**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-2.pdf>

**Problems**:

|  |
| --- |
| a)  int sum = 0;  for (int i = 0; i < N \* 2; i++) {  for (int j = 0; j < i / 3; j++) {  for (int k = 0; k < j\*j; k+=3) {  } } } sum++;  System.out.println(sum); |
| b)  int sum = 0;  for (int i = 1; i < N; i \*= 2) {  for (int j = 1; j < N; j \*= 2) {  } } sum++;  System.out.println(sum); |
| c)  Set<Integer> set1 =  new HashSet<Integer>();  for (int i = 0; i < N; i++) {  } set1.add(i);  Set<Integer> set2 =  new TreeSet<Integer>();  set2.addAll(set1);  System.out.println("done!"); |
| d)  List<Integer> list =  new LinkedList<Integer>();  for (int i = 0; i < N; i++) {  } list.add(0, i);  Set<Integer> set = new TreeSet<Integer>();  Iterator<Integer> itr = list.iterator();  while (itr.hasNext()) {  } set.add(itr.next());  System.out.println("done!"); |
| e)  List<Integer> list1 =  new ArrayList<Integer>();  for (int i = 0; i < N; i += 2) {  } list1.add(i);  List<Integer> list2 =  new ArrayList<Integer>();  for (int i = 0; i < N; i++) {  } list2.add(0, list1.remove(0));  System.out.println("done!"); |
| f)  int sum = 0;  for (int i = 0; i < N \* 2; i++) {  for (int j = 0; j < 10000; j++) {  for (int k = 0; k < j\*j; k++) {  } } } sum++;  System.out.println(sum); |

**Solutions**:

Source: <https://courses.cs.washington.edu/courses/cse373/13wi/exams/midterm-practice-2-key.pdf>

|  |
| --- |
| a)  **Original:**  int sum = 0;  for (int i = 0; i < N \* 2; i++) {  for (int j = 0; j < i / 3; j++) {  for (int k = 0; k < j\*j; k+=3) {  } } } sum++;  System.out.println(sum);  **Runtime:**  O(1)  for O(N)  for O(N)  for O(N^2)  O(1)  O(1)  **Answer:**  = O(1 + N(N(N^2(1))) + 1)  = O(1 + N(N^3) + 1)  **= O(N^4)** |
| b)  **Original:**  int sum = 0;  for (int i = 1; i < N; i \*= 2) {  for (int j = 1; j < N; j \*= 2) {  } } sum++;  System.out.println(sum);  **Runtime:**  O(1)  for O(logN)  for O(logN)  O(1)  O(1)  **Answer:**  = O(1 + logN(logN(1)) + 1)  = O(1 + logN(logN) + 1)  = O(1 + (logN)^2 + 1)  **= O((logN)^2))** |
| c)  **Original:**  Set<Integer> set1 = new HashSet<Integer>();  for (int i = 0; i < N; i++) {  } set1.add(i);  Set<Integer> set2 = new TreeSet<Integer>();  set2.addAll(set1);  System.out.println("done!");  **Runtime:**  O(1)  for O(N)  O(1)  O(1)  O(NlogN)  O(1)  **Answer:**  = O(1 + N + 1) + O(1 + NlogN + 1)  = O(N) + O(NlogN)  **= O(NlogN)** |
| d)  **Original:**  List<Integer> list = new LinkedList<Integer>();  for (int i = 0; i < N; i++) {  } list.add(0, i);  Set<Integer> set = new TreeSet<Integer>();  Iterator<Integer> itr = list.iterator();  while (itr.hasNext()) {  } set.add(itr.next());  System.out.println("done!");  **Runtime:**  O(1)  for O(N)  O(1)  O(1)  O(1)  while O(N)  O(logN) + O(1);  O(1)  **Answer:**  = O(1 + N(1)) + O(1 + 1 + N(logN + 1) + 1)  = O(N) + O(NlogN + N)  **= O(NlogN)** |
| e)  **Original:**  List<Integer> list1 = new ArrayList<Integer>();  for (int i = 0; i < N; i += 2) {  } list1.add(i);  List<Integer> list2 = new ArrayList<Integer>();  for (int i = 0; i < N; i++) {  } list2.add(0, list1.remove(0));  System.out.println("done!");  **Runtime:**  O(1)  for O(N / 2)  O(1)  O(1)  for O(N)  O(N) + O(N)  O(1)  **Answer:**  = O(1 + N/2(1)) + O(1 + N(N + N) + 1)  = O(N/2) + O(N^2 + N^2)  **= O(N^2)** |
| f)  **Original:**  int sum = 0;  for (int i = 0; i < N \* 2; i++) {  for (int j = 0; j < 10000; j++) {  for (int k = 0; k < j\*j; k++) {  } } } sum++;  System.out.println(sum);  **Runtime:**  O(1)  for O(2N)  for O(1)  for O(1)  O(1)  O(1)  **Answer:**  = O(1 + 2N(1(1(1))) + 1)  = O(2N)  **= O(N)** |