Project 1: Asymptotic



Analysis – Time Complexity

# Project Options

**What is the time complexity of this algorithm,** **in terms of n?**

|  |  |  |  |
| --- | --- | --- | --- |
| Option | Question | Option | Question |
| 0 | int j = 2  while (j < n) {  int k = j  while (k < n) {  Sum += a[j]\*b[k]  k = k \* k  }  j = 2 \* j  } | **5** | int j = 5  while (j < n/2) {  int k = 5  while (k < n) {  Sum += a[j]\*b[k]  k = k \* sqrt(2)  }  j = sqrt(3) \* j  } |
| Solution | First, for the outer loop, it goes until j reaches n (or higher). The looped time t1 satisfies:  Thus 2 \* (2^(t1 - 1)) >= n  Then t1 = O(logn)  Similarly, for inner loop, t2 satisfies: 2^(2^t2) >= n  Then t2 = O(loglogn)  Thus the time complexity is  **T(n) = O(lognloglogn)** | **Solution** | First, for the outer loop, it goes until j reaches n/2 (or higher). The looped time t1 satisfies:  Thus 5 \* (sqrt(2) ^ (t1 - 1)) >= n/2  Then t1 = O(logn)  Similarly, for inner loop, t2 satisfies:  5 \* (sqrt(3) ^ (t2 - 1)) >= n/2  Then t2 = O(logn)  Thus the time complexity is  **T(n) = O((logn)2)** |
| 1 | for (int i = 1 to n) {  for (int j = i to n) {  for (int k = j to n) {  Sum += a[i]\*b[j]\*c[k]  }  If (gcd(i,j) == 1) {  j = n  }  }  } | **6** | int j = 2  while (j < n) {  int k = j  while (k < n) {  Sum += a[k]\*b[k]  k += n1/3 log n  }  j = j\*2  } |
| Solution | Normally, these three loops runs O(n) respectively, making the time complexity O(n2).  However, consider the second loop, where there is  If (gcd(i,j) == 1) {  j = n  }  Then, since there is gcd(i,j) == 1, the loop ends. Since the GCD of any two consecutive integer is 1, the second loop would always ends when j == i + 1. Thus, the second loop runs O(1) times only.  As a result, the time complexity is  **T(n) = O(n2)** | **Solution** | First, for the outer loop, it goes until j reaches n (or higher). The looped time t1 satisfies:  Thus 2 \* (2 ^ (t1 - 1)) >= n  Then t1 = O(logn)  Similarly, for inner loop, t2 satisfies:  2 + n1/3 log n \* (t2 - 1) >= n  Then t2 = n2/3/logn  Thus the time complexity is  **T(n) = O((n2/3/logn)\*logn)= O(n2/3)** |
| 2 | int j = 2 while (j < n) {  k = 2  while (k < n) {  Sum += a[k]\*b[k]  k = k \* sqrt(k)  }  j += j/2  } | **7** | for (int i = 1 to n) {  for (int j = i to n) {  for (int k = j\*j to n) {  Sum += a[i]\*b[j]\*c[k]  }  }  } |
| Solution | First, for the outer loop, it goes until j reaches n (or higher). The looped time t1 satisfies:  Thus 2\*(3/2)t1>= n  Then t1 = O(logn)  Similarly, for inner loop, t2 satisfies: 2^(3/2^t2) >= n  Then t2 = O(loglogn)  Thus the time complexity is  **T(n) = O(lognloglogn)** | **Solution** | The outer loop takes O(n)  The second loop takes O(n)  The inner loop takes O(√n)  However, for the inner loop, it takes O(1) when l reaches greater than √n, Thus, the inner two loops together takes  O(√n)\*O(√n)+O(n-√n)\*O(1) = O(n)  With the outer loop takes O(n), the time complexity is T(n)= O(n2)  Also, the running time could be calculated as follows  T(n)=n(√(n-1))√n-((√(n-1))√n)2  **T(n) = O(n2)** |
| 3 | for (int i = 1 to n) {  for (int j = i to n) {  for (int k = j to n) {  Sum += a[i]\*b[j]\*c[k]  }  If (j == 2\*i) {  j = n  }  }  } | **8** | int j = 5  while (j < log n) {  int k = 5  while (k < n) {  Sum += a[j]\*b[k]  k = k^1.5  }  j = 1.2 \* j  } |
| Solution | Normally, these three loops runs O(n) respectively, making the time complexity O(n3).  However, consider the second loop, where there is  If (j == 2\*i) {  j = n  }  Then, since there is j == 2\*i, the loop ends. There are two situations:   1. When i<=n/2: the second loop will end at n/2 (for the worst case) or less. As we are only care the upper bound. The time complexity will be n\*(n/c)\*(n/c)=O(n^3) 2. When i>n/2: O(n^3) as usual   As a result, the time complexity is  **T(n) = O(n3)** | **Solution** | First, for the outer loop, it goes until j reaches n/2 (or higher). The looped time t1 satisfies:  Thus 5 \* (1.2 ^ (t1 - 1)) >= logn  Then t1 = O(loglogn)  Similarly, for inner loop, t2 satisfies:  5 ^ (1.5 ^ (t2 - 1)) >= n  Then t2 = O(loglogn)  Thus the time complexity is  **T(n) = O((loglogn)2)** |
| 4 | for (int i = 1 to n) {  j = i  while (j < n) {  k = j  while (k < n) {  Sum += a[i]\*b[j]\*c[k]  k += log log n  }  j += log (j+10)  }  } | **9** | j = 2 while (j < n) {  k = j  while (k < n) {  Sum += a[k]\*b[k]  k = k \* k  }  j += log k  } |
| Solution | Obviously, the outer loop runs O(n).  The second loop, the time t1 satisfies:  For the second loop, we can expand the terms of sum as:  t1 = log(j+10) + log(log(j+10)+10) + log(log(log(j+10)+10)+10)+..  Since O(j)>>O(logj)  We can estimate that t1 = O(n/logn)  For the inner loop, t2 satisfies  1 + loglogn \* (t2 - 1) >= n  t2 = n/loglogn  As a result, the time complexity is  **T(n) = O(n3/(lognloglogn))** | **Solution** | First, when we get to outer loop, k > n (that is the only reason we could be in outer loop.  Then j+= log k, can be written as j+= log n.  for the outer loop, it goes until j reaches n (or higher). The looped time t1 satisfies:  Thus 2 + logk \* t1 >= n  Since logk = logn after the assignment  Then t1 = O(n/logn)  Similarly, for inner loop, t2 satisfies: 2^(2^t2) >= n  Then t2 = O(loglogn)  Thus the time complexity is  **T(n) = O(nloglogn/logn)** |