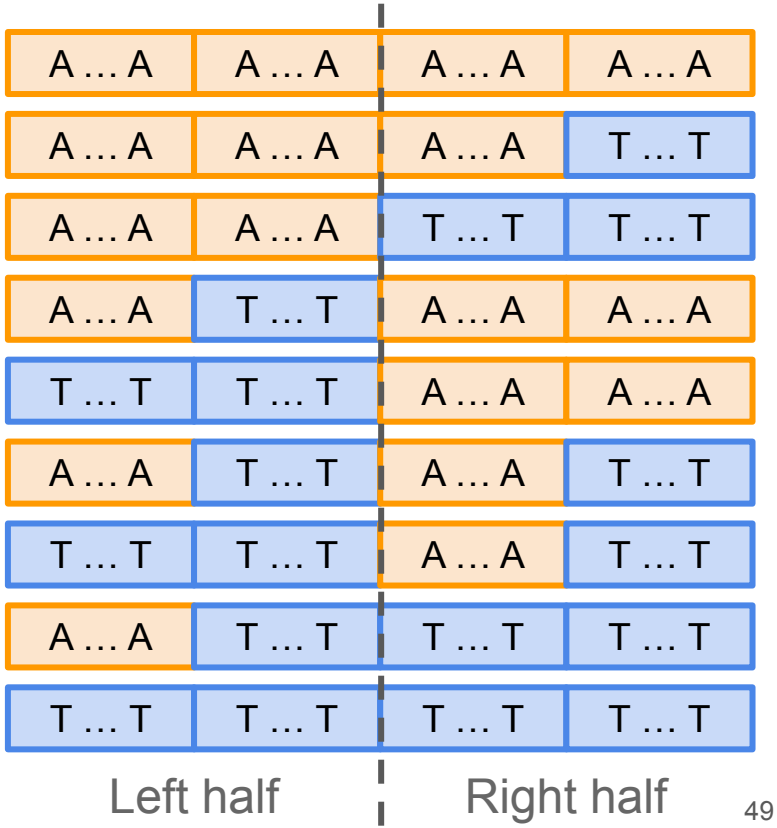
Answer: After sorting, each half will entail at most 2 homogeneous partitions each. We would end up with 9 possible distinct partition sequences (see next slide). Note that although the total number of possible combinations for 4 homogenous partitions is $2^4 = 16$, not all all these are valid since each half must be itself sorted.

9 distinct sequences possible



Guiding question

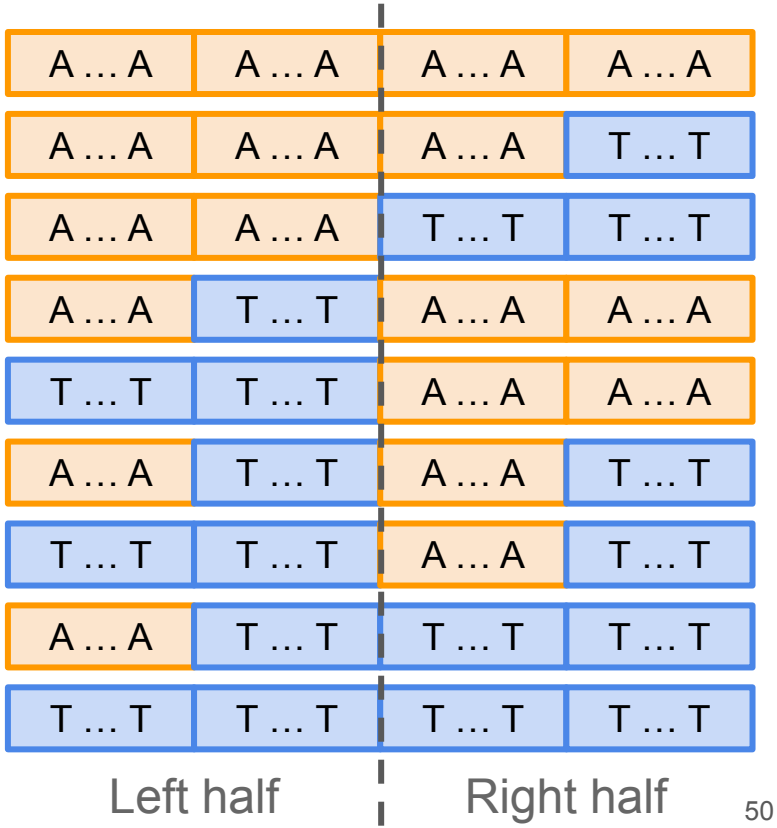
Given these cases, how do we implement the modified merge step? Can you come up with a generalisable algorithm to do so? What is its complexity?



Guiding question

Given these cases, how do we implement the modified merge step? Can you come up with a generalisable algorithm to do so? What is its complexity?

Answer: Flip [first T in left half, last A in right half]. $O(n)$ time.



Guiding question

What is the overall time complexity of our binary partitioning algorithm using our modified Merge Sort?

Guiding question

What is the overall time complexity of our binary partitioning algorithm using our modified Merge Sort?

Answer: Since we only modified the merge step and its complexity is still $\mathrm{O}(n)$. The overall complexity of our modified Merge Sort is still $\mathrm{O}(n \log n)$.

Cool ideas

Some of you have also pointed out that we can directly reuse the solution from our “two-size pancake flipping” problem. Indeed! :)

The only caveat is that it would make it harder to analyze since we have to reconcile and decide which inspections are free and how much should a flip cost in this problem. I’ll leave that to you.

Problem 2.b.

Now, assume your string consists of arbitrary characters. Devise a QuickSort-like algorithm for sorting.

Hint: use the binary algorithm above to help you implement partition

Assume for today that each element in the string is unique, i.e., there are no duplicates.

What is the recurrence? What is the running time?

QuickSort overview

1. Pick a random pivot
2. Partition based on pivot
3. Recursive step:
    - QuickSort on left half
    - QuickSort on right half

Guiding question

Which step in QuickSort can we implement using our 2.a. solution with minimal modifications?

Guiding question

Which step in QuickSort can we implement using our 2.a. solution with minimal modifications?

Answer: Step 2 (partitioning step). We can transform it into a binary partitioning problem by labeling each element (using a bit) based on whether it is greater or lesser than pivot. Thereafter, we can solve as per 2.a. solution

Guiding question

Assuming we pick a good pivot on average which partitions into two even halves, what is the time complexity $\mathrm{T}(n)$ of our solution expressed as a recurrence relation?

Guiding question

Assuming we pick a good pivot on average which partitions into two even halves, what is the time complexity $\mathrm{T}(n)$ of our solution expressed as a recurrence relation?

Answer: $\mathrm{T}(n) = 2\mathrm{T}(n/2) + \mathrm{O}(n \log n)$

Complexity analysis: loose but intuitive

Quicksort: $T(n) = 2T(n/2) + O(n) \quad = O(n \log n)$

Ours: $T(n) = 2T(n/2) + O(n \log n)$

Since there is a multiple of $\log n$ at every level, the overall time complexity should also be a factor of Quick Sort's time complexity by a multiple of $\log n$, thus giving us $O(n \log^2 n)$.

Complexity analysis: rigourous

$$
\begin{aligned}
T(n) &= 2T(n/2) + O(n \log n) \\
&= 2[2T(n/4) + O(\frac{n}{2} \log \frac{n}{2})] + O(n \log n) && \text{Expand } T(n/2) \\
&= 4T(n/4) + O(n \log \frac{n}{2}) + O(n \log n) \\
&= O(n \log n + n \log \frac{n}{2} + n \log \frac{n}{4} + n \log \frac{n}{8} + \cdots + n \log \frac{n}{n}) && \text{Expanding from pattern} \\
&= O(n[\log \frac{n}{1} + \log \frac{n}{2} + \log \frac{n}{4} + \log \frac{n}{8} + \cdots + \log \frac{n}{n}]) && \text{Factorize out } n \\
&= O(n[\log n - \log 1 + \log n - \log 2 + \cdots + \log n - \log n]) && \text{Applying log law} \\
&= O(n[\sum_{1}^{\log n} \log n - (\log 1 + \log 2 + \log 4 + \log 8 + \cdots + \log n)]) && \text{Since sequence length is } \log n
\end{aligned}
$$

Continued next page..

Complexity analysis: rigourous (cont'd)

$$= O(n[\sum_{1}^{\log n} \log n - (0 + 1 + 2 + 3 + \cdots + \log n)])$$

$$= O(n[\sum_{1}^{\log n} \log n - \sum_{i=1}^{\log n} i])$$

$$= O(n[\log^2 n - (\log n)(\log n + 1)/2]) \qquad \text{Open up summations}$$

$$= O(n[\log^2 n - (\log^2 n + \log n)/2]) \qquad \text{Expand bracket}$$

$$= O(n[\log^2 n - \frac{1}{2}\log^2 n - \frac{1}{2}\log n])$$

$$= O(n[\frac{1}{2}\log^2 n - \frac{1}{2}\log n])$$

$$= O(n[\log^2 n - \log n]) \qquad \text{Constant factors ignored by big O}$$

$$= O(n\log^2 n - n\log n]) \qquad \text{Opening up bracket}$$

$$= O(n\log^2 n) \qquad \text{Taking big O}$$

Problem 2.c.

What if there are duplicate elements in the string?

Can you still use the exact routine from the previous part? If not, what would you now do differently?

Hint: think the most extreme case for duplicate elements.

Discuss your answers in the recitation forums!

Extra exercises

Can you fill up this time complexity table?

Input order →	Random	Sorted		Nearly Sorted		Homogeneous
Algorithm ↓		Ascending	Descending	Ascending	Descending	(i.e. identical elements)
(Opt) Bubble sort	$O(N^2)$	$O(N)$ - best				
(Min) Selection Sort					$O(N^2)$	
Insertion Sort			$O(N^2)$			
Merge Sort				$O(N \log N)$		
(Naive) Quick Sort*		$O(N^2)$				
(Rand) Quick Sort	$O(N \log N)$					

* Non-random Quick Sort which always chooses first element as pivot

There is able in stable

Are you able to easily modify and turn **any** non-stable comparison-based sorting algorithm into a stable one by just incurring an additional $O(n)$ space?