1. (a) False. For every n integers, for every permutation of the n integers, a correct implementation of randomized QUICKSORT can sort the n integers in $O(n \log n)$ time w.h.p.

   (b) True.

   (c) True.

   (d) False. By a reduction from LCS to LIS.

   (e) False. They both use $2(n^2 + n^2 - n)$ scalar operations.

2. (a) Observe that $\log n = o(n^{1/140}) = O(n^{1/140})$. Here is why.

$$\lim_{n\to\infty} \frac{\log n}{n^{1/140}} = \lim_{n\to\infty} \frac{140}{n^{1/140}} \\ = 0$$

   Combining 10 copies of the above equality, one has $\log^{10} n = O(n^{1/14})$. Hence, we have

$$\sqrt{n} \log^{10} n = O(n^{1/2+1/14}) = O(n^{4/7}).$$

   (b) By the definition of big-O, there exist constants C_1, C_2, n_0 so that for every $n \geq n_0$ we have

$$f(n) + g(n) \leq C_1 n^{4/7} + C_2 n^{2/3} \leq (C_1 + C_2) n^{2/3}.$$

   Hence, $f(n) + g(n) = O(n^{2/3})$.

   (c) By the definition of big-O, there exist constants C_1, C_2, n_0 so that for every $n \geq n_0$ we have

$$f(n) \cdot g(n) \leq C_1 C_2 n^{4/7+2/3} = C_1 C_2 n^{26/21}.$$

   Hence, $f(n) \cdot g(n) = O(n^{26/21})$.

3. (a) Observe that $5n^2 = O(n^{\log_2 5 - \epsilon})$. The first case of Master Theorem applies. We get $T(n) = O(n^{\log_2 5})$.

   (b) Observe that $3n^2 = \Omega(n^{\log_3 4 + \epsilon})$ and $4 \cdot 3(n/3)^2 \leq (4/9) \cdot 3n^2$. The third case of Master Theorem applies. We get $T(n) = O(n^2)$.

   (c) Let

$$T(n = 2^k) = S(k) = \begin{cases} S(\approx k/2) + 1 & \text{if } n \geq 1 \\ d & \text{if } n = 1 \end{cases}$$

   By Master Theorem, one has a rough guess that $T(n = 2^k) = S(k) = O(\log k) = O(\log \log n)$. We use the substitution method to prove that there exists some constant $c > 0$, for every $n \geq 1$, the guess $T(n) \leq c \log \log n + c$ holds.

   The induction base $n = 1$ holds by setting c sufficiently large (w.r.t. d). Assume that for every $n < t$, the guess holds. For $n = t$, $T(t) \leq c \log \log \lfloor \sqrt{n} \rfloor + 1 \leq c \log(1/2) + c \log \log n + 1 \leq c \log \log n + c$. By induction, $T(n) = O(\log \log n)$.

   (d) Let

$$S(n) = \begin{cases} 2S(\lfloor n/2 \rfloor + \lfloor \sqrt{n} \rfloor) + dn & \text{if } n > 1 \\ d & \text{if } n = 1 \end{cases}$$

   If d is picked as a sufficiently large constant, then $T(n) \leq S(n)$ for every $n \geq 1$.

We guess that the minor term $\sqrt{n}$ does not have any effect on the asymptotic order of $S(n)$. Hence, the second case of Master Theorem applies, and our guess implies $S(n) = O(n \log n)$.

We use the substitution method to prove that there exists some constant $c > 0$, for every $n \geq 1$, the guess $S(n) \leq cn \log n + c$ holds.

The induction base $n = 1, 2, \ldots, 4096$ holds by setting c sufficiently large (w.r.t. d). Assume that for every $n \in (4096, t)$, the guess holds. For $n = t$,

$$\begin{aligned} S(t) &\leq 2c(\lfloor t/2 \rfloor + \lfloor \sqrt{t} \rfloor) \log(\lfloor t/2 \rfloor + \lfloor \sqrt{t} \rfloor) + 2c + dt \\ &\leq ct \log(t/2 + \sqrt{t}) + 2c\sqrt{t} \log t + 2c + dt && \text{(because } t \geq 4096\text{)} \\ &\leq ct \log t - 0.95ct + 0.375ct + 2c + dt \\ &\leq ct \log t - 0.5ct + dt \\ &\leq ct \log t && \text{(if } c \geq 2d\text{)} \end{aligned}$$

By induction, $S(n) = O(n \log n)$, yielding that $T(n) = O(n \log n)$.

4. (a) Let $\alpha(P)$ denote the points that path P scores; that is, # 2-entries $-$ # 3-entries. Let $sol[x][y]$ denote $\max_P \alpha(P)$ among all monotonic paths from $A[1][1]$ to $A[x][y]$. Hence, if $sol[n][n]$ is positive, then some path visits more 2-entires than 3-entries as desired. Otherwise, no such a path exist. The pseudocode of our algorithm is given as follows, and the initial call is $\textsc{Find}(n, n, sol = \{-\infty\})$.

```
1  if sol[x][y] > −∞ then
2  |  return sol[x][y];
3  end
4  if count ≡ 2 (mod 13) then
5  |  count ← 1;
6  else
7  |  if count ≡ 3 (mod 13) then
8  |  |  count ← −1;
9  |  else
10 |  |  count ← 0;
11 |  end
12 end
13 if (x, y) equals (1, 1) then
14 |  return sol[x][y] = count;
15 end
16 if x equals 1 then
17 |  return sol[x][y] = count + FIND(x, y − 1, sol);
18 end
19 if y equals 1 then
20 |  return sol[x][y] = count + FIND(x − 1, y, sol);
21 end
22 return sol[x][y] = count + max{FIND(x − 1, y, sol), FIND(x, y − 1, sol)};
```

Algorithm 1: $\textsc{Find}(x, y, sol)$

It takes $O(1)$ time to fill in each entry in sol, and there are $O(n^2)$ entries in sol. The total running time is thus $O(n^2)$.

(b) The same as (a), but insert if$(count \equiv 5 \pmod{13})\{count \leftarrow -3n;\}$ right after Line 12.

5. (a) We prove this problem by reduction. In what follows, we devise an $o(n \log n)$-time algorithm for the element uniqueness problem by using $\mathcal{A}$ as a building block, assuming that $\mathcal{A}$ runs in $o(n \log n)$ time. However, any algorithm in the comparison-based model requires $\Omega(n \log n)$ time to solve the element uniqueness problem. Hence, any $\mathcal{A}$ in the comparison-based model requires $\Omega(n \log n)$ time.

Function `uniqueness` $(a_1, a_2, \ldots, a_n)$**:**
  $k \leftarrow$ `countFreq2` $(a_1, a_1, a_2, a_2, \ldots, a_n, a_n)$
  if k *equals* n **then**
    return $a_1, a_2, \ldots, a_n$ are all distinct
  else
    return Some of $a_1, a_2, \ldots, a_n$ repeats
  end

(b) Represent each a_i in base n, so each a_i has 3 n-ary digits. By RADIXSORT, one can sort a's in $O(3n)$ time. Followed by a linear scan, one can compute the frequency of a_i for each $i \in [1, n]$. Hence, one can output the number of values that appear exactly twice. In total, we use only $O(n)$ time.

6. Let $\mathrm{solve}(S, k)$ be a solver for the targeted problem with parameters S and k, so it returns a feasible R with the minimum cardinality. We assume that all elements in S are distinct, or break ties arbitrarily. We remove all negative elements from S because they cannot be included in R so as to minimize $|R|$. We also assume that $\mathrm{sum}(S) \geq k$, which can be checked in $O(n)$ time; otherwise, no such an R exist.

Function `solve` $(S = \{a_1, a_2, \ldots, a_n\},\ k)$**:**
  $\mu \leftarrow \mathrm{median}(a_1, a_2, \ldots, a_n)$
  $S_1 = \{a \in S : a \geq \mu\}$
  $S_2 = \{a \in S : a < \mu\}$
  if $\mathrm{sum}(S_1) \geq k$ **then**
    return $\mathrm{solve}(S_1, k)$
  else
    return $S_1 \cup \mathrm{solve}(S_2, k - \mathrm{sum}(S_1))$
  end

The running time is $cn + cn/2 + cn/4 + \cdots = O(n)$.