1. Use the Master Theorem to find the asymptotic solutions for the following recurrences:

   (a) \( T(n) = 7T(n/2) + n^2 \),
   (b) \( T(n) = T(n/2) + 1 \),
   (c) \( T(n) = 4T(n/2) + n^3 \).

Solution:

   (a) We have \( a = 7, b = 2, \)
   \[
   f(n)/n^{\log_2 7} = n^2/n^{\log_2 7} = n^{2-\log_2 7} = O(n^{-0.8}),
   \]
   we get Case 1, and thus \( T(n) = \Theta(n^{\log_2 7}) \).
   
   (b) We have \( a = 1, b = 2, \)
   \[
   f(n) = \Theta(1) = \Theta(n^{\log_2 1}) = \Theta(1),
   \]
   we get Case 2, and thus \( T(n) = \Theta(\log_2 n) \).
   
   (c) We have \( a = 4, b = 2, \)
   \[
   f(n)/n^{\log_2 4} = n^3/n^2 = n,
   \]
   we get Case 3, and thus \( T(n) = \Theta(n^3) \). Note that the regularity condition is satisfied as \( 4(n/2)^3 = n^3/2 \leq cn^3 \) for \( c = 1/2 \).
2. Use indicator random variables to find the expected value of the number of fixed elements (elements left in the same position) in a random permutation of $n$ elements.

Solution: Let

$$X_i = \begin{cases} 1 & \text{if the } i\text{th element is a fixed element} \\ 0 & \text{otherwise} \end{cases}$$

Then by the linearity of expectation

$$E(X) = E\left(\sum_{i=1}^{n} X_i\right) = \sum_{i=1}^{n} E(X_i) = \sum_{i=1}^{n} \frac{1}{n} = 1.$$

(15 points)

3. We have two input arrays, an array $A$ with $m$ elements and an array $B$ with $n$ elements, where $m \leq n$. There may be duplicate elements. We want to decide if every element of $B$ is an element of $A$. Describe an algorithm to solve this problem in $O(n \log m)$ worst-case time.

Solution: First we sort $A$ by MERGESORT (in $O(m \log m)$ time). Then for each element of $B$ we do a binary search in the sorted list of $A$ (in $O(n \log m)$ time). The total worst-case running time is $O((m + n) \log m) = O(n \log m)$. (15 points)

4. Assume that you want to sort an array of $n$ numbers, each of which is a member of the set $\{0, 1, 2, 3, 4\}$. A sample input for $n = 6$ is $(3, 1, 0, 3, 4, 3)$. Describe an optimal algorithm to solve this problem and prove that it is optimal; that is give a lower bound on the worst-case running time that has the same order of magnitude as the worst-case running time of your algorithm.

Solution: This is just COUNTING-SORT with $k = 4$ with $O(n)$ running time. We can give a linear time lower bound for solving the problem by observing that if an algorithm does not examine some element (examining all elements takes linear time), then this element may be changed by the adversary and the algorithm will give the same answer, which is now incorrect. (15 points)
5. We have an input array $A$ with $n$ ($n \geq 2$) numbers.

(a) Describe a $O(n)$ worst-case time algorithm to find two elements $x, y \in A$ such that $|x - y| \geq |u - v|$ for all $u, v \in A$.

(b) Describe a $O(n \log n)$ worst-case time algorithm to find two elements $x, y \in A$ such that $|x - y| \leq |u - v|$ for all $u, v \in A$.

Solution: (a) For this we have to find the minimum and the maximum simultaneously which we can do in $3 \lceil n/2 \rceil = O(n)$ time.

(b) For this we sort the numbers first, then $x$ and $y$ must be consecutive elements in the sorted order. We go through the sorted list and we find the smallest difference between two neighboring elements. (15 points)

6. An array $A[1..n]$ of $n$ distinct numbers is called unimodal if there is a unique mode $j$, such that $A[i] > A[i+1]$ for $j \leq i < n$ and $A[i-1] < A[i]$ for $1 < i \leq j$.

(a) Assume that you have an algorithm that tests whether $A$ is unimodal and if it is then finds the mode. Use an adversary argument to show that your algorithm must have a worst-case time complexity $\Omega(n)$.

(b) Assuming that $A$ is unimodal, give an algorithm for finding the mode for which the worst-case time complexity is $O(\log n)$. Isn’t this contradictory with part (a)?

Solution: (a) A necessary condition for an array to be unimodal with mode $j$ is that it must be sorted in increasing order for $1 \leq i \leq j$ and in decreasing order for $j \leq i \leq n$. Assume that there is an algorithm to test whether $A$ is unimodal and if it is then finds the mode in worst-case $o(n)$. Then some element, say $A[k]$, is not examined by the algorithm. Then the adversary changes $A$ to $A^*$ where $A^*[i] = A[i]$ for $i \neq k$ but $A^*[k]$ is changed so that $A^*$ is not unimodal, say $A^*[k] = -\infty$ if $1 < k \leq j$ and $A^*[k] = \infty$ if $k > j$ or $k = 1$. But then every question asked by the algorithm on $A^*$ is exactly the same question asked on $A$ by the algorithm, and it receives the same answer. Hence it must report that $A^*$ is unimodal, which is incorrect, and contradicts that the algorithm works correctly. Hence the complexity of testing for unimodality is $\Omega(n)$.
(b) If we know that A is unimodal, then we seek the mode by binary search in $O(\log n)$ time:

\begin{verbatim}
i = 1, j = n
while i < j
    m = ⌊(i+j)/2⌋
    if A[m-1] < A[m] and A[m] > A[m+1]
        return m
    else if A[m-1] < A[m] then i = m + 1
    else j = m - 1
\end{verbatim}

(15 points)

7. In a variant of open-address hashing we insert the elements in pairs $(k_1, k_2)$ into a hash table, that is we insert $k_1$ into the first empty slot in the probe sequence, then we start from scratch and insert $k_2$ into the first empty slot again. What is the expected number of probes needed to insert a pair of keys into a hash table with load factor $\alpha = n/m$, assuming uniform hashing?

**Solution:** To insert the first key into an open-address hash table the expected number of probes is $\frac{1}{1-\alpha} = \frac{1}{1-\frac{n}{m}}$. To insert the second element the expected number of probes is $\frac{1}{1-\frac{n+1}{m}}$. Thus the expected total number of probes is

\[
\frac{1}{1-\frac{n}{m}} + \frac{1}{1-\frac{n+1}{m}} = \frac{m}{m-n} + \frac{m}{m-n-1}.
\]

(15 points)