1. Give a big-O bound for the number of bits needed to represent a number N

* D = Floor[log(n)+1]

1. Give a big-O bound for the bit-complexity of multiplying two integers between 1 and N. What is the worst case? Note, the bound should be simply in terms of N.

* Lecture notes say the following:

Worst case:

110101

111111

1. Given that , consider sequentially taking each number

and computing . Stop when either i) and you have found the square root, or ii) and N does not have an integer square root. Give a big-O bound for the overall complexity of this search in terms of N.

* It takes to find the square (Based on lecture notes).
* You would need to do it times.

1. We are effectively … Use this idea as the basis of a binary search-type algorithm. Give a big-O bound on the overall complexity of this search in terms of N. How does it compare to the previous?

* It takes to find the square (Based on lecture notes).
* Once we have N -> O(1), we can bound the set of integers, allowing for binary search
* Time complexity of binary search is log(n)

1. Given an integer k > 1, give a big-O bound for the bit-complexity of computing the k-th power of an integer between 1 and N. What is the worst case? Note that the bound should be in terms of N and k.

* to find

1. Similar to Question 3, 4 above, given that , … Use this idea as the basis of a binary search-type algorithm. Give a big-O bound on the overall complexity of this search, in terms of N and k.

* times

1. Questions 5, 6 above assumed that k was known. What if k is unknown? If N is a perfect (non-trivial) power, what is the smallest possible value of k that power might be? What is the largest? Give a big-O bound on the largest possible k.

* Smallest: 0
* Largest: ln(k)
* You must do the above log(k)-times, we may not use binary search because we need to test all possible values.

1. Consider repeating the algorithm in Question 6, for every k over the indicated range above, to search all possible powers to see if N = a k for some a, k. For clarity, write out the pseudocode of this algorithm given an input N. Give a big-O bound on the overall complexity of this search, in terms of N. Is this efficient?

* See if N = 0 or 1
* See if then up to log(k)-numbers
* Perform binary search on , then check all numbers of k again.
* No, it’s brute forcing since we do not know what values of k are relevant.

**Bonus:**

1. I don’t believe we can use a binary search to find k, since there are many possible solutions.
2. Consider only prime values of k, , for a composite , where p is prime.
3. O(log(n))