# 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 $T(n) = \Theta(n^{\log_b a})$.*

# CSCB63 Assignment 2

*Due: 11.59pm, Saturday March 6, 2021 ⇐ Note New Deadline*

1. (not for marks) Make sure you are capable of applying BFS, DFS, Prim's algorithm, Kruskal's algorithm and Dijkstra's algorithm to an appropriate type of graph (directed, not directed or weighted).

2. In class, we have (or will soon) seen Dijkstra's algorithm to find all the shortest paths from a single source in a weighted graph. In this question, you will construct a graph that has many shortest paths, in fact, your graph will have $3^n$ shortest paths between a source vertex $s$ and a sink vertex $t$, where the number of vertices is a function of the form $cn + k$ for some $c, k \in \mathbb{N}$. More precisely, prove:

   > *For every natural number $n$, there is an undirected graph of $cn + k$ vertices such that for some pair of vertices $s$ and $t$ in the graph, there are $3^n$ shortest paths from $s$ to $t$.*

   You select the constants $k$ and $c$ to make it work.

3. Determine whether the following claim is true and either prove the claim or it's negation.

   > *Given a graph $G$ with $n$ vertices such that for every $v \in V$, **deg**$(v) \geq \frac{n}{2}$ then $G$ is one connected component.*

4. We can model the build process of an object as a directed graph. For example, suppose we are building a house. The walls can't be painted until the drywall is installed which cannot happen until after the studs are built. Suppose we model the process with a vertex representing the components of the object and a directed edge from $a$ to $b$ if $a$ must be completed before $b$ can be completed. Notice that if this directed graph has a cycle, then there is no way to construct the object.

   (a) Give an algorithm to determine whether a directed graph has a cycle. What should your complexity be?

   (b) Using DFS, construct an algorithm that either returns a valid ordering of the vertices to build the object or a cycle confirming no such ordering exists. Again, what should your complexity be?

5. Consider an undirected graph $G = (V, E)$ with non-distinct, non-negative edge weights. If the edge weights are not distinct, it is possible to have more than one MST. Suppose we have a spanning tree $T \subset E$ with the guarantee that for every $e \in T$, $e$ belongs to some minimum-cost spanning tree in $G$. Can we conclude that $T$ itself must be a minimum-cost spanning tree in $G$? Give a proof or a counter example with explanation.

6. The programming assignment for A2 and A3 is a combined assignments that will be posted on Markus. It will not be due until A3 is due as it requires a bit more thought and problem solving that A1. As with any assignment, make sure that the work is your own - do not copy for another student or a website.

# CSCB63 Assignment 3 - Winter 2021
# Due NOON, Sunday Mar. 21, 2021

*Only a subset of the questions will be graded however you are responsible for all the material on this assignment.*

1. In class we looked at a dynamic array that expanded by doubling every time it is full. We now consider a variation that expands by 1.5 instead. In other words, if the current dimension of the array is $n$ then when full, an append operation would create a new array of size $\lceil 1.5n \rceil$ and move all the elements over appending the new element in the first available location. Use the *potential* method to prove that the amortized complexity of this variation is still $\mathcal{O}(1)$.

   You should define your function $\phi(h)$ showing that is satisfies the requirements of a potential function and then calculate the amortized cost for append when an expansion does not occur and when it does occur (as we did in class).

2. Consider the following data structure for representing a set. The elements of the set are stored in a singly linked list of sorted arrays, where the number of elements in each array is a power of 2 and the sizes of all the arrays in the list are different. Each element in the set occurs in exactly one array. The arrays in the linked list are kept in order of increasing size.

   To perform SEARCH, perform binary search separately on each array in the list until either the desired element is found or all arrays have been considered.

   To insert a new element $x$ into the set (given the precondition that $x$ is not already in the set),

   ```
   create a new array of size 1 containing x
   insert this new array at the beginning of the linked list
   while the linked list contains 2 arrays of the same size
      merge  the 2 arrays into one (sorted) array of twice the size
   ```

   (a) Draw the data structure that results after inserting the following sequence of elements into an initially empty set:

   $$2, 63, 1, 32, 77, 8$$

   (b) What is the worst case time, to within a constant factor, for performing SEARCH when the set has size $n$? Justify your answer.

   HINT: Using the following notation may help you express your answer: "*Let $I_n$ denote the set of bit positions in the binary representation of $n$ that contain the value 1.*".

   (c) What is the worst case time, to within a constant factor, for performing INSERT when the set has size $n$? Justify your answer.

   (d) Use the aggregate method to prove that the amortized insertion time in a sequence of $n$ insertions, starting with an initially empty set, is $O(\log n)$.

   (e) Use the accounting method to prove that the amortized insertion time in a sequence of $n$ insertions, starting with an initially empty set, is $O(\log n)$.

3. Let $L$ be a circularly linked list implementation of the disjoint set ADT with extra pointers to the representative of the list and using union-by-weight. A circularly linked list is simply a list with a pointer from the tail to the head. Let $T$ be a tree implementation of the disjoint set ADT using union-by-rank and path compression.