CS 6363.003 Homework 1

Due Friday, February 16 on eLearning

Please solve the following **4** problems, 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.
- Any illegible solutions will be considered incorrect. It is not required, but you might consider using LaTeX to typeset your solutions. There is a template provided on the course website to help you get 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. Seriously, Kyle enjoys busy office hours.
- You may assume that any reasonable operation involving a constant number of objects of constant complexity can 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/cs6363.003.23s/about/ and https://personal.utdallas.edu/~kyle.fox/courses/cs6363.003.23s/writing/ for more detailed policies before you begin. 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."***

   (b) Consider the following algorithm: Procedure STUDY($topics[1 .. n], exams[1 .. t]$) gives instructions on how one might study during a single course with n lectures and t exams. Parameter $topics[1 .. n]$ is an array of lecture topics where $topics[i]$ is the topic covered during the ith lecture of the course. Parameter $exams[1 .. t]$ is an array of distinct integers between 1 and n sorted in increasing order. For all k between 1 and t, there is a cumulative exam held immediately after lecture $exams[k]$. University regulations limit the total number of exams t to be at most $n/4$.

```
STUDY(topics[1 .. n], exams[1 .. t]):
  nextExam ← 1
  for i ← 1 to n
      Study topics[i].                    ⟨⟨Prepare for class.⟩⟩
      if nextExam ≤ t
          if exams[nextExam] = i
              for j ← 1 to i              ⟨⟨Prepare for an exam!⟩⟩
                  Study topics[j].
              nextExam ← nextExam + 1

  Rest and try not to forget everything.
```

   Suppose studying any single topic $topics[i]$ takes $\Theta(1)$ time. Using Θ-notation ***in terms of n only***, give a tight asymptotic bound on the ***maximum*** amount of time spent studying while following the instructions given by STUDY($topics[1 .. n], exams[1 .. t]$). You should justify your solution by arguing 1) you cannot spend any more time than you claim (i.e., argue for a big-Oh bound) and 2) there exists an array $exams$ that results in your claimed total study time (i.e., argue for a Ω-bound).

   (c) Sort the functions of n listed below from asymptotically smallest to asymptotically largest, indicating ties if there are any. **Do not submit proofs for this problem.** To simplify your answers, write $f(n) \ll g(n)$ to mean $f(n) = o(g(n))$, write $f(n) \equiv g(n)$ to mean $f(n) = \Theta(g(n))$, and list all the functions in a sequence of these inequalities. For example, if the given functions were n^2, n, $\binom{n}{2}$, and n^3 then the only correct answers would be "$n \ll n^2 \equiv \binom{n}{2} \ll n^3$" and "$n \ll \binom{n}{2} \equiv n^2 \ll n^3$".

2^n	n^2	n	$\log_9 n$	$\sqrt{n}$
4^n	$\ln^3 n$	$17n$	$n + 500$	$3 - \cos n$
$2^{4 \lg n}$	$\lg(7n)$	250	$\lg^{0.6} n$	$n \log n$

   Advice: You should be able to solve this problem using only what is written for Lecture 2 along with basic algebraic rules for manipulating logs, polynomials, and exponentials.

2. A ***binomial tree of order k*** is defined recursively as follows:

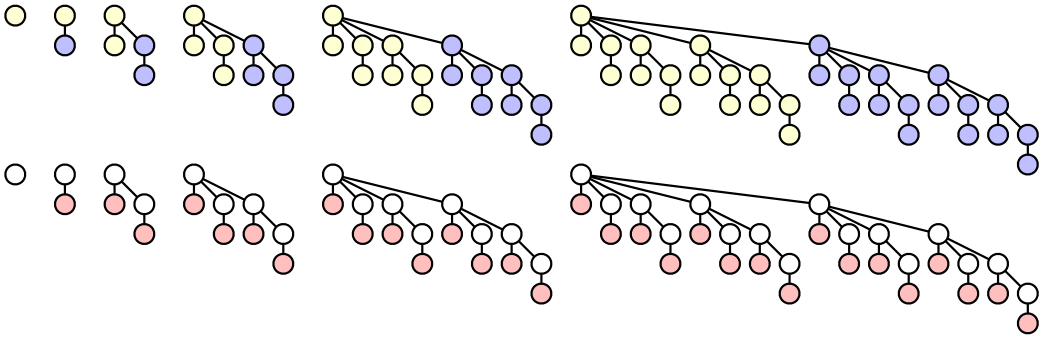
   - A binomial tree of order 0 is a single node.
   - For all $k > 0$, a binomial tree of order k consists of two binomial trees of order $k-1$, with the root of one tree connected as a new child of the root of the other. See the figure below.

   Prove the following claims:

   (a) For all non-negative integers k, a binomial tree of order k has exactly 2^k nodes.

   (b) For all positive integers k, attaching a new leaf to every node in a binomial tree of order $k-1$ results in a binomial tree of order k.

   (c) For all non-negative integers k and d, a binomial tree of order k has exactly $\binom{k}{d}$ nodes with depth d. (Hence the name!)
   Advice: You may recall $\binom{k}{d} = \binom{k-1}{d-1} + \binom{k-1}{d}$ *whenever* k *is positive.*



Binomial trees of order 0 through 5.
Top row: The recursive definition. Bottom row: The property claimed in part (b).

Advice: You may find the following template useful for part (a). You can modify it as necessary for parts (b) and (c).

> Let k be an arbitrary non-negative integer.
> Assume that for any non-negative integer $k' < k$, a binomial tree of order k' has exactly $2^{k'}$ nodes. There are several cases to consider:
>
> - Suppose k is…
> - Suppose k is…
> - …
> - Suppose k is…
>   The induction hypothesis implies that…
>
> In each case, we conclude a binomial tree of order k has exactly 2^k nodes.

3. 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.*

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

   (b) $B(n) = 8B(n/2) + n^3$

   (c) $C(n) = 5C(n/3) + n$

   (d) $D(n) = D(n/3) + D(2n/3) + n$

   (e) $E(n) = 4E(n/4) + n \lg n$

   Advice: One of the three common cases discussed in Lecture 4 still applies for part D, even though the subproblem sizes are not equal. For part E, solve it without the $\lg n$ *factor at the end and then see what the* $\lg n$ *factor does to each of the recursion tree level sums.*

4. An ***inversion*** in an array $A[1 .. n]$ is a pair of indices (i, j) such that $i < j$ and $A[i] > A[j]$. The number of inversions in an n-element array is between 0 (if the array is sorted) and $\binom{n}{2}$ (if the array is sorted backward). Describe and analyze an algorithm to count the number of inversions in an n-element array in an $O(n \log n)$ time.

   Your analysis on this and future algorithm design problems need only argue for a good big-Oh bound unless we say otherwise. Don't forget to justify why your algorithm correctly counts the inversions!

   Advice: Modify mergesort and its merge procedure so they not only sort the array but also return the number of inversions it contained before sorting. Be sure to update your count correctly as you discover inversions during the merge procedure.