



알고리즘 스터디 1주

발표자 김다인

목차

- 1부터 n 까지의 합
- N 부터 m 까지의 합
- 피보나치 수 구하기 (재귀, 반복문)
- 약수
- 최대공약수와 최소공배수
- 소수
- 조합 구하기

1부터 n 까지의 합

- [백준 8393](#)
- N 이 주어졌을 때, 1부터 n 까지 합을 구하는 프로그램
- N 의 개수는 1~10,000개

1부터 n까지의 합 : for문 쓰기

```
n = int(input())
result = 0
for i in range(1, n+1):
    result += i
print(result)
```

- 가장 간단한 솔루션
- 시간복잡도는 $O(N)$
- 공간복잡도 $O(1)$

1부터 n까지의 합 : 내장함수 쓰기

```
n = int(input())  
print(sum(range(n+1)))
```

- Sum(iterable)을 통해 한 줄로 해결 가능
- 시간복잡도 $O(1)$
- 공간복잡도 $O(N)$

1부터 n까지의 합 : 공식 쓰기

```
n = int(input())  
print((n * (n+1)) // 2)
```

- 1부터 n까지의 합을 구하는 공식은 **가우스의 공식**
- 예를 들어, 1부터 100까지 합을 구한다면
- $1 + 2 + 3 + \dots + 100$ 의 결과와,
 $1 + 100 + 2 + 99 + \dots + 50 + 51$ 의 결과가 같음
- 즉, $(1+100)$ 이 50개가 있는 상황

그럼 $(x+1) * (x//2)$ 로 구하면 되지 않나?

- 이 공식은 홀수일 때 제대로 계산하지 못하는 약점이 있음
- 따라서, **$(x * (x+1)) // 2$** 로 구하는 것이 안전함
- 이 경우 시간복잡도 $O(1)$, 공간복잡도 $O(1)$ 로 최고의 풀이법
- 참고 : <https://wikidocs.net/12258>

N부터 M까지의 합

- [프로그래머스 12912](#)
- A와 B가 주어졌을 때 A, B 사이의 모든 정수의 합을 돌려주는 함수 작성
- A와 B는 -10,000,000 이상 10,000,000 이하인 정수
 - For문으로 일일이 계산할 경우 숫자가 커질 때 연산횟수가 늘어남
- A와 B의 대소관계가 정해져있지 않음

N부터 M까지의 합

- 만약, N과 M이 같다면 둘 중 아무 숫자나 다시 돌려주면 됨
- 그렇지 않다면, 일단 주어진 A와 B 중 min을 N, max를 M으로 만들어 연산해야 함
- N부터 M까지의 합은 다음과 같이 정의 가능
- **$(M - N + 1) * (M + N) // 2$**
- 참고 : 1부터 N까지 합은 $N * (N+1) // 2$

N부터 M까지의 합

- $\text{Sum}(\text{Range}(n, m+1))$ 을 해도 구할 수 있지만,
- 이 경우에도 a와 b의 숫자가 커질 경우 공간 복잡도가 $O(N)$ 이라 꽤 많은 메모리를 소모할 수 있음

```
def solution(a, b):  
    if a == b: return a  
    return (max(a,b) - min(a,b) + 1) * (a + b) / 2
```

피보나치 수

- 재귀함수를 설명할 때 단골로 나오는 수열
- 익숙하시죠?
- 피보나치 수란?
- 0, 1, 1, 2, 3, 5, 8, 13, 21, ...
- N번째 피보나치 수는 N-1번째와 N-2번째 피보나치 수의 합이다. 단, 1번째와 2번째 수는 1이다.
- 즉...
 - if n in (1, 2): **$\text{fib}(n) = 1$**
 - if $n > 2$: **$\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2)$**

피보나치 수를 구하는 여러 가지 방법

- 재귀
- 반복문
- 다이나믹 프로그래밍 (일명 DP)

N번째 피보나치 수 구하기 문제

- [백준 2748](#)

문제

피보나치 수는 0과 1로 시작한다. 0번째 피보나치 수는 0이고, 1번째 피보나치 수는 1이다. 그 다음 2번째 부터는 바로 앞 두 피보나치 수의 합이 된다.

이를 식으로 써보면 $F_n = F_{n-1} + F_{n-2}$ ($n \geq 2$)가 된다.

$n=17$ 일때 까지 피보나치 수를 써보면 다음과 같다.

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597

n 이 주어졌을 때, n 번째 피보나치 수를 구하는 프로그램을 작성하시오.

입력

첫째 줄에 n 이 주어진다. n 은 90보다 작거나 같은 자연수이다.

출력

첫째 줄에 n 번째 피보나치 수를 출력한다.

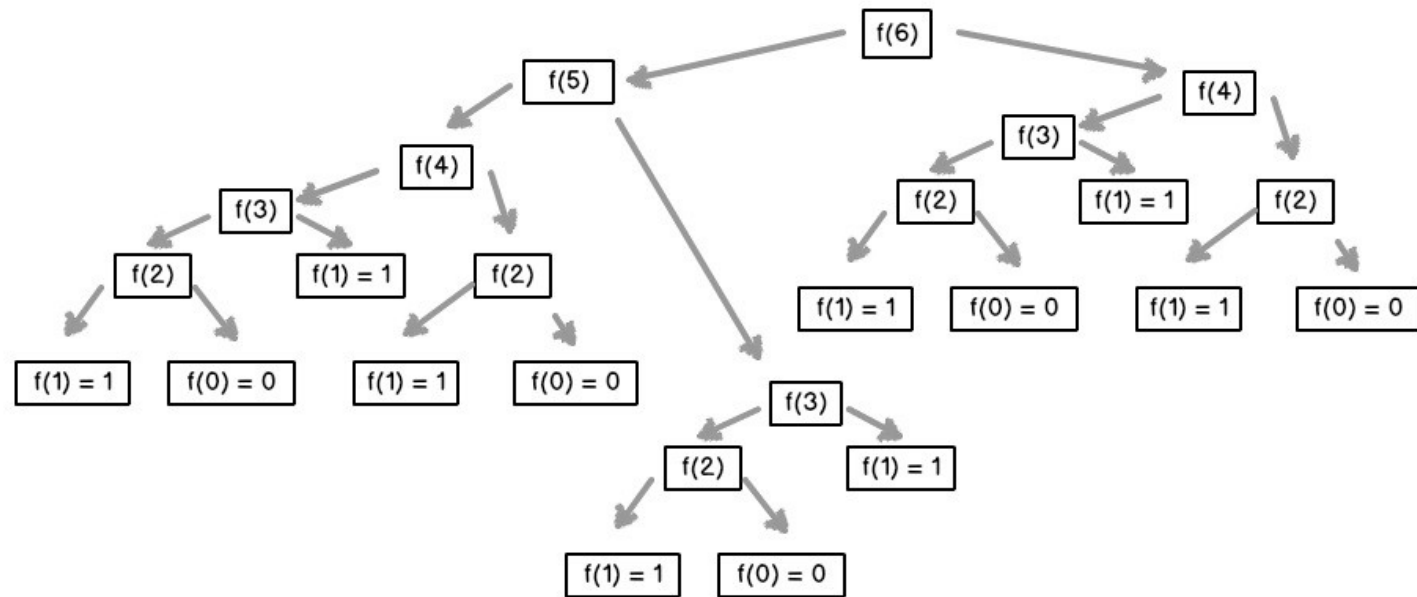
피보나치 수 구하기 : 재귀함수 사용

- 점화식을 이용해 간단히 풀이 가능
- 2 이하일 때는 1, 3 이상일 때는 $\text{fib}(n-1) + \text{fib}(n-2)$ 를 호출

```
def fib(n):  
    if n < 3: return 1  
    return fib(n-1) + fib(n-2)  
  
n = int(input())  
print(fib(n))
```

왜 피보나치 수를 재귀로 구하면 안되나요?

- 이미 다른 데서 계산한 수라고 할지라도, 그것을 사용하지 않고 그냥 계속 재귀호출을 하기 때문
- 수가 작을 때는 괜찮지만, 시간 복잡도 $O(2^N)$, 공간 복잡도 $O(2^N)$ 이기 때문에 이 방식을 쓰면 큰 피보나치 수를 구하기는 힘들



피보나치 수 구하기 : 반복문 사용

- 반복문을 사용하면 공간복잡도 $O(1)$, 시간복잡도 $O(N)$ 으로 줄어듦

```
n = int(input())
a = b = result = 1
for i in range(3, n+1):
    result = a + b
    a = b
    b = result
print(result)
```




약수 구하기

- 약수 : 어떤 수보다 같거나 작으면서 나눴을 때 나머지가 0인 숫자
- 간단하게 생각해서 n 의 약수를 전부 구하라고 하면,
- 일단 빈 리스트를 만든 다음
- 1부터 n 까지 루프를 돌면서 n 으로 나눠보고,
- 나머지가 0이면 나누는 수와 그 몫을 리스트에 합한 다음
- for 루프가 끝나고 리스트를 돌려주면 될 것 같습니다 그죠?

약수 구하는 함수

- Set()을 이용한 이유는, 만약 제곱수일 경우 i 와 $a//i$ 가 똑같아 같은 수가 2번 들어갈 수 있기 때문입니다.
- 예를 들면 16은 $4 * 4$ 인데 4가 2번 들어가면 안되겠죠
- 그리고 나누는 수와 몫을 함께 넣어 버리기 때문에 굳이 제곱근 이상으로 갈 필요가 없습니다.
- $\text{int}(a ** 0.5) + 1$ 이 별로다 하면 `math.sqrt`를 사용하셔도 효과는 똑같습니다. `Math.sqrt`가 좀 더 빠름

```
def normal_divisor(a):  
    arr = set()  
    for i in range(1, int(a ** 0.5)+1):  
        if a % i:  
            continue  
        arr.update((i, a//i))  
    return list(arr)
```



```
✓ def divisor(a):  
    return [x for x in range(1, a+1) if a % x == 0]
```

왔다 리스트 컴프리헨션

시간복잡도가 $O(N)$ 인 리스트 컴프리헨션도 맛보세요

약수 구하기 문제

- [백준 1037](#)
- 이 문제는 정확히 말하면 약수를 구하는 게 아니라 이러한 약수를 가진 수를 구하는 문제입니다.

문제

양수 A 가 N 의 진짜 약수가 되려면, N 이 A 의 배수이고, A 가 1과 N 이 아니어야 한다. 어떤 수 N 의 진짜 약수가 모두 주어질 때, N 을 구하는 프로그램을 작성하시오.

입력

첫째 줄에 N 의 진짜 약수의 개수가 주어진다. 이 개수는 50보다 작거나 같은 자연수이다. 둘째 줄에는 N 의 진짜 약수가 주어진다. 1,000,000보다 작거나 같고, 2보다 크거나 같은 자연수이고, 중복되지 않는다.

출력

첫째 줄에 N 을 출력한다. N 은 항상 32비트 부호있는 정수로 표현할 수 있다.

약수 구하기 문제 솔루션

- 저도 안 풀어봤지만 저라면...
- 약수 개수는 50보다 작습니다. 즉, 다 돌아봐도 그리 큰 시간을 차지하지 않습니다.
- 그러므로 $\min(arr)$, $\max(arr)$ 를 곱하겠습니다.
- 단, 제곱수라면 n 이 홀수겠죠? 이 경우는 따로 처리를 해주는 게 좋겠습니다.
- 각자 풀어보시면 좋겠습니다.
- 추천 : [프로그래머스 12928 \(약수의 합\)](#)

최대공약수와 최소공배수 공식

- GCD(Greatest Common Divisor) : 최대공약수
- 유클리드 호제법으로 구할 수 있음

- LCM(Least Common Multiple) : 최소공배수
- 최대공약수를 구하고 난 다음 쉽게 구할 수 있음

유클리드 호제법

- 두 수 중 큰 수를 A, 작은 수를 B로 놓는다.
- A/B를 하게 되면 몫(M)과 나머지(R)가 나오게 된다.
- 만약 나머지가 있다면(즉, 0이 아니라면) 더 쪼갤 수 있다. 이 경우 B가 A가 되며, M이 B가 된다.
- 나머지가 0이 될 때까지 2번과 3번을 계속 반복한다.
- 나머지가 0이 되었을 때의 b가 GCD이다.

$$106 / 16 = 6, \text{ remainder } 10$$

$$16 / 10 = 1, \text{ remainder } 6$$

$$10 / 6 = 1, \text{ remainder } 4$$

$$6 / 4 = 1, \text{ remainder } 2$$

$$4 / \boxed{2} = 2, \text{ remainder } 0$$

GCD

그림 출처

유클리드 호제법을 코드로 구현

- 제일 일반적인 형태의 유클리드 호제법 구현 코드
- 직관적이며 이해하기 쉬움.
- 아까 b가 GCD가 된다고 했는데, 계산법을 보면 이전 계산에서 a는 b가 되고 b는 더 작은 수인 나머지가 되므로 a를 반환하는 것이 정답

```
def gcd(a, b):
```

```
    res = 0
```

```
    while b:
```

```
        res = a % b
```

```
        a = b
```

```
        b = res
```

```
    return a
```

```
a, b = map(int, input().split())
```

```
a, b = max(a, b), min(a, b)
```

```
print(gcd(a, b))
```


극단적으로 효율적인 유클리드 호제법

- $A > B$ 가 보장되어 있는 상태에서,
• B 가 0이 아닌 동안 a 는 b 로 나누는 나머지가 된다.
• 그 다음 a, b 를 서로 바꿔주면 $a=b, b=a\%b$ 가 되어 어쨌든 식이 성립한다.
• 어렵다면 전 슬라이드의 내용으로 연습할 것

```
def gcd(a, b):  
    while b:  
        a %= b  
        a, b = b, a  
    return a
```

```
def rGcd(a, b):  
    return a if b == 0 else rGcd(b, a%b)
```

재귀로 푸는 유클리드 호제법

전 사실 재귀로 구현하는 게 더 편하긴 합니다

최소공배수 공식

- 별 거 없습니다.
- $(a * b) / \text{GCD}(a, b)$

```
✓ def LCM(a, b):  
    return (a*b) // gcd(a,b)
```

이제 여러분은 다음 문제를 풀 수 있습니다

- [백준 2609](#) (최대공약수와 최소공배수)
- [백준 5347](#) (LCM)
- [백준 13241](#) (최소공배수)
- [프로그래머스 12940](#) (최대공약수와 최소공배수)

소수 구하기

- 소수는 정수론 파트에서 정말 제일 중요한 부분
- 소수(Prime Number)는 1과 자신을 제외하고 다른 어떤 수로도 나뉘지 않는 수를 말합니다.
- 2, 3, 5, 7, 11, 13, 17, ...
- 소수란?
- 어떤 정수 n 이하의 소수를 구한다고 치면
- 약수처럼 n 보다 작은 수를 대상으로 for문을 돌리면서 소수인지 아닌지 판별할 수 있음

소수 구하기 1 : 소수를 한 번만 구해도 되는 상황

- [백준 1929](#) 같은 문제
- 이런 문제는 완전 땡큐입니다.
- 그냥 이전 슬라이드에서 말하는대로 구하면 됩니다

문제

M이상 N이하의 소수를 모두 출력하는 프로그램을 작성하시오.

입력

첫째 줄에 자연수 M과 N이 빈 칸을 사이에 두고 주어진다. ($1 \leq M \leq N \leq 1,000,000$) M이상 N이하의 소수가 하나 이상 있는 입력만 주어진다.

출력

한 줄에 하나씩, 증가하는 순서대로 소수를 출력한다.

1929번 해답

- M부터 n까지 루프를 돕니다.
- 내부 루프를 하나 더 만들되, i의 제곱근에서 나머지를 버린 값까지 내부 루프를 만듭니다.
- 만약 안쪽 루프에서 $i \% j == 0$, 즉 나누어지는 수가 있다면 소수가 아니므로 탈출합니다.
- [For-else 구문](#) : for문에서 break 없이 무사 탈출하면 else문이 실행되는 것으로서, 이런 문제를 풀 때 안성맞춤입니다

```
import math
m, n = map(int, input().split())

for i in range(m, n+1):
    for j in range(2, int(math.sqrt(i))+1):
        if i % j == 0:
            break
    else:
        print(i)
```

소수 구하기 2 : 수가 여러 번 주어져서 매번 소수를 구해야 하는 상황

- [백준 9020](#), [백준 4948](#)
- 이 문제는 아까처럼 풀면 시간 초과가 납니다.
- 테스트 케이스 개수가 얼마나 클지 예측할 수 없으며, 중복된 숫자가 나오면 아까 풀었는데 값을 기억하지 못해서 헛고생을 또 하기 때문입니다.
- 그렇다면 해결법은?

문제

1보다 큰 자연수 중에서 1과 자기 자신을 제외한 약수가 없는 자연수를 소수라고 한다. 예를 들어, 5는 1과 5를 제외한 약수가 없기 때문에 소수이다. 하지만, 6은 $6 = 2 \times 3$ 이기 때문에 소수가 아니다.

골드바흐의 추측은 유명한 정수론의 미해결 문제로, 2보다 큰 모든 짝수는 두 소수의 합으로 나타낼 수 있다는 것이다. 이러한 수를 골드바흐 수라고 한다. 또, 짝수를 두 소수의 합으로 나타내는 표현을 그 수의 골드바흐 파티션이라고 한다. 예를 들면, $4 = 2 + 2$, $6 = 3 + 3$, $8 = 3 + 5$, $10 = 5 + 5$, $12 = 5 + 7$, $14 = 3 + 11$, $14 = 7 + 7$ 이다. 10000보다 작거나 같은 모든 짝수 n 에 대한 골드바흐 파티션은 존재한다.

2보다 큰 짝수 n 이 주어졌을 때, n 의 골드바흐 파티션을 출력하는 프로그램을 작성하시오. 만약 가능한 n 의 골드바흐 파티션이 여러 가지인 경우에는 두 소수의 차이가 가장 작은 것을 출력한다.

입력

첫째 줄에 테스트 케이스의 개수 T 가 주어진다. 각 테스트 케이스는 한 줄로 이루어져 있고 짝수 n 이 주어진다.

출력

각 테스트 케이스에 대해서 주어진 n 의 골드바흐 파티션을 출력한다. 출력하는 소수는 작은 것부터 먼저 출력하며, 공백으로 구분한다.

제한

- $4 \leq n \leq 10,000$

에라토스테네스의 체

- 에라토스테네스의 체를 이용하면, 문제에서 정한 max까지의 정수 중에서 소수만 골라둔 리스트를 만들 수 있다!
- 어떤 구간까지의 소수를 계속 구해야 하고, 소수를 구해야 하는 테스트 케이스 개수가 많을 때 쓰기 좋음

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110
111	112	113	114	115	116	117	118	119	120

Prime numbers



에라토스테네스의 체 알고리즘

알고리즘 [편집]

1. 2부터 소수를 구하고자 하는 구간의 모든 수를 나열한다. 그림에서 회색 사각형으로 두른 수들이 여기에 해당한다.
2. 2는 소수이므로 오른쪽에 2를 쓴다. (빨간색)
3. 자기 자신을 제외한 2의 배수를 모두 지운다.
4. 남아있는 수 가운데 3은 소수이므로 오른쪽에 3을 쓴다. (초록색)
5. 자기 자신을 제외한 3의 배수를 모두 지운다.
6. 남아있는 수 가운데 5는 소수이므로 오른쪽에 5를 쓴다. (파란색)
7. 자기 자신을 제외한 5의 배수를 모두 지운다.
8. 남아있는 수 가운데 7은 소수이므로 오른쪽에 7을 쓴다. (노란색)
9. 자기 자신을 제외한 7의 배수를 모두 지운다.
10. 위의 과정을 반복하면 구하는 구간의 모든 소수가 남는다.

에라토스테네스의 체 구현

```
import math
def sieve(n):
    # 0, 1은 어차피 소수가 아니므로 False. 나머지는 일단 True로 만들
    prime = [0, 0] + ([1] * (n-1))
    # 2부터 n의 제곱근까지 루프 돌기
    for i in range(2, int(math.sqrt(n))+1):
        # 만약 해당 인덱스가 True, 즉 소수라면
        if prime[i]:
            # 소수의 배수들은 모두 False가 되어야 함
            for j in range(i*2, len(prime), i):
                prime[j] = 0
    # 이대로 하면 [0, 0, 1, 1, 0, 1] 이런 식이기 때문에 enumerate로 정리해줌
    return [idx for idx, val in enumerate(prime) if val]
```

팩토리얼,
순열, 조합
공식

순열 조합

$$n! = n \times (n-1) \times \cdots \times 1$$

$${}_nP_r = \frac{n!}{(n-r)!}$$

$${}_nC_r = \frac{n!}{(n-r)! r!}$$

이것을 코드로 구현할 수 있다 면...

- 다음과 같은 문제를 풀 수 있습니다.
- [백준 2407](#)

문제

nC_m 을 출력한다.

입력

n 과 m 이 주어진다. ($5 \leq n \leq 100, 5 \leq m \leq 100, m \leq n$)

출력

nC_m 을 출력한다.

근데 보통은 다음과
같은 문제들이 나오니
다

- [백준 10974](#)

문제

N이 주어졌을 때, 1부터 N까지의 수로 이루어진 순열을 사전순으로 출력하는 프로그램을 작성하시오.

입력

첫째 줄에 $N(1 \leq N \leq 8)$ 이 주어진다.

출력

첫째 줄부터 $N!$ 개의 줄에 걸쳐서 모든 순열을 사전순으로 출력한다.

파이썬은 생각보다 더 좋습니다

- Itertools의 permutations, combinations를 풀면 이런 문제는 진짜 쉽게 풀립니다.
- Permutations은 순열을 가리키고, combinations는 조합을 가리킴
- 예를 들어 1부터 3까지의 순열이라면
- [1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1]이 순열입니다.
- 반면 조합은 [1, 2, 3]만 포함.
- 감이 오시죠?

사용법

- Itertools 라이브러리에서 permutations, combinations를 import하고 쓰면 됩니다.
- 저는 주로 for문과 자주 씁니다.
- 이 둘은 제너레이터라 결과를 하나씩 반환하기 때문에 메모리 소모가 크지 않은 것이 장점

```
from itertools import permutations, combinations
n = int(input())
for combi in combinations(range(1, n+1), 3):
    print(*combi)
for perm in permutations(range(1, n+1)):
    print(*perm)
```


여러분은 다음 문제를 풀 수 있습니다

- [백준 1182](#)
- [백준 9742](#)

다음 시간에 발표할 문제 리스트

- 백준 1978 (소수 찾기, 실버4)
 - 백준 11653 (소인수분해, 실버4)
 - 백준 1182 (부분수열의 합, 실버2)
 - 백준 9020 (골드바흐의 추측, 실버1)
 - 백준 17087 (숨바꼭질 6, 실버1)
-
- 1일 1문제를 위한 추가문제 (오늘 수업과 연관없는 문제)
 - 백준 11650 (좌표 정렬하기, 실버5)

끝!!

- 수고하셨습니다~