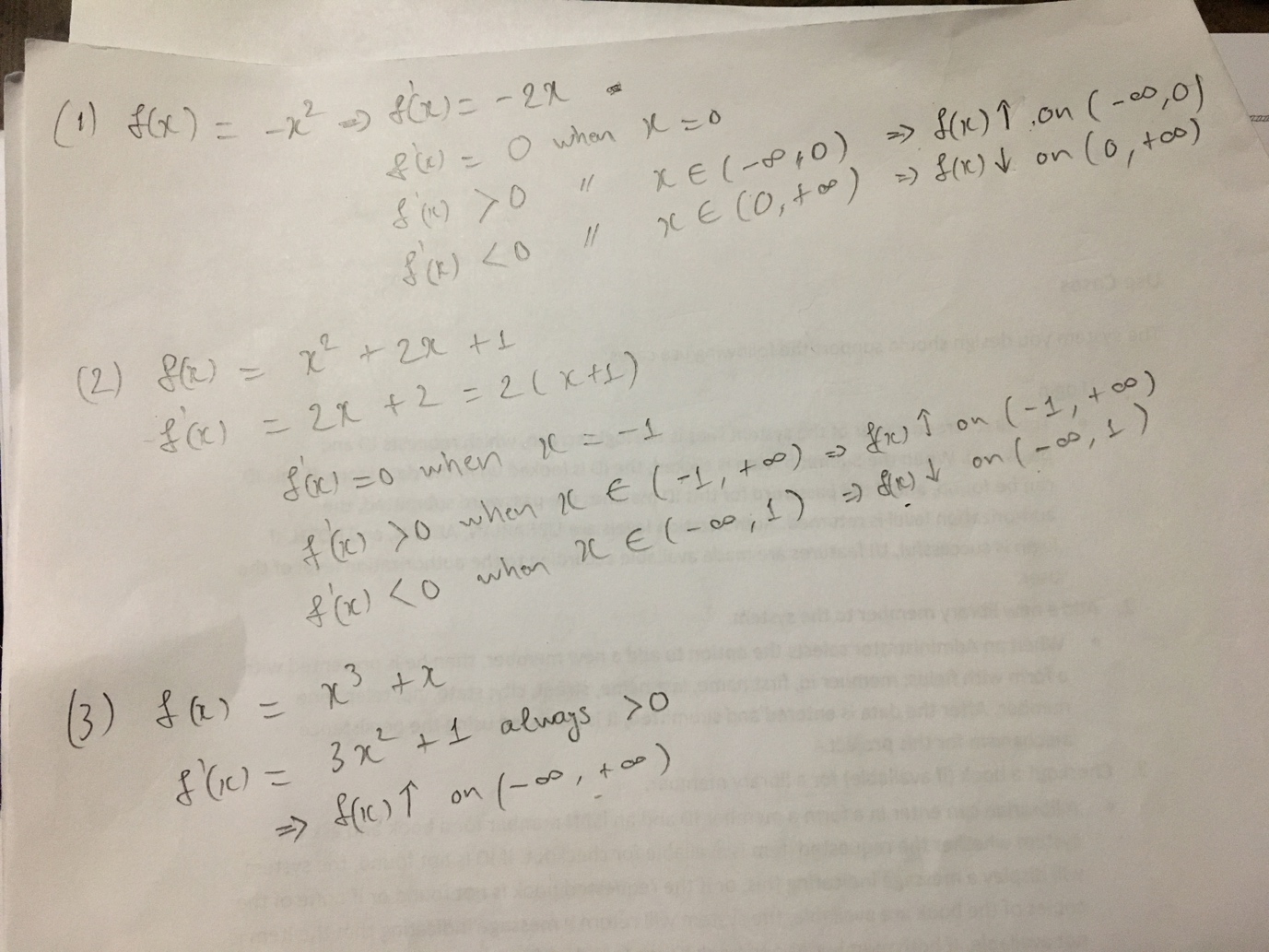
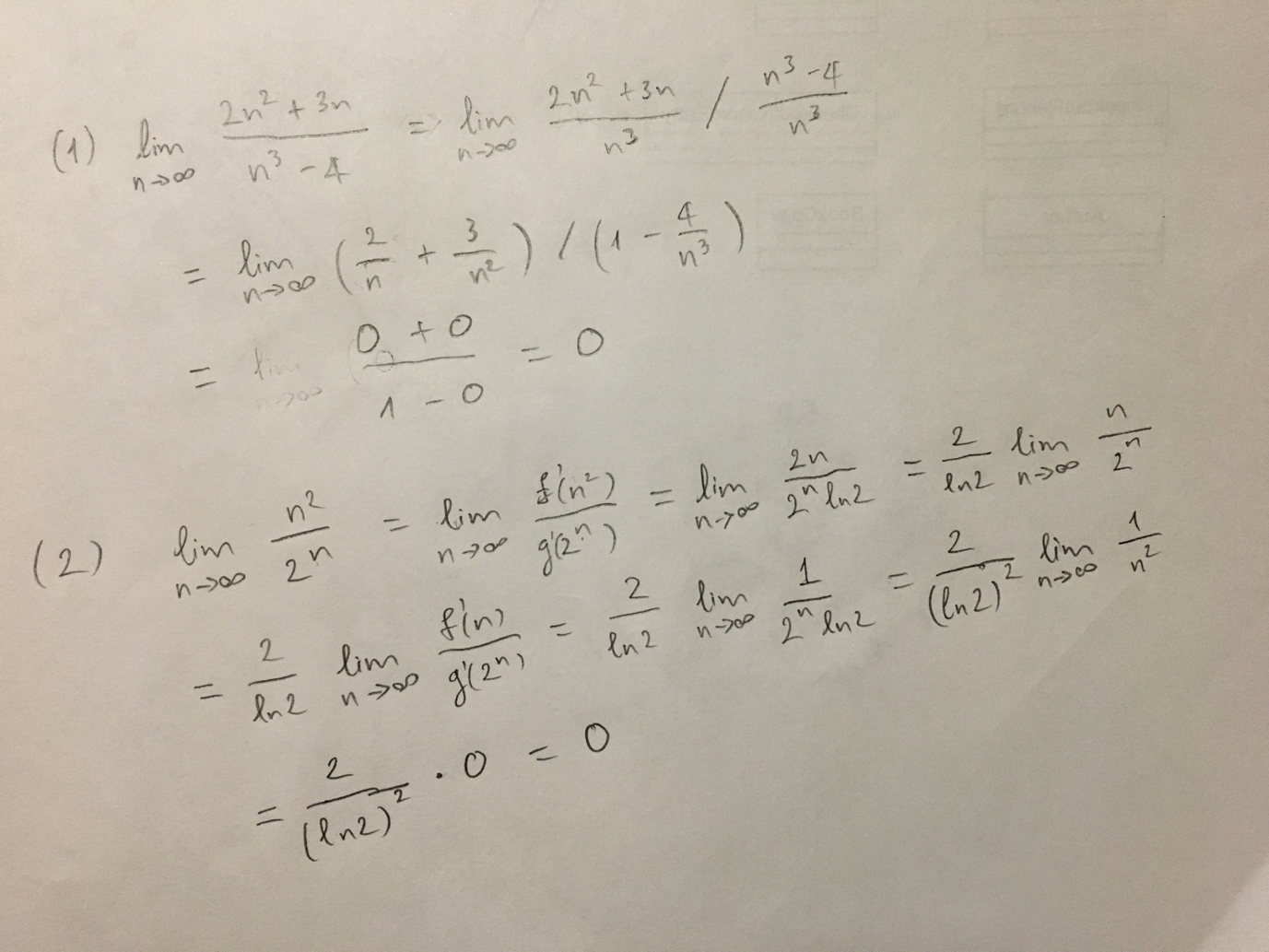
# Math Review

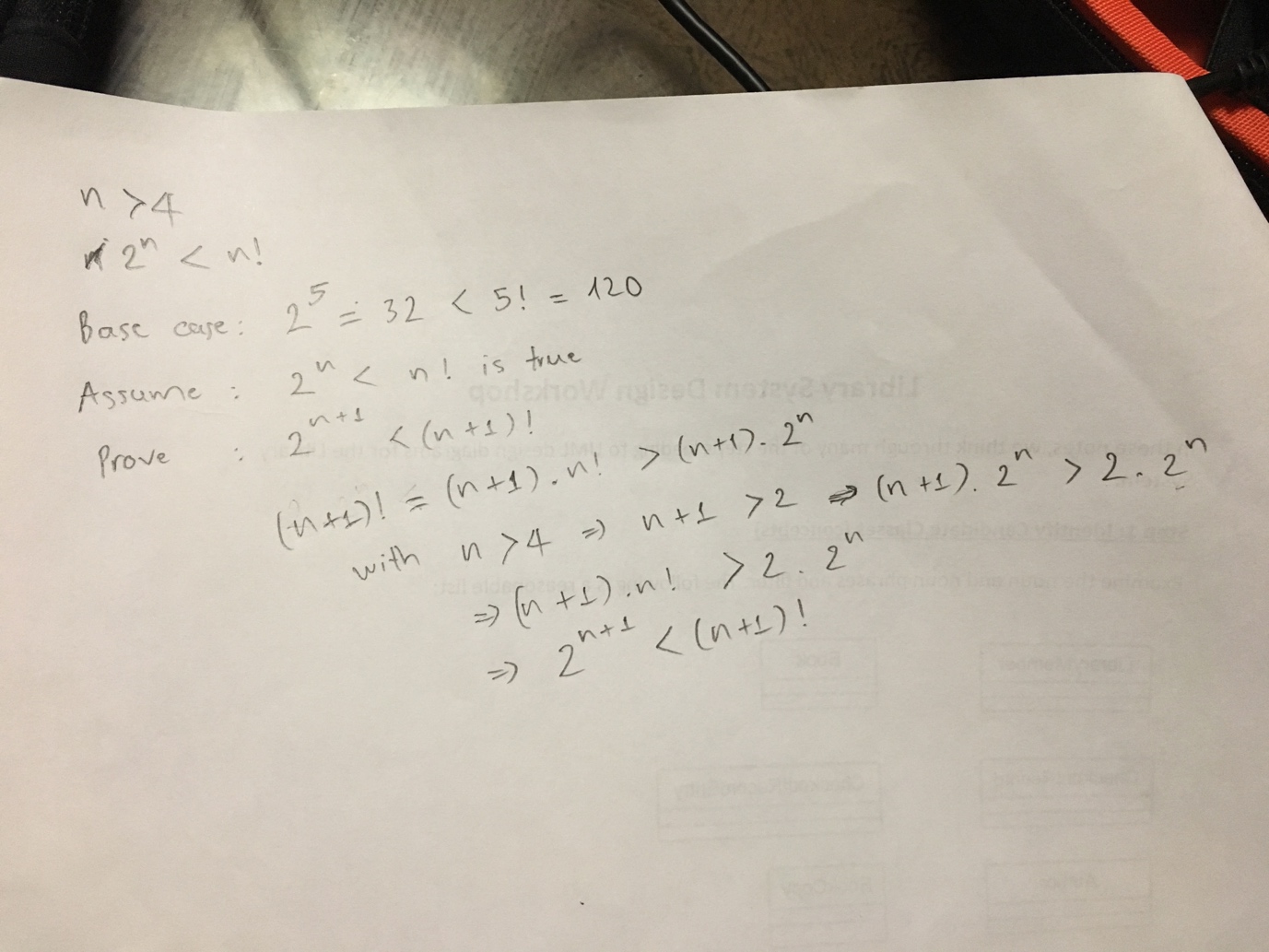
## Math Review Problem 1



## Math Review Problem 2



## Math Review Problem 3



# Problem 1

## A

1. What is a decision problem?

Answer: A problem with a "yes" or "no" answer

1. What does it mean to say that a decision problem belongs to NP?

Answer: It means a solution of problem can be verified is correct solution in polynomial time.

1. What is the Halting Problem?.

Answer: Is there a Java program(a Halting Calculator) which accepts as input a normal Java program R and an integer n (which we represent as a BigInteger), and which outputs 1 (or “true”) if R terminates normally when run on n, or else outputs 0 (“false”) if R does not terminate normally. [A “normal” Java program is one that has a public method that accepts a BigInteger argument and returns an Integer value]

1. What is a universal Java program?

Answer: A universal Java program accepts any normal Java program R, together with a BigInteger n, as inputs, and runs the method of R on n, and returns the value that R’s method returns.

## B

Why is BigInteger used as an argument for the method of a normal Java program?

Answer: Because we need a Java data type that our program can be generalize and BigInteger can satisfy that. BigInteger has no limit, it depends on the memory of the computer.

## C.

Answer:

Even though HaltingCalculator fails, it is understandable that some other programs could work. We need to argue that any attempt to solve the Halting Problem will fail.

# Problem 2

static int gcd(int m, int n) {  
 if (m > n) {  
 for (int i = n; i > 1; i--) {  
 if (n % i == 0 && m % i == 0) {  
 return i;  
 }  
 }  
 }  
  
 for (int i = m; i > 1; i--) {  
 if (m % i ==0 && n % i == 0) {  
 return i;  
 }  
 }  
 return 0;  
}

# Problem 3

public static List<List<Integer>> powerSet(List<Integer> list) {  
  
 List<List<Integer>> P = new ArrayList<>();  
 List<Integer> S = new ArrayList<>();  
 P.add(S);  
 if (list.isEmpty()) {  
 return P;  
 }  
  
 while (!list.isEmpty()) {  
 int f = list.remove(0);  
  
 List<List<Integer>> temp = new ArrayList<>();  
  
 for (List<Integer> x: P) {  
 temp.add(x);  
 }  
  
 for (List<Integer> x: temp) {  
 List<Integer> T = new ArrayList<>();  
 T.addAll(x);  
 T.add(f);  
 P.add(T);  
 }  
 }  
  
 return P;  
}  
  
public static List<Integer> subSetSum(List<Integer> S, int k) {  
 List<List<Integer>> subSet = *powerSet*(S);  
 for (List<Integer> sub: subSet) {  
 int sum = 0;  
 for (Integer i: sub) {  
 sum += i;  
 }  
 if (sum == k) {  
 return sub;  
 }  
 }  
 return null;  
}  
  
public static void main(String[] args) {  
 List<Integer> list = new ArrayList<>();  
 list.add(24);  
 list.add(1);  
 list.add(15);  
 list.add(3);  
 list.add(4);  
 list.add(15);  
 list.add(3);  
 System.*out*.println(*subSetSum*(list, 55));  
}

# Problem 4

It is true because

* T’ is T – {Sn - 1} and S’ = S – {Sn-1} so T’ still a sub set of S’
* The sum of elements in T’ now is oldSum – Sn-1 = k – Sn-1, even though there is only Sn-1 in T this is still correct (0 = 0)