Matthew Skipworth
TCSS 343
Homework2

1.) We're given a self-reduction for a MAX problem and we're asked to state the recursive algorithm using pseudo-code. Below is the self-reduction:

$$M(A[a...b]) = \begin{cases} A[a] \ if \ a{=}b \\ \\ max(A[a],\ M(A[a{+}1...b])) \ if \ a{<}b \end{cases}$$

As we can see from the self-reduction, the base case occurs when $a = b$ and the element a is returned. When a is less than b however, a max function is called on the element a and a recursive call of the function M on the elements $a + 1$ through b until just $b - 1$ and b are being compared. The max of the two elements $b - 1$ and b is returned and then the call stack pops $b - 2$ and the max is then again called on the current maximum element and $b - 2$. This process continues until the call stack is empty and the maximum element of the list is returned.

2.)Using the same reduction as part 1 we're asked to state a recurrence $T(n)$ that expresses the worst case run time of the recursive algorithm. We're also asked to find a similar recurrence in our notes and state the tight bound run time. We notice that if the index of the element we're comparing a is equal to the index of the last element b, then we simply return that element. The runtime for this operation is constant c. We also notice that for elements whose index is not equal to element b, that the number of elements in the list is decreasing with each recursive call and so for the M operation the runtime is $T(n - 1)$. For each max comparison, we simply perform an arithmetic operation which happens in constant time d. Putting this all together we get the recurrence relation:

$$T(n) = \begin{cases} c, \ if \ n{\geq}1 \\ \\ T(n{-}1){+}d, \ if \ n{>}1 \end{cases}$$

3.)We're given a self-reduction for the MAX problem and asked to state a recursive algorithm using pseudocode for finding the maximum element based on the self-reduction. Below is the self-reduction:

$$M(A[a \ldots b]) = \begin{cases} -\infty & \text{if } a > b \\ A[a] & \text{if } a = b \\ \max(M(A[a \ldots t_1]), \max(M(A[t_1 + 1 \ldots t_2]), M(A[t_2 + 1 \ldots b]))) & \text{if } a < b \end{cases}$$

For the first case we notice that the algorithm should return negative infinity in the instance that element a has a larger index than element b. We note that this will only happen if an error has occurred in the execution of the algorithm and have it return

negative infinity. The next case will occur when the index of element a is equal to the index of element b, in other words the algorithm will have gone through the entire array, we will have located the maximum element of the array, or the array size will be one in which the first option is also true. In this case we simply return the value of the index a. For the case in which a is less than b, the algorithm will perform the recursive call of the max function on the first element of the first third of the list $A[a]$ with the result of a max function call on the first element of the second $A[t_1]$ and third $A[t_2]$ thirds of the list. On each recursive call the index of the element to be compared is incremented so that eventually the elements being compared are $A[t_1 - 1]$ and the max result between $A[t_2 - 1]$ and $A[b]$. At this point the call stack will pop the result and compare it with the next elements. The call stack will continue to pop the current max and compare it until the call stack is empty. The algorithm will then return the maximum element of the list.

4.) Next we're asked to state a recurrence relation $T(n)$ that expresses the worst case runtime of the algorithm from problem 3. We note that for a list of size less than or equal to 1, the runtime $T(n)$ is constant, c. This is because the list containing 1 element has already located the max and thus only needs to return the element $A[a]$. For instances where n is greater than 1 however, the list will split up into thirds recursively and compare the elements until the max is found. In this case $T(n) = 3T(\frac{n}{3}) + d$. Note the following recursion tree table:

#	*size*	*cost*
3^0	$n = \frac{n}{3^0}$	d
3^1	$\frac{n}{3} = \frac{n}{3^1}$	d
3^2	$\frac{n}{9} = \frac{n}{3^2}$	d
i	$\frac{n}{3^i}$	d
2^k	1	c