CS 4120 Analysis of Algorithms  
A term 2018  
Solutions for the Midterm Exam

1. Use the Master Theorem to find the asymptotic solution for the following recurrence:  
   \[ T(n) = 16T\left(\frac{n}{4}\right) + n^2. \]

   **Solution:** We have \( a = 16, \ b = 4, \)
   \[ f(n) = n^2 = \Theta(n^{\log_4 16}) = \Theta(n^2), \]
   we get Case 2, and thus \( T(n) = \Theta(n^2 \log_2 n). \) (20 points)

2. A **dodecahedral die** has 12 faces that are numbered 1 through 12. Suppose that we roll 12 fair dodecahedral dice simultaneously. What is the expected number of dice that come up with 1?

   **Solution:** Let
   \[ X_i = \begin{cases} 
   1 & \text{if the } i\text{th die comes up with 1} \\
   0 & \text{otherwise} 
   \end{cases} \]
   Then by the linearity of expectation
   \[ E(X) = E\left(\sum_{i=1}^{12} X_i\right) = \sum_{i=1}^{12} E(X_i) = \sum_{i=1}^{12} \frac{1}{12} = 1. \]
   (20 points)

3. The input is a sequence \( x_1, x_2, \ldots, x_n \) of integers in an arbitrary order, and another sequence \( a_1, a_2, \ldots, a_n \) that is a permutation of the integers from 1 to \( n. \) Both sequences are given as arrays. Design an \( O(n \log n) \)-time in-place algorithm to order the first sequence according to the order imposed by the permutation. In other words, for each \( i, \) \( x_i \) should appear in the output in the position given in \( a_i. \) Show that your algorithm is optimal apart from the constant in the big-\( O \) notation.

   **Solution:** The permutation \( a_1, a_2, \ldots, a_n \) defines a total order on the sequence \( x_1, x_2, \ldots, x_n, \) that’s what we have to find. That is, we can
“compare” any pair of elements $x_i$ and $x_j$ by comparing $a_i$ and $a_j$. Therefore, any sorting algorithm that sorts according to the values of the $a_i$’s and moves both $x_i$ and $a_i$ together will lead to the desired outcome. Since we want an in-place algorithm, we can use, for example, HEAP-SORT. There are $n!$ possible outputs, so by the decision tree lower bound method, any algorithm solving this problem must have worst-case running time
\[
\Omega(\log (n!)) = \Omega(n \log n).
\]

(20 points)

4. Describe a $O(n)$ worst-case time algorithm that, given a set $S$ of $n$ distinct numbers and a positive integer $k \leq n$, determines the $k$ numbers in $S$ 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).

**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 solutions. The total worst-case running time is $O(n)$. (20 points)

5. Give an $O(n)$-time dynamic programming algorithm for the maximum subarray problem. That is, given an array of real numbers $a_1, a_2, \ldots, a_n$, your algorithm must compute the maximum sum $\sum_{i=j}^{k} a_i$, where $1 \leq j \leq k \leq n$. Draw the subproblem graph. How many vertices and edges are in the graph?

**Solution:** Here is the algorithm:

Maximum-subarray($n$)
let $M[1..n]$ be a new array
$M[1] = a_1$
for $i = 2$ to $n$
    $M[i] = \max(M[i-1] + a_i, a_i)$
we find the maximum $M$ of $M[i], 1 \leq i \leq n$
return $M$
Each number in the sequence is computed from the previous number in the sequence. The running time is clearly $O(n)$. The subproblem graph consists of $n$ vertices, $v_1, \ldots, v_n$. For $i = 2, \ldots, n$, vertex $v_i$ has one leaving edge: to vertex $v_{i-1}$. No edge leaves vertex $v_1$. Thus, the subproblem graph has $(n - 1)$ edges. (20 points)