A1 Q2-4

Q2. a

Lines	Complexity
(5)	$\theta(1)$
(4) - (6)	$\theta(n+3)$
(3) - (7)	$\theta\left(\sum_{k=1}^{n+3}(n+3)\right)$ *or* $\theta(n^2+9+6n)$
(8) - (10)	$\theta(i^3-3+1)$ *or* $\theta(i^3-2)$
(3) - (10)	$\theta(n^2+6n+i^3+7)$
(2) - (11)	$\theta\left(n^2+6n+\sum_{j=1}^{i} i^3+7\right)$ *or* $\theta(n^2+6n+7+i^4)$
(1) - (12)	$\sum_{i=1}^{n}\theta(n^2+6n+7+i^4)$ *or* $\theta\left(\frac{1}{30}\begin{array}{c}(6n^5+15n^4+40n^3\\+180n^2+209n)\end{array}\right)$ or $\boldsymbol{\theta(n^5)}$

b

Lines	Complexity
(1), (3), (5), (8)	$\theta(1)$
(3) - (7)	$j=n, n-n^{\frac{1}{3}}, n-2n^{\frac{1}{3}}, \ldots, n-(n^{\frac{2}{3}}-1)*n^{\frac{1}{3}}$ $n^{\frac{2}{3}}$ *iterations or* $\theta(n^{\frac{2}{3}})$
(1) - (9)	$i=n^3, \frac{n^3}{3}, \frac{n^3}{9} \ldots, 1$ *or* $(\log_3 n^3+1)$ *iterations* $\theta\left(n^{\frac{2}{3}}*3*\log_3 n\right)$ *or* $\theta\left(n^{\frac{2}{3}}*\log n/\log 3\right)$ *or*

	$\boldsymbol{\theta\left(n^{\frac{2}{3}} * log\, n\right)}$

Q3.

Algorithm

```
1   L: [l_1,l_2,…,l_k]                       // individually sorted
2   n: total number of elements in all lists
3   k: total number of lists
4   returns sorted merged list
5   function mergeLists(L, n, k) {
6     newList = []
7     for i = 0 to n {
8       for j = 0 to k {
9         if L[j][0] < nextElem {
10          nextElem = L[j].shift()  // pop first element
11        }
12      }
13
14      newList.push(nextElem)
15    }
16    return newList
17  }
```

Correctness

Invariant for loop on i:

```
newList[i] <= newList[j]    where 0 <= i < j < n
L[m] is sorted              for all 0 <= m < k
```

Analysis

Lines	Complexity
(6), (9)-(11), (14), (16)	$\theta(1)$
(8) - (12) for loop on j	$\theta(k)$
(5) - (17) for loop on i	$\theta\left(\sum_{i=0}^{n} k\right) or$ $\theta(nk)$

Divide and Conquer

```
L: [l_1,l_2,…,l_k]                    // k sorted lists
returns: A sorted merged list
function mergeLists(L, k) {
  if (L.len == 1) {
    return L[0]
  } else {
    mid = floor(k / 2)
    L1 = mergeLists(L[0..mid])
    L2 = mergeLists(L[(mid+1)..k])
    return mergeSortedLists(L1, L2)
  }
}

L1, L2: Two sorted lists
returns: A merged sorted list
runtime: Θ(max(L1.len, L2.len))
function mergeTwoSortedLists(L1, L2) {
  newList = []
  i, j = 0, 0
  while (i < L1.len && j < L2.len) {
    if (L1[i] < L2[j]) {
      newList.push(L1[i])
      i++
    } else {
      newList.push(L2[j])
      j++
    }
  }
  // Add leftover elements
  newList.pushElements(L1[i..(L1.len)])
  newList.pushElements(L2[j..(L2.len)])
  return newList
}
```

Q4.

```
grid: 2D, sorted left-right & top-bottom
m: # of rows in grid
n: # of cols in grid
num: To check the existence of in the grid
returns: true if num exists in grid, false if not
function numExistsInGrid(grid, m, n, num) {
```

```
7   i, j = m-1, 0            // start at bottom left
8   while (i >= 0 && j < n) {
9     if (num == grid[i][j])
10      return true
11    else if (num > grid[i][j])
12      j++
13    else
14      i--
15    end
16  }
17  return false
18 }
```

Correctness

If we start at the bottom left of the grid,

- If `num` is equal to grid cell, we found its existence
- If `num` is greater than the grid cell, we go one column right as that's where we will find bigger numbers (and we have already excluded numbers below)
- If `num` is less than the grid cell, we go one row up as numbers in the current row on the right are greater than `num` (and we have already excluded numbers on the left)

Analysis

Lines	Complexity
(7), (9)-(15), (17)	$\theta(1)$
(6) - (18) while loop on i and j	$\theta(m+n)$