Solutions for the Midterm Exam

1. Use the Master Theorem to find the asymptotic solution for the following recurrence: \( T(n) = 10T\left(\frac{n}{3}\right) + 3n^2 \).

   **Solution:** We have \( a = 10, \ b = 3, \)
   
   \[ n^{\log_3 10} / f(n) = n^{\log_3 10} / n^2 > n^\epsilon, \]

   for some \( \epsilon > 0 \), we get Case 1, and thus \( T(n) = \Theta(n^{\log_3 10}) \). (10 points)

2. In this problem we consider the simultaneous minimum and maximum problem. That is, given an array of distinct real numbers \( x_1; x_2; \ldots; x_n \), we must find the minimum and the maximum of these numbers simultaneously.

   (a) Describe a divide-and-conquer algorithm that solves this problem.

   (b) Find a recursion for the running time of the algorithm you described in part (a). Use the Master Theorem to estimate the running time.

   (c) What is the exact running time? (You may assume for simplicity that \( n \) is a power of 2.) Is this optimal?

   **Solution:** (a) Algorithm:

   - Divide: we divide the problem into two subproblems, the first \( \lfloor n/2 \rfloor \) numbers \( x_1, \ldots, x_{\lfloor n/2 \rfloor} \), and the numbers \( x_{\lfloor n/2 \rfloor + 1}, \ldots, x_n \).
   - Conquer: we solve the two subproblems recursively.
   - Combine: we combine the solutions in the following way. The minimum is the smaller of the two minimums and the maximum is the larger of the two maximums.
(b) Thus for the running time we get the following recursion:

\[ T(n) = 2T\left(\frac{n}{2}\right) + 2. \]

By the Master Theorem, since we have \( a = 2, \ b = 2, \)

\[ n^{\log_2 2} / 1 = n, \]

we get Case 1, and thus \( T(n) = \Theta(n) . \)

(c) The exact solution is \( T(n) = \frac{3n}{2} - 2, \) which can be easily proved by induction: \( T(2) = 1 \) and

\[ T(2n) = 2T(n) + 2 = 2 \left( \frac{3n}{2} - 2 \right) + 2 = 3n - 2. \]

This is optimal as we discussed in class. (15 points)

3. Suppose that 50 students take this exam. Assume that the exam grades are uniformly and independently distributed at random across the grades A, B and C (this will not be the case). What is the expected number of students who will NOT get an A?

**Solution:** Let \( m = 50 \) and

\[ X_i = \begin{cases} 1 & \text{the } i^{th} \text{ student gets a B or C} \\ 0 & \text{otherwise} \end{cases} \]

Then \( X = \sum_{i=1}^{m} X_i \) is exactly the number of students who will not get an A. Then by the linearity of expectation

\[ E(X) = E\left(\sum_{i=1}^{m} X_i\right) = \sum_{i=1}^{m} E(X_i) = \]

\[ = \sum_{i=1}^{m} Pr(\text{the } i^{th} \text{ student gets a B or C}) = \]

\[ = \sum_{i=1}^{m} \frac{2}{3} = \frac{100}{3}. \]

(15 points)
4. A sequence \( x_1, x_2, \ldots, x_n \) is \textit{cyclically sorted} if the smallest number in the sequence is \( x_i \) for some unknown \( i \), and the sequence

\[
x_i, x_{i+1}, \ldots, x_n, x_1, \ldots, x_{i-1}
\]

is sorted in increasing order. Suppose we are given a cyclically sorted list of distinct numbers.

(a) Describe an \( O(\log n) \)-time algorithm that finds the position of the smallest number.

(b) Show that your algorithm is optimal, i.e. we have a \( \log n \) worst-case lower bound for the number of comparisons needed.

Solution: (a) This is basically just binary search. Let the current search interval be \([x_k, x_l]\) with \( k < l \) and let \( m = \lceil \frac{k+l}{2} \rceil \).

If \( x_m < x_l \), then \( i \) cannot be in \( m < i \leq l \) (so we can throw away the right half).

If \( x_m > x_l \), then \( i \) cannot be in \( k \leq i \leq m \) (so we can throw away the left half).

(b) The number of possible answers is \( n \), so this follows from the decision tree lower bound. (15 points)

5. Describe a \( O(n) \) worst-case time algorithm that, given a set \( S \) of \( n \) distinct numbers and \( k = \lceil \frac{n}{\log n} \rceil \), finds the \( k \) numbers of \( S \) in sorted order that are closest to the median of \( S \) in the sorted order of \( S \). (For simplicity we assume that both \( n \) and \( k \) are odd, so there is one median. This will be the median of both the \( n \) numbers and the \( k \) numbers.)

Solution: Let \( m = \frac{n+1}{2} \) (the rank of the median). Using the worst-case linear time Selection algorithm twice, we find the two elements \( x_1, x_2 \) with ranks \( m - \frac{k-1}{2} \) and \( m + \frac{k-1}{2} \). Then we go through \( S \) and find all elements \( x \) for which \( x_1 \leq x \leq x_2 \), these are the \( k \) numbers. Then we just have to sort them. The total worst-case running time is

\[
O(n) + O(k \log k) = O(n) + O\left(\frac{n}{\log n} \log n\right) = O(n).
\]

(15 points)
6. Suppose we are given an unlimited supply of coins of denominations \( x_1, x_2, \ldots, x_k \), we wish to make change for a value \( n \); that is, we wish to find a set of coins whose total value is \( n \). This might not be possible: for example, if the denominations are 5 and 10 then we can make change for 15 but not for 12. Design a dynamic programming algorithm that given \( x_1, x_2, \ldots, x_k \) and \( n \) determines whether it is possible to make change for \( n \). Draw the subproblem graph. How many vertices and edges are there in the subproblem graph? What is the running time of your algorithm?

**Solution:** Here is the algorithm:

Make-Change\((x_1, x_2, \ldots, x_k, n)\)

let \( T[0..n] \) be a new array, where \( T[i] = 1 \) if it’s possible to make change for \( i \)

\( T[0] = 1 \)

\( \text{for } i = 1 \text{ to } n \)

\( \quad \text{if } i - x_j \geq 0 \text{ and } T[i - x_j] = 1 \text{ for some } 1 \leq j \leq k \text{ then } T[i] = 1 \)

\( \quad \text{else } T[i] = 0 \)

\( \text{return } T[n] \)

Each \( T[i] \) is computed from the \( k \) numbers \( T[i - x_j] = 1, 1 \leq j \leq k \). The running time is clearly \( O(kn) \). The subproblem graph consists of \( n + 1 \) vertices, \( v_0, v_n, \ldots, v_n \). There are at most \( k \) edges leaving each vertex, thus the total number of edges is at most \( kn \). (15 points)

7. Consider the activity selection problem but instead of always selecting the first activity to finish, we instead select the first activity to start that is compatible with all previously selected activities. Is this a greedy strategy? Does it yield an optimal solution? Explain!

**Solution:** No. Counterexample: \( n = 3 \). For \( a_1 \), we have \( s_1 = 0, f_1 = 3 \), for \( a_2 \), we have \( s_2 = 1, f_2 = 2 \) and for \( a_3 \), we have \( s_3 = 2, f_3 = 3 \). The greedy strategy will select \( \{a_1\} \) even tough the optimal solution is \( \{a_2, a_3\} \). (15 points)