**Collaborators :** None

**Sources** : Lecture Notes; <https://www.geeksforgeeks.org/check-if-a-given-number-is-a-perfect-square-using-binary-search/>

**Q4**   **Perfect Powers**

**(a)** Give pseudocode for an algorithm that takes as input a positive integer and determines whether or not is a perfect square. If it is, your algorithm should also determine the number such that . If is binary digits long, your algorithm should take steps, where is the number of steps required to multiply two numbers of at most binary digits. Thus, your algorithm should take with the usual grade school multiplication algorithm; or, it would be steps with the sophisticated Schönhage–Strassen multiplication algorithm. (Hint: binary search.)

**Ans:** To determine whether a number is a perfect square, we can use **binary search** on the range from to . We search for an integer such that . The binary search ensures that the algorithm runs efficiently in with grade-school multiplication, or faster with advanced algorithms like Schönhage–Strassen.

**Pseudocode:**

*isPerfectSquare(A): // Returns (Boolean(Square or not),B)*

*1. If A == 0 or A == 1:*

*return (True, A)*

*2. Set low = 1, high = A*

*3. While low ≤ high:*

*a. Set mid = (low + high) // 2*

*b. Set square = mid \* mid*

*c. If square == A:*

*return (True, mid)*

*d. If square < A:*

*Set low = mid + 1*

*e. Else:*

*Set high = mid - 1*

*4. Return (False, None)*

**Explanation:**

* We initialize two pointers, , to the range This is the search space to find square-root.
* Each step involves checking the square of the middle value () and comparing it to .
* If , then is a perfect square and we return .
* If , we know the square root must be greater, so we update as we don’t consider the search space below as their square roots will obviously be below
* If , the square root must be smaller, so we update as we don’t consider the search space above as their square roots will obviously be above

**Time Complexity:**

* **Binary Search**: The binary search runs in steps, which is since has bits.
* **Multiplication**: Each multiplication of -bit numbers (to compute ) takes steps with the grade-school multiplication algorithm or with the Schönhage–Strassen algorithm.
* **Total**: The overall complexity with binary search and multiplication is using grade-school multiplication or with Schönhage–Strassen.

**(b**) Give pseudocode for an algorithm that takes as input a positive integer and determines whether or not is a perfect power (i.e., a perfect square, cube, fourth power, etc.). If it is, your algorithm should also determine numbers and such that . When is an -bit number, justify that your algorithm takes at most steps for some constant (such as ).

**Ans:** To determine whether is a perfect power (i.e., if there are integers and such that ), we can iterate over possible values of and use binary search to find . The main idea is to try different values of and check whether is a perfect -th power.

**Pseudocode:**

*isPerfectPower(A): // Returns (Boolean(Power or not),B,C)*

*1. If A == 1, return (True, 1, any C > 1)*

*2. For C = 2 to ⌊log2(A)⌋:*

*a. Set low = 1, high = A*

*b. While low ≤ high:*

*i. Set mid = (low + high) // 2*

*ii. Set power = mid^C*

*iii. If power == A, return (True, mid, C)*

*iv. If power < A, set low = mid + 1*

*v. Else, set high = mid - 1*

*3. Return (False, None, None)*

**Explanation:**

* The outer loop iterates over possible values of from up to. (As )
* For each , we use binary search to find a base such that .
* In the inner loop, we perform binary search on in the range from , checking whether .
* If such a is found, the algorithm returns and .

**Time Complexity:**

* **Outer loop**: There are possible values of , where is the number of bits in .
* **Binary search for** : For each , we perform binary search on , which takes iterations, each involving an exponentiation.
* **Exponentiation**: The computation of can be done using fast exponentiation, which takes steps with binary exponentiation and for each multiplication. So, worst-case exponentiation costs time complexity. This results in a time complexity per iteration.
* **Total**: The total time complexity is with grade-school multiplication, or with more advanced multiplication algorithms.
* Thus, asfor , the algorithm doesn’t take more than steps to check if is a perfect power.