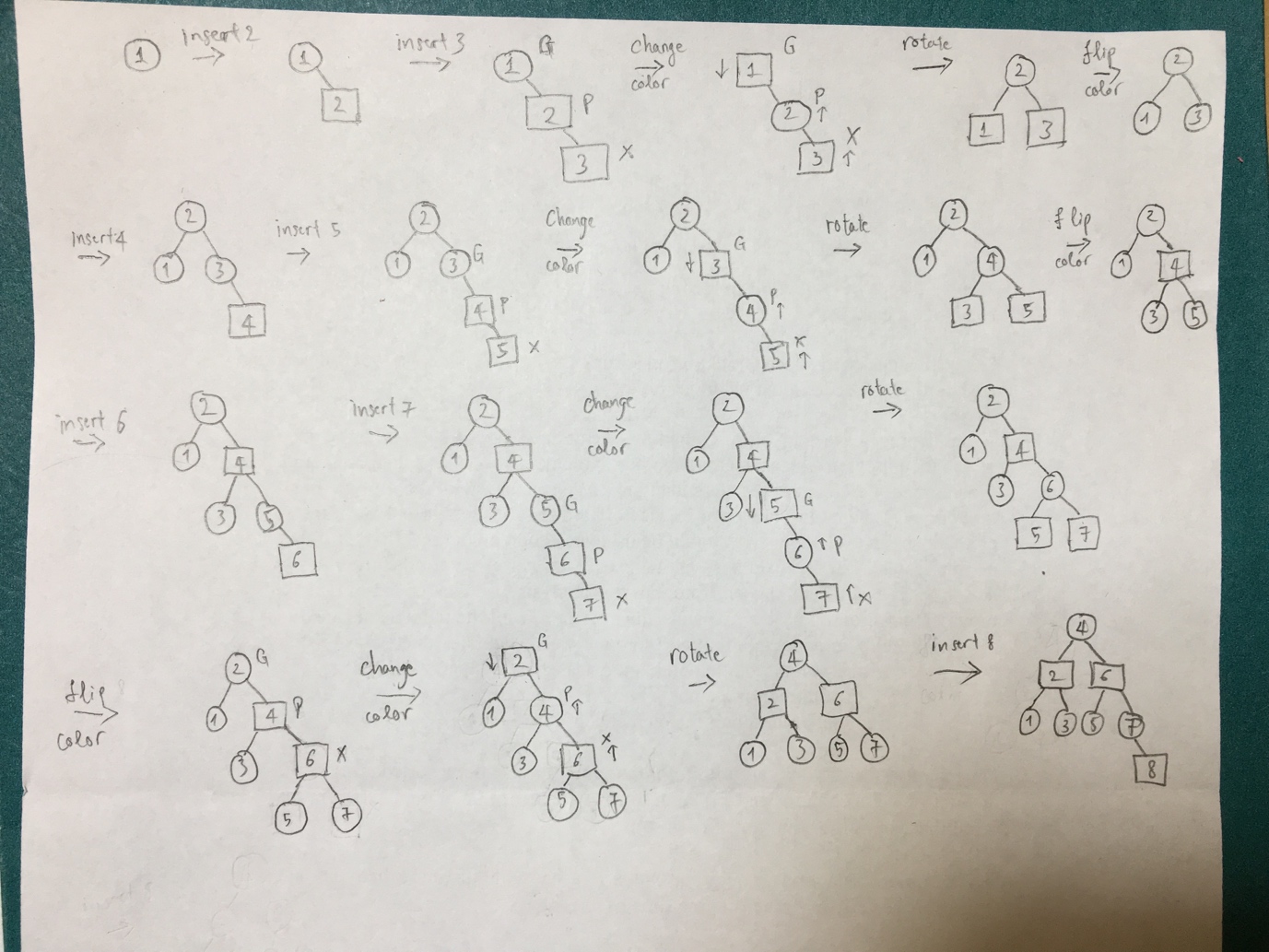
Lab 8

# Problem 1

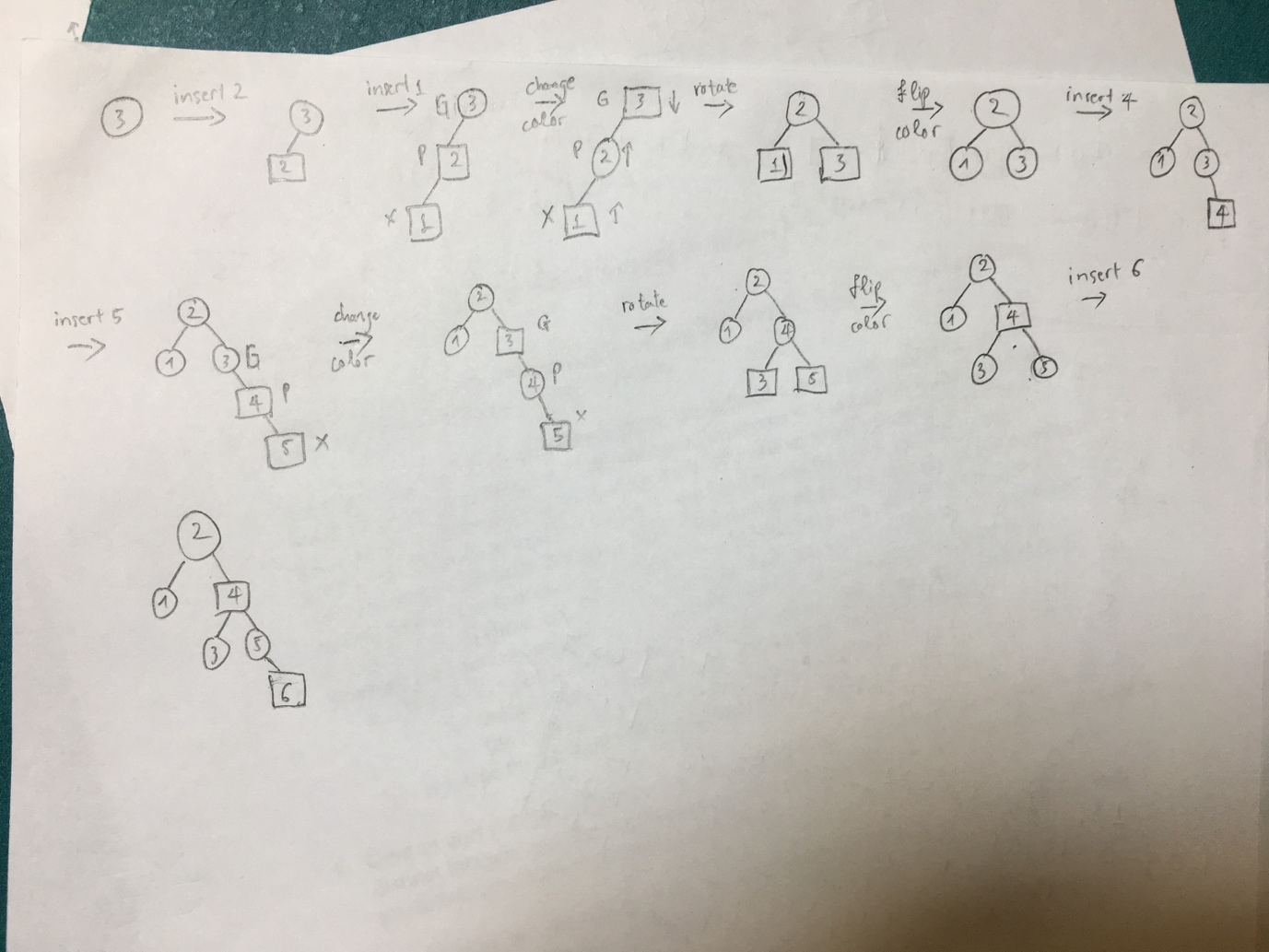
Algorithm reverse(S)  
 Input A String S  
 Output A String with each element is reverted  
  
 stack <- new Stack  
 for i <- 0 to S.length - 1 do  
 stack.push(S.charAt(i))  
  
 reverseStr <- ""  
  
 while !stack.isEmpty() do  
 reverseStr <- reverseStr + stack.pop()  
  
 reverseStrArr = reverseStr.split(" ")  
  
 result <- ""  
  
 for i <- reverseStrArr.length - 1 to 0 do  
 result <- result + reverseStrArr[i] + " "  
  
 return result

# Problem 2

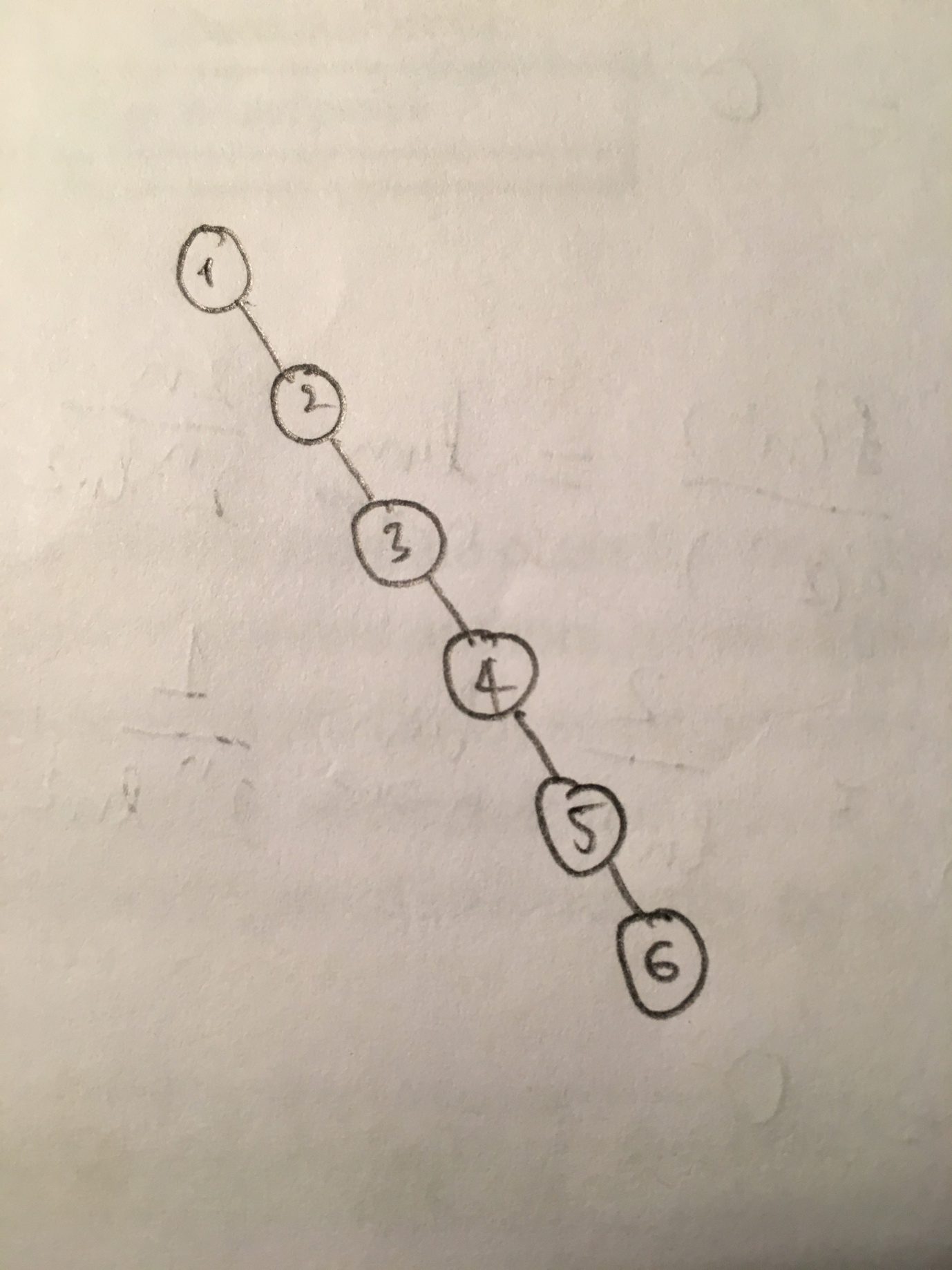
A



B



If we store data in BST with sorted array like part A we will get the worst case like image below. The Red Black tree handle sorted array is better.



# Problem 3

Algorithm

Algorithm  
 Input An integer n  
 Output True if n is prime otherwise 0  
  
 if n < 2 then  
 return false  
 if n = 2 then  
 return true  
 if n % 2 = 0  
 return false  
  
 for i <- 3 to sqrt(n) do  
 if n % i = 0 then  
 return false  
 i <- i + 2  
  
 return true

Java implementation

public static boolean isPrime(int n) {  
 if (n < 2) {  
 return false;  
 }  
  
 if (n == 2) {  
 return true;  
 }  
  
 if (n % 2 == 0) {  
 return false;  
 }  
  
 for (int i = 3; i <= Math.*sqrt*(n); i+=2) {  
 if (n % i == 0) {  
 return false;  
 }  
 }  
  
 return true;  
}

Since it takes √i steps in the for loop to check whether number i is prime, the algorithm takes √2 + √3 + √4 + ... + √n steps to find all the prime numbers less than or equal to n.

Observe that,

√2 + √3 + √4 + ... + √n <= n√n

Therefore, the time complexity for this algorithm is O(n√n).

# Problem 4

