CS 6363.003 Homework 1

Due Friday February 12th on eLearning

January 29, 2021

Please answer the following **4** questions, some of which have multiple parts.

Some important homework policies

- Groups of one or two students may work together. They should submit a single copy of their assignment using one of their eLearning accounts. Everybody in the group will receive the same grade.
- Each group must write their solutions in their own words. Clearly print your name(s), the homework number (Homework 1), and the problem number at the top of every page in case we print anything. Start each numbered homework problem on a new page.
- Unless the problem states otherwise, you must justify (prove) (argue) that your solution is correct.
- Please consider using LaTeX to typeset your solutions. Any illegible solutions will be considered incorrect. There is a template provided on the course website to help get you started.
- If you use outside sources or write solutions in close collaboration with others outside your group, then you may cite that source or person and still receive full credit for the solution. Material from the lecture, the textbook, lecture notes, or prerequisite courses need not be cited. Failure to cite other sources or failure to provide solutions in your own words, even if quoting a source, is considered an act of academic dishonesty.
- The homework is assigned to give ***you*** the opportunity to learn where your understanding is lacking and to practice what is taught in class. Its primary purpose *is not* for Kyle to grade how well you paid attention in class. Read through the questions early. Do not expect to know the answers right away. Questions are not necessarily given in order of difficulty. *Please, please, please* attend office hours or email Kyle so he can help you better understand the questions and class material.
- You may assume that reasonable operations involving a constant number of objects of constant complexity may be done in $O(1)$ time. Clearly state your assumptions if they are not something we already used in lecture.

See https://personal.utdallas.edu/~kyle.fox/courses/cs6363.003.21s/about.shtml and https://personal.utdallas.edu/~kyle.fox/courses/cs6363.003.21s/writing.shtml for more detailed policies. If you have any questions about these policies, please do not hesitate to ask during lecture, in office hours, or through email.

1. (a) Truthfully write the phrase ***"I have read and understand the policies on the course website."***

—

Recall the standard recursive definition of the Fibonacci numbers: $F_0 = 0$, $F_1 = 1$, and $F_i = F_{i-1} + F_{i-2}$ for all $i \geq 2$.

(b) Prove that every non-negative integer can be written as the sum of distinct, non-consecutive Fibonacci numbers. That is, if the Fibonacci number F_i appears in the sum, it appears exactly once, and its neighbors F_{i-1} and F_{i+1} do not appear at all. For example:

$$\begin{aligned}17 &= F_7 + F_4 + F_2\\ 42 &= F_9 + F_6\\ 54 &= F_9 + F_7 + F_5 + F_3\end{aligned}$$

[Hint: Use induction to prove a slightly stronger claim that the sum includes a particularly convinient choice of largest member.]

(c) Prove that every positive integer can be written as the sum of distinct Fibonacci numbers *with no consecutive gaps*. That is, for any index $i \geq 1$, if the consecutive Fibonacci numbers F_i and F_{i+1} do not appear in the sum, then no larger Fibonacci number F_j with $j > i$ appears in the sum. In particular, the sum *must* include either F_1 or F_2. For example:

$$\begin{aligned}16 &= F_6 + F_5 + F_3 + F_2\\ 42 &= F_8 + F_7 + F_5 + F_3 + F_1\\ 54 &= F_8 + F_7 + F_6 + F_5 + F_4 + F_3 + F_2 + F_1\end{aligned}$$

[Hint: Use induction to prove a slightly stronger claim that the sum includes a particularly convinient choice of largest member (but a different member from part (b)).]

2. Using Θ-notation, provide asymptotically tight bounds in terms of n for the solution to each of the following recurrences. Assume each recurrence $T(n)$ has a non-trivial base case of $T(n) = \Theta(1)$ for all $n < n_0$ where n_0 is a suitably large constant. For example, if asked to solve $T(n) = 2T(n/2) + n$, then your answer should be $\Theta(n \log n)$. **Give a brief explanation for each solution.** [*Hint: Only some of these recurrences can be solved using the master method, but they can all be solved using recursion trees.*]

(a) $A(n) = 8A(n/3) + n^2$

(b) $B(n) = 27B(n/9) + n^{1.5}$

(c) $C(n) = 7C(n/5) + n$

(d) $D(n) = 2D(n/4) + D(n/2) + n$

(e) $E(n) = 2E(n/2) + n \lg^2 n$

3. The following cruel and unusual sorting algorithm was proposed by Gray Miller:

```
CRUEL($A[1\,..\,n]$):
  if $n > 1$
      CRUEL($A[1\,..\,n/2]$)
      CRUEL($A[n/2+1\,..\,n]$)
      UNUSUAL($A[1\,..\,n]$)
```

```
UNUSUAL($A[1\,..\,n]$):
  if $n = 2$
      if $A[1] > A[2]$                          ⟨⟨the only comparison!⟩⟩
          swap $A[1] \longleftrightarrow A[2]$
  else
      for $i \leftarrow 1$ to $n/4$                   ⟨⟨swap 2nd and 3rd quarters⟩⟩
          swap $A[i+n/4] \longleftrightarrow A[i+n/2]$
      UNUSUAL($A[1\,..\,n/2]$)                  ⟨⟨recurse on left half⟩⟩
      UNUSUAL($A[n/2+1\,..\,n]$)                ⟨⟨recurse on right half⟩⟩
      UNUSUAL($A[n/4+1\,..\,3n/4]$)             ⟨⟨recurse on middle half⟩⟩
```

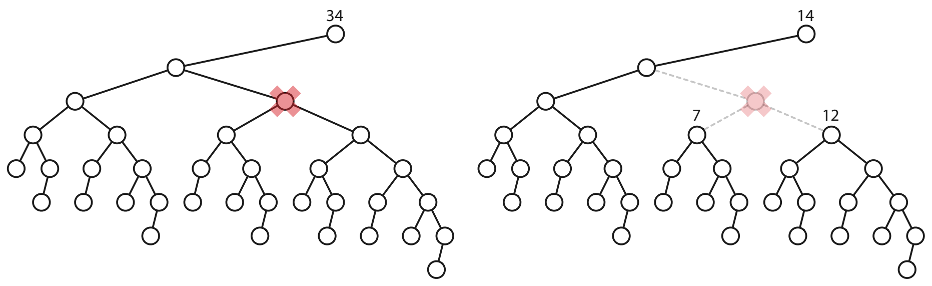
The comparisons performed by this algorithm do not depend at all on the values in the input array; such a sorting algorithm is called ***oblivious***. Assume for this problem that the input size n is always a power of 2.

(a) Prove by induction that CRUEL correctly sorts any input array. *[Hint: UNUSUAL has the same input expectations and effects as another subroutine we saw in class. You might consider what happens to an array that contains $n/4$ 1s, $n/4$ 2s, $n/4$ 3s, and $n/4$ 4s.]*

(b) What is the running time of UNUSUAL? Justify your answer.

(c) What is the running time of CRUEL? Justify your answer.

4. Let T be a binary tree with n vertices. Deleting any vertex v splits T into at most three subtrees, containing the left child of v (if any), the right child of v (if any), and the parent of v (if any). We call v a ***central*** vertex if each of these smaller trees has at most $n/2$ vertices. See the figure below.



Describe and analyze an algorithm to find a central vertex in an arbitrary given binary tree. *[Hint: First prove that every tree has a central vertex.]*