**Given** a **number, find the next smallest palindrome larger than the number. Give an optimized solution rather than a brute force algorithm.**

**0: Scanned with OKEN Scanner**

**Few examples**

**1997 2002**

**Input Number Output - The next smallest palindrome**

**125 131**

250 **252**

123 **131**

**397 404**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 4512 |  | **4554  1331** |  |
|  |  |  |  |
|  | 1234 |  |  |

**Write the below function**

**int nextPalindrome(int input){**

}

ATM Dispenser Simulator

**cy: Scanned with OKEN Scanner**

Design a function that simulates an ATM machine dispensing money. The ATM contains various denominations of bills, and a customer requests a specific withdrawal amount. the function should determine whether the ATM can fulfil the request based on the available

notes and their quantities. Input:

withdrawalamount (positive integer): 'Hie amount of money the customer wants to withdraw.

atm\_notes (data structure): A data structure representing the available denominations (keys) and their corresponding quantities (values) in the ATM. This data structure can he a dictionary (Python), HashMap (Java), object (JavaScript), etc., depending on the chosen

programming language.

**Example:**

**withdrawal amount =** 270

atm\_notes =

100: 5, # 5 notes of $100 denomination

|  |  |  |  |
| --- | --- | --- | --- |
| 50: | 3, | # | 3 notes of $50 denomination |
| 20: | 2, | # | 2 notes of $20 denomination |
| 10: | 1 | # | 1 note of $10 denomination |

}

**Output:**

* Return a data structure indicating the number of notes dispensed from each denomination (if successful). This data structure should mirror the format used for

atm\_notes.

* For 270 withdrawal ATM can dispense below notes.

**atm\_notes =**

100: 2, # 2 notes of $100 denomination

50: 1, # 1 note of $50 denomination 20: 1, # 1 note of $20 denomination



*You have* to distribute *N* coconuts **equally** among some elephants according to the following conditions:



* You can choose the number of elephants that receive coconuts.
* Each elephant should get more than one coconut.
* One elephant cannot receive all the coconuts.
* All the *N* coconuts must be distributed.
* Each elephant can only receive full coconuts. You can't distribute half ***1. 4.- '4;***

coconut.

Write a program to determine whether the coconuts can be **equally** distributed among the elephants. **There will be bonus marks for optimizing the solution.**

**Imb**

10.

**Output**

Return true or false depending upon the result.

**Is- Input**

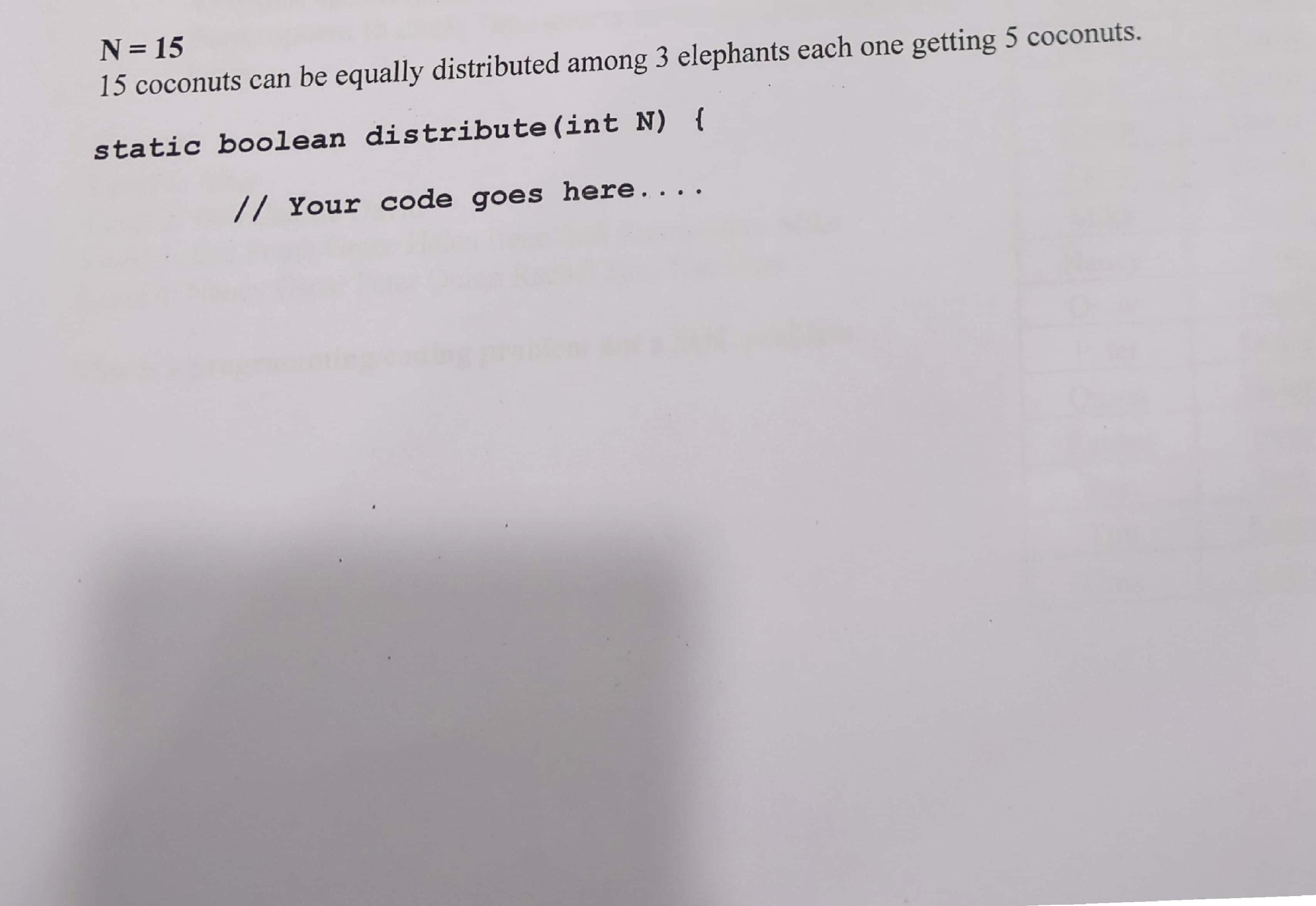
N coconuts to be distributed.

**Examples**

**Scanned with OKEN** Scanner

N = 2

size. Suppose we take a group of size 1 2 coconuts cannot be distributed among group of any then one elephant takes all the coconuts. If we take a group of size 2 each elephant will only 1 coconut which violates the rule of distribution.



N 15 ributed among 3 elephants each one getting 5 coconuts.

15 coconuts can *be* equally dist =

**static boolean distribute(int N) // Your code *goes* here...**

**Ir**

{

•

*In a company, each employee* has a *direct* manager (except the CEO) ***and zero or more direct***

***reporters. Au employee with* at least one reportee** is called a manager. The company has a *CEO,* who is the only employee with **no manager.**

The relationship between each employee and their manager is **represented by a table with** two columns: 'Employee' and 'Manager', where each row represents an employee and their direct manager. **You can *assume* a data structure to hold the *employee — manager* relationship which help you print the *required* output.**

**Write a function print() where:**

* The function should print the names of all *employees* in the company **grouped by** their ***level*** in the hierarchy.
* The CEO is at level 1, employees who directly ***report*** to the CEO are at level 2,
    
  employees who directly report to level **2 *employees*** are at level **3, and so on.**

**•**

**Manager**Alice

**Alice Alice** Bob Bob

**In *this* example: Employee**

**Bob**

* Alice ***is*** the CEO. Charlie
* Bob, Charlie and David report directly to Alice. **David**
* Eve, Frank and Grace report to ***Bob;*** Helen, Irene and Jack

report to Charlie; Karen Larry and Mike report to David. **Eve**

* Nancy reports to Eve; Oscar reports to Frank; Peter reports Frank

Bob

to Grace; Quinn reports to Helen; Rachel reports to Irene; Grace

Sam reports to ***Jack;*** Tim reports to Karen; Uma reports to Helen Charlie

Larry. Irene Charlie

Jack Charlie

**Output**

Karen David

Level 1: *Alice*

*Level 2: Bob* Charlie David Larry David

*Level 3: Eve Frank Grace Helen Irene* Jack Karen Larry Mike Mike David

*Level 4: Nancy* Oscar Peter Quinn Rachel Sam Tim Uma Nancy Eve

Oscar Frank

***This is a programming/coding* problem not a SQL problem.**

Peter Grace

Quinn Helen

|  |
| --- |
| Rachel I Irene  Sam Tim Uma I Larry |

**Jack**

Karen

**cy: Scanned with OKEN Scanner**

You have to load some order data in an application. The data object **Order** contains two attributes order-Id and entry time, for example

**CE: Scanned with OKEN Scanner**

|  |  |
| --- | --- |
| Order-  Id | Time of entry |
| 0124 | 10:15 |
| 0345 | 9:15 |
| 0873 | 13:30 |
| 0314 | 7:30 |

We will load millions of **Order** objects in memory. We have sufficient ram and memory requirements is not a constraint.

After loading the Order ***objects in a collection*** we want to perform searches on the loaded data. **Given a start and end time, the application should be able to find all**

**orders entered during the time period very efficiently.**

1. Which collection or data structure will you use to load the data so that it helps in search operation? You **just need to decide** the collection or data structure; it can be assumed the code for loading objects into the desired collection is already written and

does not need to be implemented.

1. Implement an efficient search method as follows

public Collection<Order> search (Time startTime, Time endTime){....}

1. What is the time complexity of your search method implementation? Use Big 0

notation.

Write code for

**public Collection<Order> search (Time startTime, Time endTime){**

}