COMP 3711 – Design and Analysis of Algorithms
2023 Fall Semester – Written Assignment Solution # 1
Distributed: 9:00 on September 15, 2023 Due: 23:59 on September 29, 2023

Solution 1: [28 pts]

(a) $A = \Omega(B)$;

(b) $A = \Theta(B)$;

(c) $A = O(B)$;

(d) $A = \Omega(B)$;

(e) $A = O(B)$;

(f) $A = \Omega(B)$;

(g) $A = \Omega(B)$;

Solution 2: [18 pts]

(a)

$$T(1) = 1; T(n) = 4T(n/2) + n^2 \text{ for } n > 1$$

Using the expansion method, expand the recurrence until a pattern emerges:

$$T(n/2) = 4T(n/2^2) + (n/2)^2$$

Plug that back into the original equation:

$$T(n) = 4(4T(n/2^2) + (n/2)^2) + n^2$$

$$T(n) = 4^2T(n/2^2) + 4(n/2)^2 + n^2$$

$$T(n) = 4^2T(n/2^2) + 2^2 * n^2 * 2^{-2} + n^2$$

$$T(n) = 4^2T(n/2^2) + 2n^2$$

Continue expansion:

$$T(n/2^2) = 4T(n/2^3) + (n/2^2)^2$$

Plug that back into the original equation:

$$T(n) = 4^2(4T(n/2^3) + (n/2^2)^2) + 2n^2$$

$$T(n) = 4^3T(n/2^3) + 4^2(n/2^2)^2) + 2n^2$$

$$T(n) = 4^3T(n/2^3) + 2^4 * n^2 * 2^{-4} + 2n^2$$

$$T(n) = 4^3T(n/2^3) + 2^4 * n^2 * 2^{-4} + 2n^2$$

$$T(n) = 4^3T(n/2^3) + 3n^2$$

The pattern is:

$$T(n) = 4^hT(n/2^h) + hn^2$$

Where h is the value when we hit the boundary condition.

$$n/2^h = 1 \implies h = \log_2 n$$

$$T(n) = 4^{\log_2 n}T(n/2^{\log_2 n}) + \log_2 n * n^2$$

$$T(n) = n^2 + n^2 \log_2 n$$

$$T(n) = O(n^2 \log n)$$

(b)

$$T(1) = 1; T(n) = 16T(n/4) + n \text{ for } n > 1$$

Using the expansion method, expand the recurrence until a pattern emerges:

$$T(n/4) = 16T(n/4^2) + n/4$$

Plug that back into the original equation:

$$T(n) = 16(16T(n/4^2) + n/4) + n$$

$$T(n) = 16^2T(n/4^2) + 4n + n$$

Continue expansion:

$$T(n/4^2) = 16T(n/4^3) + n/4^2$$

Plug that back into the original equation:

$$T(n) = 16^2(16T(n/4^3) + n/4^2) + 4n + n$$

$$T(n) = 16^3T(n/4^3) + 16^2n/4^2 + 4n + n$$

$$T(n) = 16^3T(n/4^3) + 16n + 4n + n$$

The pattern is:

$$T(n) = 16^hT(n/4^h) + \sum_{i=0}^{h-1} 4^i n$$

The geometric series sum can be rewritten using the formula:

$$a(\frac{1 - r^{n+1}}{1 - r})$$

$$T(n) = 16^hT(n/4^h) + n * (\frac{1 - 4^h}{1 - 4})$$

Where h is the value when we hit the boundary condition.

$$n/4^h = 1 \implies h = \log_4 n$$

$$T(n) = 16^{\log_4 n}T(n/4^h) + n * (\frac{1 - 4^h}{1 - 4})$$

$$T(n) = n^2 + n * (\frac{1 - n}{1 - 4})$$

$$T(n) = O(n^2)$$

(c)

$$T(2) = 1; T(n) = T(\sqrt{n}) + 1 \text{ for } n > 2$$

Using the expansion method, expand the recurrence until a pattern emerges:

$$T(n^{\frac{1}{2}}) = T(n^{\frac{1}{2^2}}) + 1$$

Plug that back into the original equation:

$$T(n) = T(n^{\frac{1}{2^2}}) + 1 + 1$$

$$T(n) = T(n^{\frac{1}{2^2}}) + 2$$

Continue expansion:

$$T(n^{\frac{1}{2^2}}) = T(n^{\frac{1}{2^3}}) + 1$$

Plug that back into the original equation:

$$T(n) = T(n^{\frac{1}{2^3}}) + 3$$

The pattern is:

$$T(n) = T(n^{\frac{1}{2^h}}) + h$$

Where h is the value when we hit the boundary condition.

$$n^{\frac{1}{2^h}} = 2$$

$$\log_2 n^{\frac{1}{2^h}} = \log_2 2$$

$$\frac{1}{2^h} \log_2 n = 1$$

$$\log_2 n = 2^h$$

$$\log_2 \log_2 n = \log_2 2^h$$

$$\log_2 \log_2 n = h$$

$$h = \log_2 \log_2 n$$

$$T(n) = T(n^{\frac{1}{2^{\log_2 \log_2 n}}}) + \log_2 \log_2 n$$

$$T(n) = O(\log \log n)$$

Solution 3: [24 pts]

We will begin my defining an algorithm that generates all possible binary arrays of size n. Let us call this algorithm generateBinaryArrays(n). The basic idea is to generate all $n-1$ arrays and add 0 to the end of each, and then generate $n-1$ arrays and add 1 to the end of each.

Algorithm 1 Generate all possible binary arrays of size n

```
function generateBinaryArrays($n$)
    if $n == 1$ then
        return $[[0], [1]]$                                          ▷ Base Case
    end if
    $previous\_arrays\_0 \leftarrow generateBinaryArrays(n-1)$          ▷ T(n-1)
    for previous_array in previous_arrays_0  do
        $previous\_array.append(0)$
    end for
    $previous\_arrays\_1 \leftarrow generateBinaryArrays(n-1)$          ▷ T(n-1)
    for previous_array in previous_arrays_1  do
        $previous\_array.append(1)$
    end for
    return $previous\_arrays\_0 + previous\_arrays\_1$
end function
```

We will show that the running time of $T(n)$ is upper-bounded by the number of possible binary arrays 2^n, that is $T(n) = O(2^n)$.

The recurrence for the following algorithm is $T(n) = 2T(n-1) + 1$

$$T(1) = 1; T(n) = 2T(n-1) + 1 \text{ for } n > 1$$

Using the expansion method, expand the recurrence until a pattern emerges:

$$T(n-1) = 2T(n-2) + 1$$

Plug that back into the original equation:

$$T(n) = 2(2T(n-2) + 1) + 1$$

$$T(n) = 2^2T(n-2) + 2 + 1$$

Continue expansion:

$$T(n-2) = 2T(n-3) + 1$$

Plug that back into the original equation:

$$T(n) = 2^2(2T(n-3) + 1) + 2 + 1$$

$$T(n) = 2^3T(n-3) + 2^2 + 2 + 1$$

The pattern is:

$$T(n) = 2^hT(n-h) + \sum_{i=0}^{h-1} 2^i n$$

The geometric series sum can be rewritten using the formula:

$$a(\frac{1-r^{n+1}}{1-r})$$

$$T(n) = 2^h T(n-h) + (\frac{1-2^h}{1-2})$$

$$T(n) = 2^h T(n-h) + 2^h - 1$$

Where h is the value when we hit the boundary condition.

$$n - h = 1 \implies h = n - 1$$

$$T(n) = 2^{n-1} + 2^{n-1} - 1$$

$$T(n) = 2 * 2^{n-1} - 1$$

$$T(n) = 2^n - 1$$

$$T(n) = O(2^n)$$

Now we can leverage this algorithm to write the algorithm to generate all possible $n \times n$ binary matrices. Let us call this algorithm generateBinaryMatricies(n). The basic idea is to generate all $n-1 \times n-1$ matrices, using our generateBinaryArrays algorithm, generate all binary arrays of size n that will be used as columns to extend the previous matrices and using generateBinaryArrays generate all binary arrays of size $n-1$ that will be used as rows to extend the previous matrices.

Algorithm 2 Generate all possible binary matricies of size $n \times n$

```
function GENERATEBINARYMATRICIES($n$)
    if $n == 1$ then
        return $[[[0]], [[1]]]$                                          ▷ Base Case
    end if
    $previous\_matricies \leftarrow generateBinaryMatricies(n-1)$          ▷ T(n-1)
    $output \leftarrow []$
    for previous_matrix in previous_matricies do
        $rows \leftarrow generateBinaryArrays(n-1)$                        ▷ $O(2^{n-1})$
        for row in rows do
            $cols \leftarrow generateBinaryArrays(n)$                      ▷ $O(2^n)$
            for col in cols do
                $new\_matrix \leftarrow add\_row\_and\_column\_to\_matrix(matrix, row, col)$
                $output.append(new\_matrix)$
            end for
        end for
    end for
    return $output$
end function
```

We will show that the running time of $T(n)$ is upper-bounded by the number of possible 2D binary arrays 2^{n^2}, that is $T(n) = O(2^{n^2})$.

The recurrence for the following algorithm is $T(n) = T(n-1) * 2^n * 2^{n-1}$

$$T(1) = 1; T(n) = 2^{2n-1}T(n-1) \text{ for } n > 1$$

$$T(n) = 2^{2n-1}T(n-1)$$

Using the expansion method, expand the recurrence until a pattern emerges:

$$T(n-1) = 2^{2(n-1)-1}T(n-2)$$

Plug that back into the original equation:

$$T(n) = 2^{2n-1}(2^{2(n-1)-1}T(n-2))$$

$$T(n) = 2^{4n-4}T(n-2)$$

Continue expansion:

$$T(n-2) = 2^{2(n-2)-1}T(n-3)$$

Plug that back into the original equation:

$$T(n) = 2^{4n-4}(2^{2(n-2)-1}T(n-3))$$

$$T(n) = 2^{6n-9}T(n-3)$$

The pattern is:

$$T(n) = 2^{2hn-h^2}T(n-h)$$

$$n - h = 1 \implies h = n-1$$

$$T(n) = 2^{2*(n-1)*n-(n-1)^2}$$

$$T(n) = 2^{2*n^2-2n-n^2+2n-1}$$

$$T(n) = 2^{n^2-1}$$

$$T(n) = O(2^{n^2})$$

Solution 4: [30 pts]

(a) At most 9 10-major items can co-exist in an array. To prove this, suppose some array with n elements has 10-major items $w_1, w_2, ...w_t,$ *where* $t > 9$. Then the number of items in the array that take on one of those t values is:

$$> t\frac{n}{10} \geq 10\frac{n}{10} = n$$

leading to a contradiction. Thus $t \leq 9$. Next, let $n = 36$ and for $i = 1, 2, ..., 9$ set the value in the array as

$$A[4i-3] = A[4i-2] = A[4i-1] = A[4i] = i$$

Then $1, 2, 3..., 9$ are all 10-major items in $A[]$ so 9 is possible.

(b) Here is the general idea. It is a direct generalization of the majority algorithm from the tutorial. If $n = 1$ then the single item in the array is a 10-major item. If $n > 1$ then split $A[]$ into two sub-arrays of (almost) equal size. Recursively find the up to 9 solutions items in each subarray. Store those up to 18 items as candidate 10-major items. It is also important to remove any potential duplicates. Do at most 18 linear scans through A checking if any of those items are 10-majority items. Return those (if any), which are 10-majority. You may find the pseudocode at the end of the document. To receive full credit, you needed to explicitly:

(a) Note that the total number of 10-major candidates that needed to be checked is ≤ 18. This was important because without bounding this number, you can't prove that the conquer step requires only $O(n)$ time.

(b) Remove duplicates. This was important because without removing duplicates, the number of candidates returned can grow too large. Depending upon how you wrote your code, this can either result in losing track of some candidate 10-major items or increasing the size of the conquer step above $\Theta(n)$.

(c) The recurrence for the algorithm is:

$$T(1) = 1; T(n) = 2T(\frac{n}{2}) + O(18n) = 2T(\frac{n}{2}) + n \text{ for } n > 1$$

(d) Using the expansion method, expand the recurrence until a pattern emerges:

$$T(\frac{n}{2}) = 2T(\frac{n}{2^2}) + \frac{n}{2}$$

Plug that back into the original equation:

$$T(n) = 2(2T(\frac{n}{2^2}) + \frac{n}{2}) + n$$

$$T(n) = 2^2T(\frac{n}{2^2}) + 2 * \frac{n}{2}) + n$$

Continue expansion:

$$T(\frac{n}{2^2}) = 2T(\frac{n}{2^3}) + \frac{n}{2^2}$$

Plug that back into the original equation:

$$T(n) = 2^2(2T(\frac{n}{2^3}) + \frac{n}{2^2}) + 2 * \frac{n}{2}) + n$$

$$T(n) = 2^3T(\frac{n}{2^3}) + 2^2\frac{n}{2^2}) + 2 * \frac{n}{2}) + n$$

$$T(n) = 2^3T(\frac{n}{2^3}) + 3n$$

The pattern is:

$$T(n) = 2^hT(\frac{n}{2^h}) + hn$$

$$\frac{n}{2^h} = 1 \implies h = \log_2 n$$

$$T(n) = 2^h * T(1) + hn$$

$$T(n) = 2^{\log_2 n} * T(1) + n\log_2 n$$

$$T(n) = n + n\log_2 n$$

$$T(n) = O(n\log n)$$

Algorithm 3 Find all the 10-major items in an array

```
function MAJORITY($A, s, t$)
    $n \leftarrow A.size$
    if $n == 1$ then
        return $[A[0]]$                                        ▷ Base Case
    end if
    $m \leftarrow \frac{s+t}{2}$
    $e_1 \leftarrow majority(A, s, m)$                          ▷ $T(\frac{n}{2})$
    $e_2 \leftarrow majority(A, m+1, t)$                        ▷ $T(\frac{n}{2})$
    $output \leftarrow []$
    $seen \leftarrow Set()$
    for candidate in $e_1$ and $e_2$ do                         ▷ $O(18n)$
        if $candidate$ in $seen$ then
            $continue$
        end if
        $seen.insert(candidate)$
        $freq \leftarrow 0$
        for $element$ in $A$ do                                 ▷ $O(n)$
            if $element == candidate$ then
                $freq += 1$
            end if
        end for
        if $freq > \frac{n}{10}$ then
            $output.append(candidate)$
        end if
    end for
    return $output$
end function
```