# CS3000: Algorithms & Data — Spring 2019 — Paul Hand

Homework 2
Due Wednesday 1/30/2019 at 2:50pm via Gradescope

Name:
Collaborators:

- Make sure to put your name on the first page. If you are using the LaTeX template we provided, then you can make sure it appears by filling in the `yourname` command.
- This assignment is due Wednesday 1/30/2019 at 2:50pm via Gradescope. No late assignments will be accepted. Make sure to submit something before the deadline.
- Solutions must be typeset in LaTeX. If you need to draw any diagrams, you may draw them by hand as long as they are embedded in the PDF. I recommend using the source file for this assignment to get started.
- I encourage you to work with your classmates on the homework problems. *If you do collaborate, you must write all solutions by yourself, in your own words.* Do not submit anything you cannot explain. Please list all your collaborators in your solution for each problem by filling in the `yourcollaborators` command.
- Finding solutions to homework problems on the web, or by asking students not enrolled in the class is strictly forbidden. You may use the internet for general research and learning pertaining to class material.

**Problem 1.** *Asymptotic Order of Growth*

(a) Rank the following functions in increasing order of asymptotic growth rate. That is, find an ordering $f_1, f_2, \ldots, f_{10}$ of the functions so that $f_i = O(f_{i+1})$. No justification is required.

$$\begin{array}{ccccc} n^3 & 7^{\log_2 n} & n! & 12^n & \log_2(n!) \\ 2^{4n} & 100n^{3/2} & 10n & 2^{\log_3 n} & \log_2^3 n \end{array}$$

**Solution:**

(b) Suppose $f(n), g(n), h(n)$ are non-decreasing, non-negative functions and that $f(n) = O(h(n))$ and $g(n) = O(h(n))$. Prove that the $f(n)g(n) = O(h(n)^2)$.

**Solution:**

**Problem 2.** *Improve the MBTA*

You have been commissioned to design a new bus system that will run along Huntington Avenue. The bus system must provide service to $n$ stops on the eastbound route. Commuters may begin their trip at any stop $i$ and end at any other stop $j > i$. Here are some naïve ways to design the system:

1. You can have a bus run from the western-most point to the eastern-most point making all $n$ stops. The system would be cheap because it only requires $n-1$ route segments for the entire system. However, a person traveling from stop $i = 1$ to stop $j = n$ has to wait while the bus makes $n-1$ stops.

2. You can have a special express bus from $i$ to $j$ for every stop $i$ to every other stop $j > i$. No person will ever have to make more than one stop. However, this system requires $\Theta(n^2)$ route segments and will thus be expensive.

Using divide-and-conquer, we will find a compromise solution that uses only $\Theta(n \log n)$ route segments, but with the property that a user can get from any stop $i$ to any stop $j > i$ with at most 2 segments. That is, it should be possible to get from any $i$ to any $j > i$ either by taking a direct route $i \to j$ or by taking a two-segment route $i \to m$ and $m \to j$.

(a) For the base cases $n = 1, 2$, design a system using at most 1 route segment.

**Solution:**

(b) For $n > 2$ we will use divide-and-conquer. Assume that we already put in place routes connecting the first $n/2$ stops and routes connecting the last $n/2$ stops so that if $i$ and $j$ both belong to the same half, we can get from $i$ to $j$ in at most 2 segments. Show how to add $O(n)$ additional route segments so that if $i$ is in the first half and $j$ is in the second half we can get from $i$ to $j$ with at most two segments.

**Solution:**

(c) Using part (b), write (in pseudocode) a divide-and-conquer algorithm that takes as input the number of stops $n$ and outputs the list of all the route segments used by your bus system.

**Solution:**

(d) Write the recurrence for the number of route segments your solution uses and solve it. You may use any method for solving the recurrence that we have discussed in class.

**Solution:**

(e) **(Extra Credit Challenge Question! You are not required to submit an answer. A complete answer is worth a quarter of a HW assignment. )** Suppose the MBTA is willing to compromise even further by designing a solution where users may need *three* segments to get from $i$ to $j$. Design a solution that uses asymptotically fewer route segments as the previous divide and conquer solution. That is, it should use $o(n \log n)$ route segments. Clearly show asymptotically how many route segments your algorithm uses.

**Problem 3.** *MergeSort*

(a) Suppose you implement a variant of MergeSort. Instead of breaking the given list into 2 sublists, you break the given list into 3 sublists. Write the complete resulting algorithm here in pseudocode.

**Solution:**

**Algorithm 1:** MergeSort with a 2 piece recursion (model your answer on this)

```
Function MERGESORT(A):
    If LEN(A) = 1 :
        Return A
    Let m ← ⌈LEN(A)/2⌉
    Let L ← A[1 : m], R ← A[m + 1 : n]
    Let L ← MERGESORT(L)
    Let R ← MERGESORT(R)
    Let A ← MERGE(L, R)
    Return A
Function MERGE(L, R):
    Let n ← LEN(L) + LEN(R)
    Let A ← []
    Let j ← 1, k ← 1
    For i = 1 . . . n
        If j > LEN(L) :
            A[i] ← R[k]
            k ← k + 1
        ElseIf k > LEN(R) :
            A[i] ← L[j]
            j ← j + 1
        ElseIf L[j] ≤ R[k] :
            A[i] ← L[j]
            j ← j + 1
        Else
            A[i] ← R[k]
            k ← k + 1
    Return A
```

(b) Is this algorithm more efficient, less efficient, or equally efficient as the standard MergeSort algorithm in terms of asymptotic time complexity? Justify your answer.

**Solution:**

**Problem 4.** *Recurrences*

There are many divide-and-conquer algorithms for multiplying together two integers. In one algorithm, called Karatsuba's algorithm, the product of two $n$-digit numbers can be solved by recursively computing 3 different products of $\frac{n}{2}$-digit numbers. There are algorithms that can multiply $n$-digit numbers recursively by solving:

1. 3 multiplications each of $\frac{n}{2}$-digit numbers (Karatsuba's Algorithm),
2. 5 multiplications each of $\frac{n}{3}$-digit numbers,
3. 7 multiplications each of $\frac{n}{4}$-digit numbers, or
4. 9 multiplications each of $\frac{n}{5}$-digit numbers,

and then doing $O(n)$ work to combine the solutions.

Determine the asymptotic running time of each of these four algorithms and rank them in ascending order of their asymptotic running time. You do not need to justify your answers.

**Solution:**