Question 1

Consider non-negative integers represented in binary (i.e., the numbers are given as arrays of bits).

• How fast can you double an n-bit number?

• How fast can you add two n-bit numbers?
Question 1 (cont’d)

So doubling is faster than arbitrary addition.
Consider a similar question about multiplication:

Is squaring faster than multiplying?

More precisely, let

- $S(n)$ be the time to square an n-bit number, and
- $M(n)$ be the time to multiply two n-bit numbers.

Clearly $S(n) = O(M(n))$ (we can square by multiplying)

Is $M(n) = O(S(n))$?

If so, squaring is just as hard as multiplying; if not, it is easier.
Question 2

Consider the randomized selection algorithm.

Redefine the notion of "good splitter":

$s$ is a **good splitter** of a list $A$ if it is

- $\geq$ at least $1/8$ of the elements of $A$, and
- $\leq$ at least $1/8$ of the elements of $A$.

Under the new definition

- What proportion of elements of $A$ are good splitters?
- What is the maximum size of the subproblem solved recursively, if the chosen splitter is good?
Question 2 (cont’d)

So under the new definition
• splitters are more abundant (3/4 vs. 1/2), but
• the subproblem is larger (7/8 vs. 3/4)

Analyze the expected running time of the algorithm under this definition.
• Do we still get expected linear time?
• If so, is the constant we get the same or better?
Question 2 (cont’d)

• What is the expected number of trials to chose a good splitter?

• What is the expected number of steps taken by recursive calls in phase j? (Phase j is now defined to consist of the calls in which the size of the list is \( \leq (7/8)^j n \) and \( > (7/8)^{j+1} n \), where \( n \) is the size of the original list).

• What is the expected number of steps taken by the algorithm on inputs of size \( n \)?
Question 3

We say that an array \text{A}[1..n] of distinct integers is swap-sorted if moving the last \(n-k\) elements of \text{A} before the first \(k\) elements yields a sorted array, where \(k\) is an integer such that \(1 \leq k \leq n\).

E.g. \([7, 11, 13, 17, 19, 2, 3, 5]\) is swap-sorted (take \(k=3\))

Note that a sorted array of distinct integers is swap-sorted: take \(k=n\).

Describe an algorithm \text{Search(A,x)} that, given a swap-sorted array \text{A} of distinct integers and an integer \(x\), returns the index of \text{A} that contains \(x\), if such an index exists; and returns 0 otherwise.

Your algorithm should run in \(O(\log n)\) time.
Question 4

To think about at home:

Adapt the closest pair of points algorithm to work for points in 3-dimensional space:
Given a set P of n points in 3D, where each point is specified by its three coordinates $(x,y,z)$, find two distinct points $p$ and $q$ in $P$ whose distance is at most the distance of any two distinct points in $P$.
The (Euclidean) distance between $(x,y,z)$ and $(x',y',z')$ is the square root of $(x-x')^2+(y-y')^2+(z-z')^2$.
Your algorithm should run in $O(n\log n)$ time.