ICS 311: Algorithms Spring 2025

Problem Set 3

Kyle Berney *Due: Friday, January 31, 2025 at 4pm*

You may discuss the problems with your classmates, however **you must write up the solutions on your own** and **list the names** of every person with whom you discussed each problem.

Start **every** problem on a separate sheet of paper, with the exception of Problems 1 (Peer credit assignment) – Problem 1 can be on the same page as any other problem. Any problem that starts on the same sheet of paper as some other problem will receive 0 points!

1 Peer Credit Assignment (1 point extra credit for replying)

Please list the names of the other members of your peer group for this week and the number of extra credit points you think they deserve for their participation in group work.

- You have a total of 60 points to allocate across all of your peers.
- You can distribute the points equally, give them all to one person, or do something in between.
- You need not allocate all the points available to you.
- ***You cannot allocate any points to yourself!*** Points allocated to yourself will not be recorded.

2 Master Method Practice (30 pts)

Use the Master Method to give tight Θ bounds for the following recurrence relations. Show a, b, and $f(n)$. Then explain why it fits one of the cases, if any. If it fits a case, write and *simplify* the final Θ result.

(a) **(10 pts)** $T(n) = 4T(n/3) + n^2$

(b) **(10 pts)** $T(n) = 7T(n/4) + n$

(c) **(10 pts)** $T(n) = 9T(n/3) + n^2$

3 Substitution Method (30 pts)

Use substitution *as directed below* to solve

$$T(n) = \begin{cases} 7T(n/4) + n & \text{if } n \geq 4 \\ 1 & \text{if } n \leq 3 \end{cases}$$

It is strongly recommended that you read pages 92-93 "A trick of the trade" (page 85-86 "Subtleties" in the 3rd edition) in the CLRS textbook before trying this!

(a) First, use the result from the Master Method in Problem 2 as your "guess" and inductive assumption. We will do this without Θ and c: just use the algebraic portion. Take the proof up to where it fails and say where and why it fails. (See steps below.)

(b) Redo the proof, but subtracting dn from the guess to construct a new guess. Remember, d is just some constant, so you should determine what d is equal to.

(c) Redo the proof by guessing the value of $T(n)$ with the specific value of d that you computed in part (b) above. If you computed $T(n)$ correctly, your proof should succeed.

As a reminder, to do a proof by substitution you:

1. Write the definition $T(n) = 7T(n/4) + n$
2. Replace the $T(n/4)$ with your "guess" instantiated for $n/4$ (you can do that by the inductive hypothesis because $n/4$ is smaller than n).
3. Operating only on the right hand side of the equation, transform that side into the exact form of your "guess".
4. Determine any constraints on the constants involved.
5. Show the base case holds. **For this problem you don't have to worry about the base case. We want you to focus on how to do the other steps.**

4 Recursive Algorithm Analysis (20 pts)

Write down the recurrence, which defines the running time of the following algorithm and solve it using any method you like. Justify your answers.

(30 pts)

```
1:  SillyRecursion(n)
2:      if n < 2
3:          return 1
4:      else
5:          x = 0
6:          for i = 1 to 4
7:              x = x + SillyRecursion(n/2)
8:          for i = 1 to n − 1
9:              for j = i to n
10:                 x = x + 1
11:         return x
```

5 Largest k Elements in an Array (20 pts)

Given an array $A[1..n]$, one way to find the largest $k < n$ elements in the array is to first sort the array in descending order then return the first k elements ($A[1..k]$). This approach has a runtime of $O(n \log n)$ (when using a $O(n \log n)$ sorting algorithm). Provide an implementation using binary heaps that returns the largest k elements in the array more efficiently than the trivial $O(n \log n)$ sorting-based solution and analyze the runtime of your algorithm.

6 Cruel and Unusual Sort (OPTIONAL - 0 pts)

Consider the following cruel and unusual sorting algorithm.

```
1:  CRUEL(A[1..n])
2:      if n > 1
3:          CRUEL(A[1..(n/2)])
4:          CRUEL(A[(n/2 + 1)..n])
5:          UNUSUAL(A[1..n])


1:  UNUSUAL(A[1..n])
2:      if n == 2                                   // The only comparison!
3:          if A[1] > A[2]                          // The only comparison!
4:              temp = A[1]
5:              A[1] = A[2]
6:              A[2] = temp
7:      else
8:          for i = 1 to n/4                        // Swap 2nd and 3rd quarters
9:              temp = A[i + n/4]
10:             A[i + n/4] = A[i + n/2]
11:             A[i + n/2] = temp
12:         UNUSUAL(A[1..(n/2)])                    // recurse on left half
13:         UNUSUAL(A[(n/2 + 1)..n])                // recurse on right half
14:         UNUSUAL(A[(n/4 + 1)..(3n/4)])           // recurse on middle half
```

Notice that the comparisons performed by the algorithm do not depend at all on the values in the input array; such a sorting algorithm is called ***data oblivious***.

Assume for this problem that the input size n is always a power of 2.

(a) Analyze the running time of UNUSUAL. Show your work (write down the recurrence, which defines the running time of the algorithm and solve it using any method you like).

(b) Analyze the running time of CRUEL. Show your work (write down the recurrence, which defines the running time of the algorithm and solve it using any method you like).

7 Building A Heap Using Insertions (OPTIONAL - 0 pts)

BUILD-MAX-HEAP as defined on page 167 of CLRS works by repeatedly calling MAX-HEAPIFY from the middle element down to the first element of the array. Consider an alternative definition:

```
BUILD-MAX-HEAP'(A)
1   A.heap-size = 1
2   for i = 2 to A.length
3       MAX-HEAP-INSERT(A, A[i])
```

(a) (0 pts) Do BUILD-MAX-HEAP and BUILD-MAX-HEAP' always build the same heap when run on the same input array? Prove that they do or provide a counter-example to show that they do not.

(b) (0 pts) The text and lecture notes showed that BUILD-MAX-HEAP requires $O(n)$ time to build an n element heap. Show that there is a worst case where BUILD-MAX-HEAP' requires $\Theta(n \log n)$ time.