Solution to Written Assignment 2

1. (a) By the definition, there exist constants $n_0, c_1, c_2 > 0$ so that

$$0 \leq f(n) \leq c_1 h(n) \text{ and } 0 \leq g(n) \leq c_2 k(n) \text{ for every } n \geq n_0.$$

Combining the two inequalities, we get $0 \leq f(n) \cdot g(n) \leq c_1 c_2 h(n) k(n)$. Thus,

$$f(n) \cdot g(n) = O(h(n) \cdot k(n)).$$

(b) Observe that $\log n = o(n^{1/8}) = O(n^{1/8})$. Here is why.

$$\begin{aligned} \lim_{n \to \infty} \frac{\log n}{n^{1/8}} &= \lim_{n \to \infty} \frac{8}{n^{1/8}} \\ &= 0 \end{aligned}$$

Combining 8 copies of the above equality, one has $\log^8 n = O(n)$. Together with $f(n) = O(n)$, $f(n) + \log^8 n = O(n)$.

2. Let $\alpha(P)$ denote the points that path P scores. Let $sol[x][y]$ denote $\max_P \alpha(P)$ among all monotonic paths from $A[1][1]$ to $A[x][y]$. Hence, $sol[n][n]$ gives the solution. The pseudocode of our algorithm is given as follows, and the initial call is $\textsc{Find}(n, n, sol = \{-\infty\})$.

if $sol[x][y] > -\infty$ **then**
  return $sol[x][y]$;
end
$count \leftarrow 0$;
if $count \equiv 0 \pmod 3$ **then**
  $count \leftarrow count + 1$;
end
if $count \equiv 0 \pmod 5$ **then**
  $count \leftarrow count - 1$;
end
if (x, y) *equals* $(1, 1)$ **then**
  return $sol[x][y] = count$;
end
if x *equals* 1 **then**
  return $sol[x][y] = count + \textsc{Find}(x, y-1, sol)$;
end
if y *equals* 1 **then**
  return $sol[x][y] = count + \textsc{Find}(x-1, y, sol)$;
end
return $sol[x][y] = count + \max\{\textsc{Find}(x-1, y, sol), \textsc{Find}(x, y-1, sol)\}$;

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 $O(n^2)$.

3. For each $k \in [1, n]$, let $\ell[k]$ be the length of a longest increasing subsequence of $S[1..k]$ that contains $S[k]$. For each $k \in [1, n]$, let $r[k]$ be the length of a longest increasing subsequence of $S[n..k]$ that contains $S[k]$. By appealing to the $O(n \log n)$-time algorithm for LIS, we have $\ell[k], r[k]$ for every $k \in [1, n]$.

   To obtain a longest bitonic subsequence, in time linear in n one can find

$$u = \arg \max_{k \in [1,n]} \ell[k] + r[k] - 1.$$

   Given u, to find a longest bitonic subsequence, it is equivalent to compute an LIS of $S[1..u]$ that contains $S[u]$ and an LIS of $S[n..u]$ that contains $S[u]$. In total, $O(n \log n)$ time suffices.

4. For each $i \in [1, n]$, for each $j \in [1, n]$, define $c[i][j]$ to be the count of sets whose elements sum to i and whose maximum element is at most j. Thus we have

$$c[i][j] = \begin{cases} c[i][j-1] + c[i-j][j] & \text{if } j > 1 \\ 1 & \text{otherwise} \end{cases}$$

   To see why, the sets contribute to $c[i][j]$ can be classified into (1) sets that do not contain j and (2) sets that contain j. The number of sets in the former collection is exactly $c[i][j-1]$, and the number of sets in the latter collection is exactly $c[i-j][j]$ (remove one j from each set).

   The running time is $O(n^2)$ because there are $O(n^2)$ entries to fill and each entry can be calculated in $O(1)$ time.

5. We assume that the number of stones of weight 1 is at least m. If not, we add stones of weight 1, value 0 to the given set of stones. This can be done in $O(m)$ time.

   We assume that m is a multiple of 3. If not, say $m \equiv k \pmod 3$, then the k most valuable stones of weight 1 must be contained in the final output. These k stones can be pulled out in $O(2m)$ time.

   Then, for each stone s_i, if $w_i = 3$, we imagine to split s_i into three stones of weight 1, value $v_i/3$. By selection, one can find the m-th valuable stone in the imaginary set of stones. Followed by a linear scan, one can identify the set R of the m most valuable stones from the imaginary set of stones. If R does not contain any stone of weight 3 in part, then we are done. Otherwise, R contains exactly one stone of weight 3 in part, say $s_1, \ldots, s_k$ (note that $k \leq 2$). The optimal solution can be obtained by either replacing $s_1, \ldots, s_k$ with the k most valuable not-yet-used stones of weight 1, or replacing the $(3-k)$ least valuable used stones of weight 1 with the missing part of the incomplete stone. Pick one that yields the optiamlity.

   The total runtime is $O(m)$.

6. We prove this problem by reduction. In what follows, we devise an $o(n \log n)$-time algorithm for the element uniqueness problem using an $o(n \log n)$-time algorithm for the k-th mode. However, any algorithm in the comparison-based model requires $\Omega(n \log n)$ time to solve the element uniqueness problem. Hence, the $o(n \log n)$-time algorithm for the k-th mode does not exist in the comparison-based model.

Function `uniqueness` $(a_1, a_2, \ldots, a_n)$**:**
- $b \leftarrow \min\{a_1, a_2, \ldots, a_n\}$
- $\mu \leftarrow$ `kthMode` $(\underbrace{b-2, b-2, \ldots, b-k, b-k}_{2 \textit{ copies for each}}, a_1, a_2, \ldots, a_n)$
- **if** $\mathrm{freq}(\mu)$ *equals* 1 **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**

(a) Set $k = 3$.

(b) Set $k = \lfloor \log(n + 2k - 2) \rfloor$.