1 Big-O Notation

Prove each of the following using the definition of big-O notation (find constants $c$ and $n_0$ such that $f(n) \leq c \times g(n)$ for $n > n_0$).

- $3n^3 + 9n^2 + n + 1 = O(n^3)$
- $5n \log_2 n + 8n - 200 = O(n \log_2 n)$

2 More Big-O

Order the following by their growth rates from smallest to largest.

1. $O(n^{1.9})$
2. $O(n^3)$
3. $O(\log n)$
4. $O(n)$
5. $O(n^n)$
6. $O(\sqrt{n})$
7. $O(2^n)$
8. $O(n!)$
9. $O(n \log n)$
3 And some more...

Write each of the following in big-O notation. No proof necessary.

- \(2n + 3n \log n + 15n^{1.1} =\)
- \(3n + 5n(\log n)^{55} + n^3 =\)
- \(2n(3 + \log n + n^2) =\)
- \(2n^{55} + 2^n =\)
- \(2n(\log n)^4 + n^2 =\)

4 Merge sort

Slow-Merge-Sort is an algorithm that works the same as the merge-sort we looked at in class, but in the merge step it uses an algorithm called Slow-Merge that takes \(O(n^2)\) time.

- Write a recurrence for the runtime of Slow-Merge-Sort
- Solve the recurrence using the substitution method
- Express the runtime of Slow-Merge-Sort using Big-O notation

5 Recurrences

Solve the following recurrences. Assume \(T(n) \leq c\) for some constant \(c\) and for all \(n \leq 10\).

- \(T(n) = 2T(\frac{n}{4}) + n^{0.3}\)
- \(T(n) = 4T(\frac{n}{2}) + n^2 \sqrt{n}\)

6 Compare two lists

You are given two unsorted lists of integers, \(L_1\) and \(L_2\). There may be duplicates in the lists and the length of \(L_1\) and \(L_2\) are both \(n\). Now we want to test whether \(L_1\) and \(L_2\) contain exactly the same list of integers (with duplicates allowed). For example, let
$L_1 = \{3,1,1,5\}$ and $L_2 = \{1,5,3,1\}$. Then we say $L_1$ and $L_2$ contain the same list of integers. But $\{1,1,2,2\}$ and $\{1,2,2,2\}$ do not.

Now, write down an efficient algorithm and analyze the running time.

## 7 Partition

The following array has been partitioned. Which elements could have been the pivot value?

$[16, 25, 8, 40, 32, 42, 55, 67, 59, 73]$ 

## 8 Pancake flipping

A stack of $n$ pancakes is placed in front of you. You have a spatula which you can insert anywhere into the stack and flip over all the pancakes above the spatula. You want to arrange the pancakes in order of their diameter (they are perfectly round), and you want to use as few flips as possible. As an example suppose $n = 6$, and the pancakes are numbered 1 through 6 in order of their diameter with 1 the smallest and 6 the largest. Suppose the original order is 346215, and the left end of the sequence represents the top of the stack. In one flip I can get 643215 (by flipping the first three pancakes: 346), then in the next flip 512346, then 432156, then 123456, so four flips are enough in this case. Let $F(n)$ be the worst case number of flips needed to arrange a stack of $n$ pancakes. Find an efficient algorithm for this problem, where efficiency is measured by the worst case number of flips.

Remember that your algorithm should work for pancakes with any order. To start you off you should easily be able to show that $F(n)$ is at most $2n$. Next, reduce that bound a little more if you can.

## 9 Glass Jar Problem

Suppose you have 2 identical glass jars and you’re in a skyscraper with a 100 floor spiral stairwell. Your goal is to determine the highest floor from which you can drop a jar without it breaking. Describe an algorithm to do this with the fewest drops in the worst case. Repeat for a general building with $n$ floors.