2주차

9020번

풀이법

> 1. n보다 2만큼 작은 소수를 모두 구한다. (가장 작은 소수가 2이기 때문에 더해서 n이 나오기 위해서는 n-2여야 한다 for 문을 돌리꺼면 2부터 n-1까지 돌리면 []에 적합하게 만들수 있다)
> 2. 새로운 n이 시작할 때 list의 마지막 값과 n-2와 비교하여 n-2가 더 크면 list의 마지막 값부터 이어서 더 큰 소수를 찾는다
> 3. 소수 list에 존재하는 값들을 더해서 값이 나오는지 확인한다.

방법 1: 아래코드는 시간 초과가 뜸 (안 봐도 됨 방법 3만 보셈)

In [3]:

```
prime_number = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29] #일단 생각나는 정도만 써놓고 시작
n = 0

T = input()

for _ in range(int(T)):
    n = int(input())


    ##============================================
    ##n-1보다 작은 모든 소수를 찾고 저장하는 부분
    ##============================================
    if prime_number[-1] < n-1:
        for i in range(prime_number[-1]+1, n-1): # n-2 까지 실행
            if i% 2 == 0:  ## 짝수라며는 절대 소수일 수 없다.
                continue
            for j in range(3, i//2, 2): # i에 2를 나눈 값까지만 나눠보면 됨 그 뒤로는 나뉠 수 없음. /2 하면 짝수던 홀수던 만족 (나누기 2가 아니라 root 해야되는데 numpy 못 씀 ㅅㅂ)
                if i%j == 0:
                    break
            else:
                prime_number.append(i)

    ##============================================
    ##소수의 값들을 다 더해보는 부분
    ##============================================
    partition1 = 0
    partition2 = 0
    for i in range(len(prime_number)):
        if partition2 != 0 and num1 >= partition2: ## 파티션이 a와 b일때 b부터는 검사할 필요 X
            break
        num1 = prime_number[i]

        for j in range(i, len(prime_number)):
            num2 = prime_number[j]
            if n < num2:
                break
            if num1 + num2 == n:
                partition1 = num1
                partition2 = num2
                break
    print(partition1,partition2)
```

```
3
8
3 5
10
5 5
16
```

```
5 11
```

소수 빠르게 구하기 (에라토스테네스의 체)

조금의 검색 결과 가장 빠르게 소수를 구하는 방식이라고 나와있는데 실제로 코드도 빠른지 확인하는 부분

먼저 내 코드부터 *(여기서 //2가 아니라 루트를 했어야 되는데 실수함)*

In [7]:

```
import time

start = time.time()
a =[]
for i in range(0, 10000): # n-2 까지 실행
    if i% 2 == 0:  ## 짝수라며는 절대 소수일 수 없다.
        continue
    for j in range(3, i//2, 2):
        if i%j == 0:
            break
    else:
        a.append(i)
end = time.time()
print(end-start)
print(a[:50])
```

```
0.1655564308166504
[1, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97,
101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197
, 199, 211, 223, 227, 229]
```

In [11]:

```
import time

start = time.time()

a=[]
sieve = [True] * 10000

# n의 최대 약수가 sqrt(n) 이하이므로 i=sqrt(n)까지 검사
m = int(10000 ** 0.5)
for i in range(2, m + 1):
    if sieve[i] == True:           # i가 소수인 경우
        for j in range(i+i, 10000, i): # i이후 i의 배수들을 False 판정
            sieve[j] = False

    # 소수 목록 산출
a = [i for i in range(2, 10000) if sieve[i] == True]
end = time.time()
print(end-start)
print(a[:50])
```

```
0.0059833526611328125
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97,
101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197
, 199, 211, 223, 227, 229]
```

위의 결과에서 볼 수 있 듯이 훨씬 빠르다 소수는 자주 나오는 문제니 에라토스테네스의 체는 기억하자

관련 코드 및 정보는 다음 사이트에서 볼 수 있다.
https://ko.wikipedia.org/wiki/%EC%97%90%EB%9D%BC%ED%86%A0%EC%8A%A4%ED%85%8C%EB%84%A4%EC%8A%A4%EC

위 코드를 사용할 때의 문제점

현재 만들어 놓은 코드는 prime_number 리스트의 앞부분은 무시하고 그보다 큰 수만 구하면 되는데 이게 힘들다. 그렇기 때문에 인풋 자체를 인트형으로 다 받아놓고 max 값을 찾아서 그거에 대한 소수를 찾아놓고 계산이 더 효율적일 것으로 보인다.

방법2 아래도 시간 초과가 뜸

In [14]:

```
##========================================================================
## 에라토스테네스의 체
##========================================================================
def prime_list(n):
    # 에라토스테네스의 체 초기화: n개 요소에 True 설정(소수로 간주)
    sieve = [True] * n

    # n의 최대 약수가 sqrt(n) 이하이므로 i=sqrt(n)까지 검사
    m = int(n ** 0.5)
    for i in range(2, m + 1):
        if sieve[i] == True:           # i가 소수인 경우
            for j in range(i+i, n, i): # i이후 i의 배수들을 False 판정
                sieve[j] = False

    # 소수 목록 산출
    return [i for i in range(2, n) if sieve[i] == True]

##========================================================================
## 데이터 받아오기
##========================================================================
T = int(input())
n=[]
for _ in range(T):
    n.append(int(input()))

prime_number = prime_list(max(n))

##========================================================================
## 파티션 찾기
##========================================================================

for num in n:
    part1 = 0
    part2 = 0

    for i in range(len(prime_number)):
        num1 = prime_number[i]
        if (num1 > num) or (part2 != 0 and i >= part2):
            break
        for j in range(i,len(prime_number)):
            num2 = prime_number[j]
            if (num2 > num):
                break
            if num1 + num2 == num:
                part1 = num1
                part2 = num2
                break
    print(part1,part2)
```

```
3
8
10
16
3 5
5 5
5 11
```

위의 값도 시간초과가 뜸 파티션 구하는 부분의 속도를 줄여야 할 것 같다

방법 3

리스트에서 값들을 하나 하나 다 더해보는 것이 아니라 두 수의 합이 n이 될거면 n을 하나의 긴 list로 양쪽에서 한칸씩 와서 가운데서 만나면 된당

In [17]:

```
##========================================================================
## 에라토스테네스의 체
##========================================================================
```

```
def prime_list(n):
    # 에라토스테네스의 체 초기화: n개 요소에 True 설정(소수로 간주)
    sieve = [True] * n

    # n의 최대 약수가 sqrt(n) 이하이므로 i=sqrt(n)까지 검사
    m = int(n ** 0.5)
    for i in range(2, m + 1):
        if sieve[i] == True:           # i가 소수인 경우
            for j in range(i+i, n, i): # i이후 i의 배수들을 False 판정
                sieve[j] = False

    # 소수 목록 산출
    sieve[0] = False
    sieve[1] = False
    return sieve

##====================================================================
## 데이터 받아오기
##====================================================================
T = int(input())
n=[]
for _ in range(T):
    n.append(int(input()))

prime_number = prime_list(max(n))

##====================================================================
## 파티션 찾기
##====================================================================

for num in n:

    part1 = 0
    part2 = 0
    i = 1
    j = num-1
    for _ in range(num//2):
        if prime_number[i] and prime_number[j]:
            part1 = i
            part2 = j
        i+=1
        j-=1

    print(part1,part2)
```

```
3
8
10
16
3 5
5 5
5 11
```

5568번

풀이법

> 1. 주어진 리스트에서 차례대로 한 숫자를 뽑는 재귀를 사용
> 2. argument는 리스트와 K에 해당하는 값
> 3. K가 0이 되면 base case
> 4. K가 0이 아니면 전달? 전해? 받은 리스트에서 순차적으로 값을 하나씩 뽑고 다시 재귀 호출
> 5. K가 0일 때 남아있는 리스트에서 한 값만 뽑고 지금까지 뽑은 값들을 모두 연결 (string이 쌉이득임 이거 계산 개 귀찮. 난 해보고 싶어서 걍 숫자했는데 씨발임 5시간 날림 씨발)
> 6. 재귀를 나가기 전에 순차적으로 저장해 놓은 값을 모두 POP 시킨다 (STACK 개이덕)

자세한 설명은 만나서

In [1]:

```
"""
숫자의 총 수 n (4이상 10 이하)
```

```
숫자의 총 수 : n (4이상, 10 이하)
뽑는 수 : k (2이상 4 이하)
각 숫자는 1이상 99이하
"""

from itertools import permutations

nums = []
picked = []
result = []

##================================
## 재귀 함수
##================================

def choose (myList, to_go):
    global picked
    global result

    if to_go ==0:
        # 끝 여기서 result 넣어줘
        for num in myList:  # 남아있는 모든 list의 값들을 순차적으로 1의 자리에 넣어준다
            picked.append(num)
            res_num = 0
            two_digit = 0 ## 2의자리까지 가능하니깐
            for i, j in enumerate(reversed(picked)):
                check = j
                if two_digit != 0: ## 그 전 숫자가 2의 자리일 경우 씨바 두자리 수 두개일 경우 생각해줘야 됨
                    j = j * (10**two_digit)
                if check >= 10:
                    two_digit += 1
                res_num += j*(10**i)
            result.append(res_num)
            picked.pop()
        return

    for i, num in enumerate(myList):
        picked.append(num)
        del myList[i]
        choose(myList, to_go-1)
        picked.pop()
        myList.insert(i,num)

##================================================
## main part
##================================================

n = int(input())
k = int(input())

for _ in range(n):
    nums.append(int(input()))

choose(nums, k-1)

answer = set(result)

print(len(answer))
```

```
4
2
1
2
12
1
7
```

카펫 문제

이거 말고 다른 문제는 해보니깐 민준이랑 거의 완전 똑같이 나와서 굳이 안올림

이것도 비슷 할 것 같은데 민준이꺼를 안 봐봐서 그냥 올림

In [ ]:

```python
def solution(brown, red):
    answer = []
    row = 1
    col = 1
    while True:
        if red % row != 0:
            row += 1
            continue
        else:
            col = int(red / row) # 2
            t_row = row+2
            t_col = col+2 # row * col + 2 row + 2 col + 4
            if ((col*2) + 2 * (row+2)) == brown: # 4 + 6 = 10
                answer = [t_col, t_row]
                break
            else:
                row += 1

    return answer
```