Analysis of Algorithms

Dynamic Programming

If more than one question appears correct, choose the more specific answer, unless otherwise instructed.

**Concept: memoization**

Assume zero-based indexing.

1. Consider memoizing this function:

   ```
   function f(x)
   {
     if (x == 0) return 0;
     if (x == 1) return 1;
     return f(x-2) + f(x-1);
   }
   ```

   What would be memoization table’s largest index/indices?

   (A) $x + 1$
   (B) $x$ and $x$
   (C) $x + 1$ and $x + 1$
   (D) $x$
   (E) $x - 1$
   (F) $x + 1$ and $x - 1$
   (G) $x + 1$ and $x$
   (H) $x - 2$

2. Consider memoizing this function:

   ```
   function f(x)
   {
     if (x == 0) return 0;
     if (x == 1) return 1;
     return f(x-2) + f(x-1);
   }
   ```

   In order to remove the two original base cases, how would the memoization table be initialized? Assume the memoization table is named `memo`.

   (A) `memo[0] = 1; memo[1] = 1;`
   (B) `memo[1] = 1;`
   (C) `memo[0] = 0; memo[1] = 1;`
   (D) `memo[0] = 0;`

3. Consider memoizing this function:

   ```
   function f(x)
   {
     if (x == 0) return 0;
     if (x == 1) return 1;
     return f(x-2) + f(x-1);
   }
   ```

   What base case would replace the two original base cases, assuming a properly initialized memoization table (named `memo`)?

   (A) `if (memo[x] != EMPTY) return memo[x];`
   (B) `if (memo[x] != EMPTY) return 0;`
   (C) `if (memo[x] != EMPTY) return x;`
   (D) `if (memo[x] != EMPTY) return 1;`
4. Consider memoizing this function:

```javascript
function g(x, items, y) {
    if (x == 0) return 1;
    if (x < 0) return 0;
    if (y == items.size) return 0;
    return minimum(g(x-items[y], items, y), g(x, items, y+1));
}
```

Assuming all three original base cases are retained and the smallest memoization table possible, What would be the memoization table’s largest index/indices, where \( x \) refers to the original value of \( x \)?

(A) \( x \) and \( \text{items.size} \)
(B) \( x + 1 \)
(C) \( x \)
(D) \( x - 1 \)
(E) \( x \) and \( \text{items.size} - 1 \)
(F) \( x + 1 \) and \( \text{items.size} + 1 \)
(G) \( x \) and \( \text{items.size} + 1 \)
(H) \( x - 2 \)

5. Consider memoizing this function:

```javascript
function g(x, items, y) {
    if (x == 0) return 1; // first base case
    if (x < 0) return 0; // second base case
    if (y == items.size) return 0; // third base case
    return minimum(g(x-items[y], items, y), g(x, items, y+1));
}
```

Which of the original base cases cannot be removed, given no knowledge of the values in `items`?

(A) the second
(B) the third
(C) the first and second
(D) the first
(E) the second and third

6. Consider memoizing this function:

```javascript
function g(x, items, y) {
    if (x == 0) return 1; // first base case
    if (x < 0) return 0; // second base case
    if (y == items.size) return 0; // third base case
    return minimum(g(x-items[y], items, y), g(x, items, y+1));
}
```

Removing all possible base cases, what would be the memoization table’s largest index/indices? Assume no knowledge of the values in `items`.

(A) \( x + 1 \)
(B) \( x \) and \( \text{items.size} - 1 \)
(C) \( x + 1 \) and \( \text{items.size} \)
(D) \( x \) and \( \text{items.size} \)
(E) \( x + 1 \) and \( \text{items.size} + 1 \)
(F) \( x - 1 \)
(G) \( x - 2 \)
(H) \( x \)
Dynamic programming

7. Consider using dynamic programming to improve the efficiency of the following function:

```c
function f(a, b, c, d, e)
{
    if (a == 0) return 0;
    if (a < 0) return -INFINITY;
    if (d == e) return 0;
    return
    max(
        f(a, b, c, d+1, e),
        f(a-b[d], b, c, d, e) + c[d]
    );
}
```

What would be the dimensionality of the dynamic programming table?

(A) 1   (D) 6
(B) 4   (E) 3
(C) 5   (F) 2

8. Consider using dynamic programming to improve the efficiency of the following function:

```c
function f(a, b, c, d, e)
{
    if (a == 0) return 0;
    if (a < 0) return -INFINITY;
    if (d == e) return 0;
    return
    max(
        f(a, b, c, d+1, e),
        f(a-b[d], b, c, d, e) + c[d]
    );
}
```

How would the dynamic programming table be filled, using `a` as an index?

(A) larger `a` to smaller `a`   (C) `a` is not used as an index
(B) smaller `a` to larger `a`

9. Consider using dynamic programming to improve the efficiency of the following function:

```c
function f(a, b, c, d, e)
{
    if (a == 0) return 0;
    if (a < 0) return -INFINITY;
    if (d == e) return 0;
    return
    max(
        f(a, b, c, d+1, e),
        f(a-b[d], b, c, d, e) + c[d]
    );
}
```

How would the dynamic programming table be filled, using `b` as an index?

(A) larger `b` to smaller `b`   (C) smaller `b` to larger `b`
(B) `b` is not used as an index
10. Consider using dynamic programming to improve the efficiency of the following function:

```javascript
function f(a, b, c, d, e)
{
    if (a == 0) return 0;
    if (a < 0) return -INFINITY;
    if (d == e) return 0;
    return
        max(
            f(a, b, c, d+1, e),
            f(a-b[d], b, c, d, e) + c[d]
        );
}
```

How would the dynamic programming table be filled, using `d` as an index?

(A) smaller `d` to larger `d`
(B) larger `d` to smaller `d`
(C) `d` is not used as an index

11. Consider using dynamic programming to improve the efficiency of the following function:

```javascript
function f(a, b, c, d, e)
{
    if (a == 0) return 0;
    if (a < 0) return -INFINITY;
    if (d == e) return 0;
    return
        max(
            f(a, b, c, d+1, e),
            f(a-b[d], b, c, d, e) + c[d]
        );
}
```

What is wrong, if anything, about the following loop for filling out the dynamic programming table?

```javascript
for (a = 0; a < max_a; ++a)
    for (d = 0; d < max_d; ++d)
    {
        if (a == 0) table[a][d] = 0;
        else if (d == e) t[a][d] = 0;
        else
        {
            var x = a-b[d];
            t[a][d] =
                max(
                    t[a][d+1],
                    x < 0? -INFINITY : (t[x][d] + c[d])
                );
        }
    }
```

(A) the `a` loop goes in the wrong direction
(B) there should be three nested loops
(C) one or more of the loop indices is incorrect
(D) the `d` loop goes in the wrong direction
(E) the table is filled out correctly
(F) there should only be one loop (no nesting)