Please answer each of the following problems. You can discuss with your classmates, but make sure that you understand the solutions. Don't plagiarize! （可以使用中文回答问题。）

1. **COVID-19 Risk Detection.** (12 points) Each of n customers spends some time in a cafe shop. For each $i = 1, \ldots, n$, user i enters the shop at time a_i and leaves at time $b_i \geq a_i$. You are interested in the question: how many distinct pairs of customers are ever in the shop at the same time? (Here, the pair (i, j) is the same as the pair (j, i)).

   Example: Suppose there are 5 customers with the following entering and leaving times:

Customer	Enter time	Leave time
1	1	4
2	2	5
3	7	8
4	9	10
5	6	10

   Then, the number of distinct pairs of customers who are in the shop at the same time is three: these pairs are $(1, 2)$, $(4, 5)$, $(3, 5)$.

   (a) (3 pts) Suppose a customer c_i is reported as a suspected case, can you give an algorithm to retrieve all the potential inflicted customers in $O(n)$-time?

   (b) (4 pts) Given input $(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)$ as above, there is a straightforward algorithm that takes about[1] n^2 time to compute the number of pairs of customers who are ever in the shop at the same time. Give this algorithm and explain why it takes time about n^2.

   (c) (5 pts) Give an $O(n \log(n))$-time algorithm to do the same task and analyze its running time. (**Hint:** consider sorting relevant events by time).

2. **Proof of correctness.** (6 points) Consider the following Selection Sort algorithm that is supposed to sort an array of integers. Provide a proof that this algorithm is correct. (**Hint**: you may need to use more than one loop invariant.)

```
# Sorts an array of integers.
Sort(array A):
    for i = 1 to A.length:
        minIndex = i
        for j = i + 1 to A.length:
```

[1] Formally, "about" here means $\Theta(n^2)$, but you can be informal about this.

```
            if A[j] < A[minIndex]:
                minIndex = j
        Swap(A[i], A[minIndex])

# Swaps two elements of the array.  You may assume this function is correct.
Swap(array A, int x, int y):
    tmp = A[x]
    A[x] = A[y]
    A[y] = tmp
```

3. **Needlessly complicating the issue.** (12 points)

   (a) (3pts) Give a linear-time (that is, an $O(n)$-time) algorithm for finding the minimum of n values (which are not necessarily sorted).

   (b) (3pts) Argue that any algorithm that finds the minimum of n items must do at least n operations in the worst case.

   Now consider the following recursive algorithm to find the minimum of a set of n items.

Algorithm 1: `findMinimum`

Input: List $A = [a_1, \ldots, a_n]$ of n items

Output: $\min_i\{a_1, \ldots, a_n\}$

if n=*1* **then**

  return ________________

$A_1 = A[0 : n/2]$

$A_2 = A[n/2 : n]$

return min(`findMinimum`(A_1), `findMinimum`(A_2))

   (c) (3pts) Fill in the blank in the pseudo-code: what should the algorithm return in the base case? Briefly argue that the algorithm is correct with your choice.

   (d) (3pts) Analyze the running time of this recursive algorithm. How does it compare to your solution in part (a)?

4. **Recursive local-minimum-finding.** (12 points)

   (a) Suppose A is an array of n integers (for simplicity assume that all integers are distinct). A *local minimum* of A is an element that is smaller than all of its neighbors. For example, in the array $A = [1, 2, 0, 3]$, the local minima are $A[1] = 1$ and $A[3] = 0$.

      i. (2 points) Design a recursive algorithm to find a local minimum of A, which runs in time $O(\log(n))$.

      ii. (2 points) Prove formally that your algorithm is correct.

      iii. (2 points) Formally analyze the runtime of your algorithm.

   (b) Let G be a square $n \times n$ grid of integers. A *local minimum* of A is an element that is smaller than all of its neighbors (diagonals do not count). For example, in the grid

$$G = \begin{bmatrix} 5 & 6 & 3 \\ 6 & 1 & 4 \\ 3 & 2 & 3 \end{bmatrix}$$

   some of the local minima are G[1][1]=5 G[2][2] = 1.

      i. (2 points) Design a recursive algorithm to find a local minimum in $O(n)$ time (you can assume that all integers are distinct).

      ii. (2 points) We are not looking for a formal correctness proof, but please explain why your algorithm is correct.

      iii. (2 points) Give a formal analysis of the running time of your algorithm.

5. **Probability refresher** (4 points)

   (a) (1 point) What is the cardinality of the set of all subsets of $\{1, 2, ..., n\}$? **[We are expecting a mathematical expression along with one or two sentences explaining why it is correct.]**

   (b) (1 point) Suppose we choose a subset of $\{1, ..., n\}$ uniformly at random: that is, every set has an equal probability of being chosen. Let X be a random variable denoting the cardinality (that is, the size) of a set randomly chosen in this way. Calculate the expected value of X and show your work. **[We are expecting a mathematically rigorous argument establishing your answer. Your solution should not include summation signs.]**

   (c) (2 points) Let $rand(a, b)$ return an integer uniformly at random from the range $[a, b]$. Each call to $rand(a, b)$ is independent. Consider the following function:

```
f(k,n):
    if k <= 1:
        return rand(1,n)
    return 2 * f(k/2, n)
```

What is the expected value and variance of $f(k, n)$? Assume that k is a power of 2. Please show your work. [**In addition to your answer, we are expecting a mathematical derivation along with a brief (1-2 sentence) analysis of what the pseudocode above does in order to explain why your derivation is the right thing to do.**]

6. **Fun with Big-O notation.** (6 points; 1 point each) Mark the following as `True` or `False`. Briefly but convincingly justify all of your answers, using the definitions of $O(\cdot)$, $\Theta(\cdot)$ and $\Omega(\cdot)$. [**To see the level of detail we are expecting, the first question has been worked out for you.**]

   (x) $n = \Omega(n^2)$. This statement is `False`. To see this, we will use a proof by contradiction. Suppose that, as per the definition of $\Omega(\cdot)$, there is some n_0 and some $c > 0$ so that for all $n \geq n_0$, $n \geq c \cdot n^2$. Choose $n = \max\{1/c, n_0\} + 1$. Then $n \geq n_0$, but we have $n > 1/c$, which implies that $c \cdot n^2 > n$. This is a contradiction.

   (a) $n = O(n \log(n))$.

   (b) $n^{1/\log(n)} = \Theta(1)$.

   (c) If

   $$f(n) = \begin{cases} 5^n & \text{if } n < 2^{1000} \\ 2^{1000} n^2 & \text{if } n \geq 2^{1000} \end{cases}$$

   and $g(n) = \frac{n^2}{2^{1000}}$, then $f(n) = O(g(n))$.

   (d) For all possible functions $f(n), g(n) \geq 0$, if $f(n) = O(g(n))$, then $2^{f(n)} = O(2^{g(n)})$.

   (e) $5^{\log\log(n)} = O(\log(n)^2)$

   (f) $n = \Theta\left(100^{\log(n)}\right)$

7. **Fun with recurrences.** (6 points; 1 point each)

   Solve the following recurrence relations; i.e. express each one as $T(n) = O(f(n))$ for the tightest possible function $f(n)$, and give a short justification. Be aware that some parts might be slightly more involved than others. Unless otherwise stated, assume $T(1) = 1$. [**To see the level of detail expected, we have worked out the first one for you.**]

   (x) $T(n) = 6T(n/6) + 1$. We apply the master theorem with $a = b = 6$ and with $d = 0$. We have $a > b^d$, and so the running time is $O(n^{\log_6(6)}) = O(n)$.

   (a) $T(n) = 2T(n/2) + 3n$

   (b) $T(n) = 3T(n/4) + \sqrt{n}$

   (c) $T(n) = 7T(n/2) + \Theta(n^3)$

   (d) $T(n) = 4T(n/2) + n^2 \log n$

   (e) $T(n) = 2T(n/3) + n^c$, where $c \geq 1$ is a constant (that is, it doesn't depend on n).

   (f) $T(n) = 2T(\sqrt{n}) + 1$, where $T(2) = 1$