CS21003: Algorithms-I (Theory)
Tutorial – 2 (Divide-and-Conquer Algorithms)
Date: 30-January-2020

1. Let $X = \{x_1, x_2, \ldots, x_n\}$ be an array of n integers. A *majority element* in this array is an element $x_i \in X$ which occurs (repeats) at least $\lceil \frac{n}{2} \rceil$ number of times. If the existence of such an element is *not* found, then we simply say that the array has no majority element. We can easily find the majority element of a given n-element integer array using the following two methods:

   (i) Use sorting techniques to find the majority element in $O(n \log n)$ time.

   (ii) Use median finding approach to find the majority element in $O(n)$ time.

   For the two above-mentioned approaches, formulate the initial recursive definitions and convert these into algorithmic pseudo-codes. Then, verify your solution and derive the given time-complexities as well.

   Now, if you are also provided with the fact that the majority element *definitely* exists for a given array, can you think of any other efficient $O(n)$-time algorithm, without using sorting or median-finding procedures? Write the initial recursive definition and derive the time-complexity of the proposed algorithm.

2. Let $X = \{x_1, x_2, \ldots, x_n\}$ be an array of n integers and x is an integer. Propose an efficient algorithm to determine whether there are *two* elements in X whose sum is exactly x. Derive the time-complexity of the proposed algorithm.

3. You are given an array $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$ of n distinct integers. It is given that the elements of $\mathcal{A}$ satisfy the following inequalities: $a_1 < a_2 < \cdots < a_{m-1} < a_m > a_{m+1} > a_{m+2} > \cdots > a_n$, for some (unknown) index m in the range $1 < m < n$. Let us call such an array a *hill-valued array*. The sequence $a_1, a_2, \ldots, a_m$ is called the *ascending part* of the hill, and the remaining part $a_m, a_{m+1}, \ldots a_n$ is called the *descending part* of the hill. The element a_m is the *peak* of the hill and is the largest element in the array. Your task is to locate the peak (that is, a_m) in the hill-valued array $\mathcal{A}$ using a suitable algorithm. What is the time-complexity of your designed algorithm?

4. Recall the algorithm for *Median Finding* from an n-element array. You may have noticed that, in the initial phase of the algorithm, we split n element array into $\lfloor \frac{n}{5} \rfloor$ subparts each having 5-elements. Instead, let us do something different in this splitting at the beginning, and check the impact on the time-complexity of the same algorithm.

   First, your task is to analyze the time-complexity of this algorithm, when we split the elements as follows:

   (i) $\lfloor \frac{n}{3} \rfloor$ subparts each having 3-elements; *(Odd-split of < 5 elements)*

   (ii) $\lfloor \frac{n}{7} \rfloor$ subparts each having 7-elements; *(Odd-split of > 5 elements)*

   (iii) $\lfloor \frac{n}{4} \rfloor$ subparts each having 4-elements; and *(Even-split of < 5 elements)*

   (iv) $\lfloor \frac{n}{6} \rfloor$ subparts each having 6-elements. *(Even-split of > 5 elements)*

   In general, deduce the generic time-complexity recurrence relations and present the solution for the following cases:

   - $\lfloor \frac{n}{2k-1} \rfloor$ subparts each having (odd number of) $(2k-1)$-elements, for $k = \{1, 2, 3, \ldots\}$; and
   - $\lfloor \frac{n}{2k} \rfloor$ subparts each having (even number of) $2k$-elements, for $k = \{1, 2, 3, \ldots\}$.

   Doing this analysis, you may get a feel of the optimal position of splitting in the median-finding algorithm!

   So, what is your general observation? Formally explain (briefly).