**Big-Oh Analysis**

Give a tight bound of the runtime complexity class for each of the following code fragments in Big-Oh notation, in terms of the variable *N*.

Source: <https://courses.cs.washington.edu/courses/cse373/13wi/exams/midterm-practice-1.pdf>

**Problems**:

|  |
| --- |
| a)  int sum = 0;  for (int i = 1; i <= N + 3; i++) {  for (int j = 1; j <= N \* N; j++) {  sum++;  }  } sum++;  System.out.println(sum); |
| b)  int sum = 0;  for (int i = 1; i <= N - 3; i++) {  for (int j = 1; j <= i + 4; j += 2) {  sum++;  }  sum++;  }  for (int i = 1; i <= 100; i++) {  } sum++;  System.out.println(sum); |
| c)  int sum = 0;  for (int i = 1; i <= N; i++) {  for (int j = 1; j <= 1000; j += 2) {  sum++;  }  }  for (int k = -50; k <= -1; k++) {  } sum++;  System.out.println(sum); |
| d)  Set<Integer> set = new TreeSet<Integer>();  for (int i = 1; i <= N \* 2; i++) {  set.add(i);  }  for (int k : set) {  } System.out.println(k);  System.out.println("done!"); |
| e)  List<Integer> list =  new ArrayList<Integer>();  for (int i = 1; i <= N; i++) {  list.add(i);  }  while (!list.isEmpty()) {  } list.remove(0);  System.out.println("done!"); |
| f)  Deque<Integer> deque =  new ArrayDeque<Integer>();  for (int i = 1; i <= N / 2; i++) {  } deque.addLast(i);  for (int i = 1; i <= N / 2; i++) {  deque.addFirst(i);  }  while (!deque.isEmpty()) {  deque.removeFirst();  } deque.removeLast();  System.out.println("done!"); |

**Solutions**:

Source: <https://courses.cs.washington.edu/courses/cse373/13wi/exams/midterm-practice-1-key.pdf>

|  |
| --- |
| a)  **Original:**  int sum = 0;  for (int i = 1; i <= N + 3; i++) {  for (int j = 1; j <= N \* N; j++) {  sum++;  }  } sum++;  System.out.println(sum);  **Runtime:**  O(1)  for O(N + 3)  for O(N^2)  O(1)  O(1)  O(1)  **Answer:**  = O(1) + O( N (N^2(1) + 1) )  = O( N (N^2 + 1) )  = O(N^3 + N)  **= O(N^3)** |
| b)  **Original:**  int sum = 0;  for (int i = 1; i <= N - 3; i++) {  for (int j = 1; j <= i + 4; j += 2) {  sum++;  }  sum++;  }  for (int i = 1; i <= 100; i++) {  } sum++;  System.out.println(sum);  **Runtime:**  O(1)  for O(N)  for O(N)  O(1)  O(1)  for O(1)  O(1)  O(1)  **Answer:**  = O(1 + N(N(1) + 1)) + O(1) + O(1)  = O(N^2 + N)  **= O(N^2)** |
| c)  **Original:**  int sum = 0;  for (int i = 1; i <= N; i++) {  for (int j = 1; j <= 1000; j += 2) {  sum++;  }  }  for (int k = -50; k <= -1; k++) {  } sum++;  System.out.println(sum);  **Runtime:**  O(1)  for O(N)  for (1)  O(1)  for O(1)  O(1)  O(1)  **Answer:**  = O(1 + N(1(1))) + O(1(1)) + O(1)  **= O(N)** |
| d)  **Original:**  Set<Integer> set = new TreeSet<Integer>();  for (int i = 1; i <= N \* 2; i++) {  set.add(i);  }  for (int k : set) {  } System.out.println(k);  System.out.println("done!");  **Runtime:**  O(1)  for O(2N)  O(logN)  for (N)  O(1)  O(1)  **Answer:**  = O(1 + N(logN)) + O(N(1)) + O(1)  = O(NlogN) + O(N)  **= O(NlogN)** |
| e)  **Original:**  List<Integer> list = new ArrayList<Integer>();  for (int i = 1; i <= N; i++) {  list.add(i);  }  while (!list.isEmpty()) {  } list.remove(0);  System.out.println("done!");  **Runtime:**  O(1)  for O(N)  O(1)  while O(N)  } O(N)  O(1)  **Answer:**  = O(1 + N(1)) + O(N(N)) + O(1)  = O(N) + O(N^2)  **= O(N^2)** |
| f)  **Original:**  Deque<Integer> deque = new ArrayDeque<Integer>();  for (int i = 1; i <= N / 2; i++) {  } deque.addLast(i);  for (int i = 1; i <= N / 2; i++) {  deque.addFirst(i);  }  while (!deque.isEmpty()) {  deque.removeFirst();  } deque.removeLast();  System.out.println("done!");  **Runtime:**  O(1)  for O(N/2)  O(1)  for O(N/2)  O(1)  while O(N/2)  O(1)  } O(1)  O(1)  **Answer:**  = O(N/2) + O(N/2) + O(N(1 + 1))  = O(N + N)  **= O(N)** |