ECE 606, Fall 2021, Assignment 5

Due: Tuesday, October 19, 11:59pm

<u>Note</u>: deadline has switched to Tuesday.

Submission: your solutions for the written problems to crowdmark. There are no **[python3]** problems in this assignment.

1. Suppose, given input a positive integer n, we want to compute $1^3 + 2^3 + \ldots + n^3$. Bob proposes the following algorithm that employs a kind of divide-n-conquer. It is first invoked as CubeSumBob$(1, n)$.

```
CubeSumBob($i, j$)
1 if $i = j$ then return $i \times i \times i$
2 else
3     $mid \leftarrow \left\lfloor \frac{i+j}{2} \right\rfloor$
4     return CubeSumBob($i, mid$) + CubeSumBob($mid + 1, j$)
```

   Alice, on the other hand, prefers to use the following algorithm with appears to adopt the incremental strategy. It is first invoked as CubeSumAlice(n).

```
CubeSumAlice($i$)
11 if $i = 1$ then return 1
12 else  return CubeSumAlice($i - 1$) + $i \times i \times i$
```

   In $\Theta(\cdot)$ notation, is one of the algorithms more time-efficient than the other?

2. Suppose, given a non-negative integer x and a positive integer y, we want to compute the quotient and remainder of x divided by y. Recall that both the quotient and remainder are unique, and the remainder is an integer in $[0, y)$.

   Suppose $\langle q, r\rangle$ denotes the quotient and remainder of x divided by y, and $\langle q', r'\rangle$ denotes the quotient and remainder of $\lfloor x/2 \rfloor$ divided by y. Consider the following recurrence for $\langle q, r\rangle$.

$$\langle q, r\rangle = \begin{cases} \langle 0, 0\rangle & \text{if } x = 0 \\ \langle 2q', 2r'\rangle & \text{if } x > 0,\ x \text{ is even and } 2r' < y \\ \langle 2q' + 1, 2r' - y\rangle & \text{if } x > 0,\ x \text{ is even and } 2r' \geq y \\ \langle 2q', 2r' + 1\rangle & \text{if } x > 0,\ x \text{ is odd and } 2r' + 1 < y \\ \langle 2q' + 1, 2r' + 1 - y\rangle & \text{otherwise} \end{cases}$$

   (a) Prove that the recurrence is correct.

   (b) Suppose we encode the recurrence as a recursive algorithm. As a function of the size of the input, assuming binary encoding, what is the worst-case running time of the algorithm in $\Theta(\cdot)$ notation?

3. Assume that Partition on pdf page 35 of Lecture 5 of your textbook is correct. That is, on input $\langle A, p, r\rangle$ with $r \geq p$, upon return from Partition, (i) every $A[p], A[p+1], \ldots, A[q-1] \leq A[q]$, (ii) every $A[q+1], \ldots, A[r] \geq A[q]$, and, (iii) the multiset $\{A[p], A[p+1], \ldots, A[r]\}$ on output is the same as the multiset $\{A[p], \ldots, A[r]\}$ on input.

Prove: the running-time of QUICKSORT from pdf page 35 of Lecture 5 of your textbook, as measured by the number of comparisons between array entries, is $\Omega(n^2)$ if the input array (i) has all distinct entries, and, (ii) is sorted in decreasing order.

4. Parts (b) and (c) of this problem pertain to the following algorithm, MYQUICKSORT. It invokes as subroutine the algorithm RANDOMIZED-PARTITION from pdf page 44 of Lecture 5 of your textbook.

```
MyQuickSort($A, p, r$)
1 while $p < r$ do
2     $q \leftarrow$ Randomized-Partition($A, p, r$)
3     if $q - p < r - q$ then
4         MyQuickSort($A, p, q - 1$)
5         $p \leftarrow q + 1$
6     else
7         MyQuickSort($A, q + 1, r$)
8         $r \leftarrow q - 1$
```

   (a) What is the worst-case depth of the call-stack for RANDOMIZED-QUICKSORT from pdf page 44 of Lecture 5 of your textbook in $\Theta(\cdot)$ notation as a function of n, where $n = r - p + 1$? n is num of elems in A

   (b) Assume that RANDOMIZED-PARTITION is correct. Prove: MYQUICKSORT is correct.

   (c) What is the worst-case depth of the call-stack for MYQUICKSORT in $\Theta(\cdot)$ notation as a function of n, where $n = r - p + 1$?

5. You have n bank cards each of which is associated with an account. You cannot read an account number directly off a card. However, you are provided an equivalence checker that given two cards, tells you whether they correspond to the same account. Devise an algorithm that tells us whether (strictly) more than half the cards correspond to the same account. Your algorithm should exercise the equivalence checker $O(n \lg n)$ times only, each time with two cards.

   (Hints: a running-time of $n \lg n$ is a huge hint that divide-n-conquer, similar to Merge sort, works. Given a set of n cards $S = \{c_1, \ldots, c_n\}$, suppose we split that into two sets, $S_{\text{left}} = \{c_1, \ldots, c_{\lfloor n/2 \rfloor}\}$ and $S_{\text{right}} = \{c_{\lfloor n/2 \rfloor + 1}, \ldots, c_n\}$. Now, there is an account number a that corresponds to more than half the cards in S only if the account number a corresponds to more than half the cards in at least one of S_{left} or S_{right}. If you can strengthen that "only if" into an "if and only if" then you would have an algorithm.)