**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-13wi.pdf>

**Problems**:

|  |
| --- |
| a)  int sum = 0;  for (int i = 1; i <= N + N; i++) {  } sum++;  for (int j = 1; j <= N \* 2; j++) {  } sum += 5;  System.out.println(sum); |
| b)  int sum = 0;  for (int i = 1; i <= N - 5; i++) {  for (int j = 1; j <= N - 5; j = j \* 2) {  } } sum++;  System.out.println(sum); |
| c)  int sum = N;  for (int i = 0; i < 1000000; i++) {  for (int j = 1; j <= i; j++) {  } sum += N;  for (int j = 1; j <= i; j++) {  } sum += N;  for (int j = 1; j <= i; j++) {  } } sum += N;  System.out.println(sum); |
| d)  List<Integer> list = new ArrayList<Integer>();  for (int i = 1; i <= N; i++) {  for (int j = 1; j <= N; j++) {  list.add(0, i + j);  }  }  int count = 0;  for (int i = 1; i <= 2 \* N; i++) {  if (list.contains(i)) {  count++;  }  }  System.out.println("done!"); |
| e)  Set<Integer> set1 = new HashSet<Integer>();  for (int i = 1; i <= N; i++) {  } set1.add(i);  Set<Integer> set2 = new TreeSet<Integer>();  for (int i = 1; i <= N; i++) {  set1.remove(i);  } set2.add(i + N);  System.out.println("done!"); |

**Solutions**:

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

|  |
| --- |
| a)  **Original:**  int sum = 0;  for (int i = 1; i <= N + N; i++) {  } sum++;  for (int j = 1; j <= N \* 2; j++) {  } sum += 5;  System.out.println(sum);  **Runtime:**  O(1)  for O(N + N)  O(1)  for O(2N)  O(1)  O(1)  **Answer:**  = O(1) + O(N+ N) + O(N) + O(1)  = O(N + N) + O(N)  **= O(N)** |
| b)  **Original:**  int sum = 0;  for (int i = 1; i <= N - 5; i++) {  for (int j = 1; j <= N - 5; j = j \* 2) {  } } sum++;  System.out.println(sum);  **Runtime:**  O(1)  for O(N)  for O(logN)  O(1)  O(1)  **Answer:**  = O(1) + O(N(logN(1))) + O(1)  **= O(NlogN)** |
| c)  **Original:**  int sum = N;  for (int i = 0; i < 1000000; i++) {  for (int j = 1; j <= i; j++) {  } sum += N;  for (int j = 1; j <= i; j++) {  } sum += N;  for (int j = 1; j <= i; j++) {  } } sum += N;  System.out.println(sum);  **Runtime:**  O(1)  for O(1)  for O(1)  O(1)  for O(1)  O(1)  for O(1)  O(1)  O(1)  **Answer:**  **= O(1)** |
| d)  **Original:**  List<Integer> list = new ArrayList<Integer>();  for (int i = 1; i <= N; i++) {  for (int j = 1; j <= N; j++) {  list.add(0, i + j);  }  }  int count = 0;  for (int i = 1; i <= 2 \* N; i++) {  if (list.contains(i)) {  count++;  }  }  System.out.println("done!");  **Runtime:**  O(1)  for O(N)  for O(N)  O(N)  O(1)  for O(N)  O(N)  O(1)  O(1)  **Answer:**  = O(N(N(N))) + O(N(N + 1))  = O(N^3) + O(N^2 + N)  **= O(N^3)** |
| e)  **Original:**  Set<Integer> set1 = new HashSet<Integer>();  for (int i = 1; i <= N; i++) {  } set1.add(i);  Set<Integer> set2 = new TreeSet<Integer>();  for (int i = 1; i <= N; i++) {  set1.remove(i);  } set2.add(i + N);  System.out.println("done!");  **Runtime:**  O(1)  for O(N)  O(1)  O(1)  for O(N)  O(1)  O(logN)  O(1)  **Answer:**  = O(1) + O(N(1)) + O(1) + O(N(1 + logN)) + O(1)  = O(N) + O(N + NlogN)  **= O(NlogN)** |