Hw 2

Sang Hyeong Woo
Sangwoo

1.

1) Sum = 0;
for (i = 0; i < n; i++)
  Sum ++;

Running Time = C + Cn
= O(N)

∴ Big-Oh = O(N)

2) Sum = 0;
for (i = 0; i < n; i++) n times
  for (j = 0; j < n; j++) n times
    Sum++;

Running Time = C + Cn²
= O(N²)

∴ Big-Oh = O(N²)

3) Sum = 0;
for (i = 0; i < n; i++) n times
  for (j = 0; j < n × n; j++) n² times
    Sum++;

Running Time = C + C(n × n²)
= C + C(n³)
= O(n³)

∴ Big-Oh = O(N³)

4) Sum = 0;
for (i = 0; i < n; i++) n times
  for (j = 0; j < i; j++) $\frac{n}{2}$ times
    Sum++;

Running Time = C + $\frac{C}{2}n^2$
= O(n²)

∴ Big-Oh = O(N²)

5) Sum = 0;
for (i = 0; i < n; i++) n times
  for (j = 0; j < i × i; j++) $\frac{n^2}{2}$ times
    for (k = 0; k < j; k++) $\frac{n^2}{4}$ times
      Sum++;

Running Time = C + $(n \times \frac{n^2}{2} \times \frac{n^2}{4})C$
= C + $\frac{n^5}{8}C$
= O(N⁵)

∴ Big-Oh = O(N⁵)

2. $f(x) = \sum_{i=0}^{N} a_i x^i$

```
Poly = 0;
for (i = n; i >= 0; i--)
    Poly = x * Poly + a[i]
```

a) $x = 3$, $f(x) = 4x^4 + 8x^3 + x + 2$

iteration	i	poly	a[i]	Poly
1	4	0	4	$3 \times 0 + 4 = 4$
2	3	4	8	$3 \times 4 + 8 = 20$
3	2	20	0	$3 \times 20 + 0 = 60$
4	1	60	1	$3 \times 60 + 1 = 181$
5	0	181	2	$3 \times 181 + 2 = 545$

$f(3) = 4 \times (3)^4 + 8 \times (3)^3 + 3 + 2$

$= 4 \times 81 + 8 \times 27 + 3 + 2$

$= 324 + 216 + 3 + 2$

$= 545$

∴ $f(x)$ when $x = 3$ is 545.

c) The running time of this algorithm is $O(N)$ as the algorithm only has one for-loop.

Big-Oh = $O(N)$

3. c) Worst Case, Big-Oh = $O(N^2)$

```
38      throw new NoMajorityException();
39    }
40
41    /**
42     * Returns an array of 0, 1, or 2 candidates for majority element in
43     * a. Implement the recursive algorithm described in the main
44     * comment.
45     */
46    public static int[] findCandidates(int[] a) {
47      if (a.length == 0 || a.length == 1 || a.length == 2) {
48          return a;
49      }
50
51      else {
52          int[] b = new int[] {};
53
54          for (int i = 0; i < a.length - 2; i += 2) {
55              if (a[i] == a[i + 1]) {
56                  b = Arrays.copyOf(b, b.length + 1);
57                  b[b.length - 1] = a[i];
58              }
59
60          }
61          if (a.length % 2 == 1) {
62              b = Arrays.copyOf(b, b.length + 1);
63              b[b.length - 1] = a[a.length - 1];
64          }
65
66          return findCandidates(b);
67      }
68    }
69
70    /**
71     * Returns true iff x appears in more than half of the elements of a.
72     */
```

```java
public class MatrixSearch {

  /**
   * Returns true iff val is in the n x n array a. Assume that a is
   * arranged so that the elements in every row are in increasing
   * order from left to right, and the elements in every col are in
   * increasing order from top to bottom. The worst-case running time
   * of this method must be O(n).
   */
  public static boolean contains(int val, int[][] a) {
      int row = 0;
      int col = a.length - 1;

      while (row < a.length && col >= 0) {
          if (a[row][col] == val) {
              return true;
          }
          else if (a[row][col] > val) {
              col--;
          }
          else {
              row++;
          }
      }
      return false;
  }

  /**
   * Run some tests.
   */
  public static void main(String... args) {
    int[][] a;
```