CSCB63 Assignment 1 - Winter 2021
Due Noon (12pm EST), Feb. 7th, 2021

Only a subset of the questions will be graded however you are responsible for all the material on this assignment. Read the guidelines for plagiarism on the course website.

1. We often see time (or space) complexity written as a recurrence relation (think of any recursive algorithm). In order to determine the complexity in asymptotic notation ($\mathcal{O}(\cdot), \Omega(\cdot), \Theta(\cdot)$) we solve the recurrence relation as a *closed form function.* A recurrence relation is a function $f(n)$ defined in terms of itself. A closed form function $f(n)$ is defined in terms of n. There are two common ways to do this; repeated back substitution or apply the Master Theorem. Solving complexity recurrence relations is an important skill you will need in CSCC73.

   We will illustrate both methods for the time complexity recurrence of Merge Sort, $T(n)$, which can be expressed as

   $$T(n) = 2T(\frac{n}{2}) + n \text{ and } T(1) = 0$$

   Repeated back substitution is the process of repeatedly substituting into the recurrence. For simplicity assume n is a power of 2.

   $T(n) = 2T(\frac{n}{2}) + n$ and observe that $T(\frac{n}{2}) = 2T(\frac{n}{4}) + \frac{n}{2}$ so we can replace the $T(\frac{n}{2})$ with $2T(\frac{n}{4}) + \frac{n}{2}$.

   $T(n) = 2(2T(\frac{n}{4}) + \frac{n}{2}) + n$ and applying the same concept again gives:

   $T(n) = 2(2(2T(\frac{n}{8}) + \frac{n}{4}) + \frac{n}{2}) + n$. Repeating until $\frac{n}{2^i} = 1$, ie, until $i = \log_2 n$.

   $$T(n) = \sum_{i=0}^{\log_2 n} 2^i \frac{n}{2^i}$$

   Simplifying the summation gives

   $$T(n) = \sum_{i=0}^{\log_2 n} n = n \log_2 n$$

   so Merge Sort's asymptotic complexity $T(n) \in \Theta(n \log_2 n)$.

   If your recurrence has the correct format, you can apply the Master Theorem to find the closed form:

   Generalized Master Theorem

   Let a and b be positive real numbers with $a \geq 1$ and $b \geq 2$. Let $T(n)$ be defined by

   $$T(n) = \begin{cases} aT(\lceil n/b \rceil) + f(n) & if\ n > 1 \\ d & if\ n = 1 \end{cases}$$

   Then

   (a) if $f(n) = \Theta(n^c)$ where $log_b a < c$, then $T(n) = \Theta(n^c) = \Theta(f(n))$.

   (b) if $f(n) = \Theta(n^c)$, where $log_b a = c$, then $T(n) = \Theta(n^{\log_b a} \log_b n)$.

   (c) if $f(n) = \Theta(n^c)$, where $log_b a > c$, then $T(n) = \Theta(n^{\log_b a})$.

The same results apply with ceilings replaced by floors.

For example, the run time of Mergesort, $T(n)$, can be expressed as

$$T(n) = 2T(\frac{n}{2}) + n$$

Applying the Master Theorem produces

$$T(n) = \Theta(n^{\log_b a} \log n) = \Theta(n^{\log_1 1} \log n) = \Theta(n \log n).$$

For each of the following determine the asymptotic bound $\Theta(n)$ for the recurrence relation.

(a) $A(n) = n^2 A(n-1)$ for $n > 1$ and $A(1) = 1$.

(b) $C(n) = 8C(\lfloor n/2 \rfloor) + 2n^3 + 4n$, where $C(0) = 6$. You can give your answer in Θ notation.

2. Let f and g be functions such that $f(n) \geq 1$ and $\lg(g(n)) \geq 1$ for all natural n.

   Prove: If $f(n) \in \mathcal{O}(g(n))$, then $\lg(f(n)) \in \mathcal{O}(\lg(g(n)))$.

3. Consider an abstract datatype RECENT that keeps track of *recently* accessed values. In this ADT, we consider a set $S = \{1, ..., n\}$ and a subset R of S, of size m. Initially, $R = \emptyset$. The only operation is *access*, which takes a parameter $i \in S$. It adds i to R and, if this causes the size of R to become bigger than m, removes the element from R that was least recently the parameter of an access operation.

   Give a data structure for this abstract data type that uses $O(m \log n)$ space (measured in bits) and has worst case time complexity $O(\log m)$. Justify that your data structure is correct and satisfies the desired complexity bounds.

4. Consider inserting the following keys in order into an AVL tree. Use the convention from lecture of calculating the balance factor = height of left subtree minus height of right subtree. Only show your tree at the indicated times.

$$5, 4, 6, 9, 12, 16, 14, 2, 3$$

   Show your tree. Now insert

$$1, 20, 19, 18, 17$$

   Show your tree. Now using the predecessor when there is a choice, delete in order

$$1, 2, 3, 4, 5, 12, 6$$

   Show your tree.

5. **Programming question *warm up*.** To enable you to solve the programming question answer the following question first. It will *not be graded* but will make coding much easier.

   Starter code (in C) and description coming soon!

   Consider an ADT consisting of a set S of distinct integers and the following operations:

   INSERT(S,x): Insert the element x into the set S. This operation has no effect if x is already in S.

   DELETE(S,x): Delete the element x from the set S. This operation has no effect if x is not in S.

   QUERY(S,x): Return true if x is in S and return false if x is not in S.

CLOSEST-PAIR(S): Return two integers in S which are closest together in value.

In other words, if CLOSEST-PAIR(S) returns the integers a and b, then they must satisfy the condition

$$\forall x \forall y (x \neq y \rightarrow |a-b| \leq |x-y|).$$

It is an error if S contains fewer than two elements.

Describe a data structure to implement this ADT. All the usual operations of INSERT, DELETE and QUERY must be performed in $O(\log n)$ time, where $n = |S|$ and CLOSEST-PAIR must be in $O(1)$.

Describe any new information that will be stored. Justify why your algorithms are correct and why they achieve the required time bound.