3장 리스트 처리 프로그램

3.1 리스트 회전

프로그램 개요

- 회전 단계수 k를 입력으로 받아 리스트의 원소들을 k 단계만큼 오른쪽으로 회전시킴
  - 회전 단계별로 리스트의 마지막 원소는 리스트의 첫 번째 원소로 이동

- 입력 : 리스트 원소의 개수 N과 회전 단계수 k
- 출력 : k 단계만큼 오른쪽으로 회전한 리스트

- 예) 회전 단계수 k가 3일 때
  - 리스트 [1, 2, 3, 4, 5, 6, 7]를 오른쪽으로 3 단계 회전한 리스트 [5, 6, 7, 1, 2, 3, 4]를 출력

프로그램 작성

- 원래 리스트와 동일한 크기의 추가 리스트를 사용하여 리스트의 원소를 k 단계만큼 회전시키는 프로그램 작성

```
>>>
원소의 개수 : 7
회전 단계수 : 3
원래 리스트 : [1, 2, 3, 4, 5, 6, 7]
회전 리스트 : [5, 6, 7, 1, 2, 3, 4]
>>>
원소의 개수 : 10
회전 단계수 : 6
원래 리스트 : [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
회전 리스트 : [5, 6, 7, 8, 9, 10, 1, 2, 3, 4]
>>>
```

프로그래밍 과제

- 추가 리스트를 사용하지 않고 원래 리스트에서 원소를 k 단계만큼 회전시키는 프로그램을 작성
  - 마지막 원소부터 시작하여 연속된 원소를 두 개씩 계속 교환해 나가면, 추가 리스트를 사용하지 않고도 리스트의 원소를 회전시킬 수 있음
  - n = 5이고 k = 3일 때, 리스트의 원소들이 교환되는 과정

```
>>>
원소의 개수 : 5
회전 단계수 : 3
원래 리스트 : [1, 2, 3, 4, 5]
[1, 2, 3, 5, 4]
[1, 2, 5, 3, 4]
[1, 5, 2, 3, 4]
[5, 1, 2, 3, 4]
[5, 1, 2, 4, 3]
[5, 1, 4, 2, 3]
[5, 4, 1, 2, 3]
[4, 5, 1, 2, 3]
[4, 5, 1, 3, 2]
[4, 5, 3, 1, 2]
[4, 3, 5, 1, 2]
[3, 4, 5, 1, 2]
회전 리스트 : [3, 4, 5, 1, 2]
>>>
```

3.2 두 수의 합

프로그램 개요 (1)

- 자연수로 이루어진 리스트의 두 원소를 더한 합이 사용자로부터 입력 받은 목표값과 같은 것이 있으면, 두 원소가 각각 리스트의 몇 번째 원소인지 출력

  - 입력 : 리스트의 원소 개수와 목표값
  - 출력 : x 번째와 y 번째 원소

- 리스트를 생성할 때는 리스트의 원소 개수를 입력 받아, 원소의 개수만큼 1부터 원소 개수의 2배 사이의 난수를 발생시킴
  - 예) 원소의 개수로 10을 입력하면 1부터 20 사이의 난수를 발생시킴

프로그램 개요 (2)

- 리스트 [5, 1, 4, 6, 3]의 모든 원소 쌍을 더하는 순서

[0]	[1]	[2]	[3]	[4]	두 수의 합	출력
5	1				6	
5		4			9	1 번째 원소와 3 번째 원소
5			6		11	
5				3	8	
	1	4			5	
	1		6		7	
	1			3	4	
		4	6		10	
		4		3	7	
			6	3	9	4 번째 원소와 5 번째 원소

파이썬 문법

모듈 (1)

- 라이브러리 모듈(library module)
  - 프로그래밍을 하다 보면 여러 프로그램에서 같은 동작을 하는 부분이 필요한 경우가 있음
  - 프로그램에서 같은 동작을 하는 부분을 구현하는 가장 일반적인 방법은 함수를 사용하는 것임
  - print(), int(), len() 등의 함수는 사용자가 작성하지 않았지만 시스템에서 이미 구현이 되어 있기 때문에 그냥 가져다 쓰기만 하면 됨
  - 파이썬에서 제공하는 라이브러리 모듈(library module)은 함수보다 더 큰 단위의 코드 묶음을 의미하는데, 변수, 함수, 클래스 등의 프로그램 단위들이 모여 독립적으로 실행되는 구조를 가지고 있음
  - 파이썬에서는 프로그래밍에 필요한 많은 기능들을 내장 라이브러리 모듈로 제공하고 있으므로 프로그램을 작성하기 전에 동일한 기능을 하는 모듈이 있는지 찾아볼 필요가 있음

모듈 (2)

- 모듈을 불러오는 방법

```
import 모듈
from 모듈 import 클래스
```

- 예) datetime 모듈의 클래스

datetime의 클래스	설명
class datetime.date	일반적으로 사용되는 그레고리안 달력의 년, 월, 일을 나타냄
class datetime.time	시간을 시, 분, 초, 마이크로 초, 시간대로 나타냄
class datetime.datetime	date 클래스와 time 클래스의 조합으로 년, 월, 일, 시, 분, 초, 마이크로 초, 시간대로 나타냄
class datetime.timedelta	두 날짜 혹은 시간 사이의 간격을 표현함

모듈 (3)

- datetime 모듈을 사용하기 위해 import 명령문 실행

```
>>> import datetime
```

- today() 메소드 : 오늘 날짜 출력 (date 클래스)

```
>>> datetime.date.today()
datetime.date(2015, 10, 15)
```

모듈 (4)

- 오늘 날짜 중에서 연월일을 알고 싶은 경우

```
>>> d = datetime.date.today()
>>> d.year
2015
>>> d.month
10
>>> d.day
15
```

- now() 메소드 : 현재 시간 출력 (datetime 클래스)

```
>>> datetime.datetime.now()
datetime.datetime(2015, 10, 15, 19, 32, 53, 250709)
```

모듈 (5)

- datetime 모듈에 있는 date 클래스만을 불러오기 위한 명령문

```
>>> from datetime import date
```

  - datetime 모듈에서 date 클래스만 불러왔으므로 date 클래스의 메소드만 실행이 되고, 나머지 클래스의 메소드를 실행하면 오류 발생

```
>>> date.today()
datetime.date(2015, 10, 15)
>>> datetime.now()
Traceback (most recent call last):
File "<pyshell#8>", line 1, in <module>
datetime.now()
NameError: name 'datetime' is not defined
```

난수 발생 함수 (1)

- random 모듈
  - 난수(random number)를 발생시킴
- 난수 관련 메소드

메소드	설명
random()	0에서 1 사이의 난수를 발생시킴
randint(시작 범위, 끝 범위)	시작 범위와 끝 범위 사이에 있는 정수인 난수를 발생시킴
randrange()	range() 함수에서 정의하는 범위에 있는 정수인 난수를 발생시킴
shuffle()	리스트의 원소를 뒤죽박죽 섞어 놓음
choice()	리스트의 원소 중 하나를 고름

난수 발생 함수 (2)

- random() 메소드

```
>>> import random
>>> random.random()
0.021902111521846845
>>> random.random()
0.9326712728512175
```

- randint() 메소드

```
>>> random.randint(1,100)
59
>>> random.randint(1,100)
23
>>> random.randint(1,100)
91
>>> random.randint(1,100)
33
```

난수 발생 함수 (3)

- randrange() 메소드

```
>>> random.randrange(1,7)
4
>>> random.randrange(1,7)
1
>>> random.randrange(7)
6
>>> random.randrange(7)
0
```

- shuffle() 메소드

```
>>> a = [1, 2, 3, 4, 5, 6]
>>> random.shuffle(a)
>>> a
[3, 1, 6, 5, 4, 2]
```

난수 발생 함수 (4)

- choice() 메소드

```
>>> b = ['짜장면', '짬뽕', '볶음밥', '잡채밥']
>>> random.choice(b)
'짜장면'
>>> random.choice(b)
'잡채밥'
```

프로그래밍 연습

- 주사위를 N번 던졌을 때 각 면이 나오는 확률을 계산하는 프로그램 작성
- 함수 castDice()에서 randint() 메소드 사용
- 프로그램의 실행 예

```
>>>
N = 10000
1    16.7%
2    16.3%
3    16.2%
4    17.7%
5    16.7%
6    16.5%
>>>
```

프로그램 작성

- 중복을 허용하고 순서가 없는 리스트에서 두 원소를 더한 합이 사용자로부터 입력 받은 목표값과 같은 것이 있으면, 두 원소가 각각 리스트의 몇 번째 원소인지 출력하는 프로그램 작성

```
>>>
리스트의 원소 개수 입력 : 10
리스트 : [2, 19, 18, 14, 2, 17, 7, 14, 4, 15]
목표값 입력 : 16
두 수의 합이 16인 원소 쌍
1 번째와 4 번째 원소
1 번째와 8 번째 원소
4 번째와 5 번째 원소
5 번째와 8 번째 원소
>>>
```

프로그래밍 과제 (1)

- 중복 원소가 없는 정렬된 리스트에서 두 원소를 더한 합이 사용자로부터 입력 받은 목표값과 같은 것이 있으면, 두 원소가 각각 리스트의 몇 번째 원소인지 출력하는 프로그램 작성

```
>>>
리스트의 원소 개수 입력 : 15
정렬된 리스트 : [7, 9, 11, 11, 13, 14, 16, 17, 19, 21, 24, 26, 29, 30, 30]
중복이 제거된 리스트 : [7, 9, 11, 13, 14, 16, 17, 19, 21, 24, 26, 29, 30]
목표값 입력 : 30
두 수의 합이 30인 원소 쌍
2 번째와 9 번째 원소
3 번째와 8 번째 원소
4 번째와 7 번째 원소
5 번째와 6 번째 원소
>>>
```

프로그래밍 과제 (2)

- 대략적인 알고리즘
  - 변수 i는 리스트의 왼쪽부터 시작하도록 하고, 변수 j는 리스트의 오른쪽에서 시작하도록 한다. i의 값이 j의 값보다 작은 경우 두 원소 num[i]와 num[j]의 합을 구해 이것이 목표값과 같으면 i+1 번째 원소와 j+1 번째 원소를 출력한 다음, i의 값은 1 증가시키고 j의 값은 1 감소시킴
  - 만일 두 원소의 합이 목표값보다 작으면 i의 값을 1 증가시키고, 두 원소의 합이 목표값보다 같거나 크면 j의 값을 1 감소시킴

3.3 최장 연속 순차

프로그램 개요

- 중복 원소가 없고 자연수로 이루어진 리스트에서 원소들의 연속 순차 중에 가장 긴 순차의 길이를 구하는 프로그램을 작성

  - 입력 : 난수의 개수
  - 출력 : 최장 연속 순차의 길이

- 예) 리스트 [15, 3, 9, 6, 5, 4, 11, 10]이 있다고 할 때
  - [3, 4, 5, 6]과 [9, 10, 11]을 연속 순차라고 함
  - 연속 순차가 여러 개 있을 경우 이 중에서 가장 긴 것을 최장 연속 순차(longest consecutive sequence)라고 부름
  - 이 예에서는 가장 긴 연속 순차가 [3, 4, 5, 6]이므로 이것의 길이인 4를 출력

파이썬 문법

집합 자료형 (1)

- 집합 자료형(set data type)의 특징
  - 원소의 순서가 없음(unordered)
  - 중복을 허용하지 않음
- 집합 자료형의 생성 : 키워드 set을 사용
  - 집합 자료형은 원소의 순서가 없어서 인덱스를 지원하지 않기 때문에 인덱스를 사용하여 s[0]을 접근하려고 하면 오류 발생

```
>> s1 = set([1, 2, 3, 4])
>>> s1
{1, 2, 3, 4}
>>> type(s1)
<class 'set'>
>>> s1[0]
Traceback (most recent call last):
File "<pyshell#23>", line 1, in <module>
s1[0]
TypeError: 'set' object does not support indexing
```

집합 자료형 (2)

- 집합 자료형의 생성 : 중괄호({, }) 사용
  - 중복 원소가 있으면 하나만 남기고 나머지는 제거됨

```
>>> s2 = {5, 6, 7, 5, 8, 9, 8, 9, 9}
>>> s2
{8, 9, 5, 6, 7}
>>> type(s2)
<class 'set'>
>>>
```

- 문자열을 집합으로 만드는 예

```
>>> s2 = set('School')
>>> s2
{'S', 'c', 'o', 'h', 'l'}
>>> s3 = set('Hello')
>>> s3
{'H', 'e', 'o', 'l'}
```

집합의 연산과 메소드 (1)

- 집합 자료형에서 사용할 수 있는 연산
  - 합집합, 교집합, 차집합
- 집합 s1과 s2

```
>>> s1 = {1, 2, 3, 4, 5}
>>> s1
{1, 2, 3, 4, 5}
>>> s2 = {3, 4, 5, 6, 7}
>>> s2
{3, 4, 5, 6, 7}
```

- 합집합

```
>>> s1 | s2
{1, 2, 3, 4, 5, 6, 7}
>>> s1.union(s2)
{1, 2, 3, 4, 5, 6, 7}
```

집합의 연산과 메소드 (2)

- 교집합

```
>>> s1 & s2
{3, 4, 5}
>>> s1.intersection(s2)
{3, 4, 5}
```

- 차집합

```
>>> s1 - s2
{1, 2}
>>> s1.difference(s2)
{1, 2}
```

집합의 연산과 메소드 (3)

- 집합의 함수와 메소드

함수/메소드	설명
x in s	원소 x가 집합 s의 원소인가? (x ∈ s)
x not in s	원소 x가 집합 s의 원소가 아닌가? (x ∉ s)
s.add(x)	원소 x를 집합 s에 추가
s.remove(x)	원소 x를 집합 s에서 제거. 없으면 KeyError 발생.
s.discard(x)	원소 x가 있다면 집합 s에서 제거
s.pop()	집합 s에서 임의의 원소를 하나 반환하고 집합에서 제거. 공집합이면 KeyError 발생.
s.clear()	집합 s의 모든 원소 삭제

집합의 연산과 메소드 (4)

- 'in'과 'not in'

```
>>> s = {1, 2, 3}
>>> 1 in s
True
>>> 4 in s
False
>>> 2 not in s
False
>>> 5 not in s
True
>>>
```

집합의 연산과 메소드 (5)

- 원소를 추가하고 삭제하는 연산 (1)

```
>>> s = set()
>>> s
set()
>>> type(s)
<class 'set'>
>>> s.add(1)
>>> s
{1}
>>> s.add(2)
>>> s
{1, 2}
>>> s.add(3)
>>> s
{1, 2, 3}
```

집합의 연산과 메소드 (6)

- 원소를 추가하고 삭제하는 연산 (2)

```
>>> s.remove(2)
>>> s
{1, 3}
>>> s.discard(3)
>>> s
{1}
>>> s.remove(2)
Traceback (most recent call last):
File "<pyshell#51>", line 1, in <module>
s.remove(2)
KeyError: 2
>>> s.discard(2)
>>> s
{1}
```

집합의 연산과 메소드 (7)

- 공집합을 만들 때
  - 's = set()'를 사용
  - 's = {}'을 사용하면 집합 자료형이 아니라 사전 자료형(dict data type)이 됨

```
>>> s = {}
>>> s
{}
>>> type(s)
<class 'dict'>
>>>
```

집합의 연산과 메소드 (8)

- pop()과 clear() 메소드

```
>>> s = {1, 2, 3, 4, 5}
>>> s
{1, 2, 3, 4, 5}
>>> s.pop()
1
>>> s
{2, 3, 4, 5}
>>> s.pop()
2
>>> s
{3, 4, 5}
>>> s.clear()
>>> s.pop()
Traceback (most recent call last):
File "<pyshell#69>", line 1, in <module>
s.pop()
KeyError: 'pop from an empty set'
```

프로그래밍 연습 (1)

- 1부터 N 사이의 정수 난수 N개를 발생시켜 집합에 삽입한 다음, 1부터 N 사이의 정수 중 집합에 없는 수를 출력하는 프로그램 작성
- 프로그램의 실행 예

```
>>>
N = 10
s =  {1, 3, 4, 6, 8, 10}
집합에 없는 원소 : 2 5 7 9
>>>
N = 20
s =  {1, 3, 5, 6, 7, 10, 11, 12, 14, 15, 16, 18, 20}
집합에 없는 원소 : 2 4 8 9 13 17 19
>>>
```

튜플 자료형 (1)

- 튜플 자료형(tuple data type)
  - 소괄호를 사용하여 원소들을 묶어줌
  - 상수와 비슷하게 원소가 한번 입력되면 변경이 되지 않음

```
>>> a = (1, 2, 3)
>>> a
(1, 2, 3)
>>> b = (4, 5, 6, 7)
>>> b
(4, 5, 6, 7)
>>> c = a + b
>>> c
(1, 2, 3, 4, 5, 6, 7)
>>> d = (a, b, c)
>>> d
((1, 2, 3), (4, 5, 6, 7), (1, 2, 3, 4, 5, 6, 7))
```

튜플 자료형 (2)

- 인덱싱과 슬라이싱

```
>>> e = (1, 2, 3, 4)
>>> e[0]
1
>>> e[1]
2
>>> e[0:3]
(1, 2, 3)
>>> e[1:4]
(2, 3, 4)
>>> e[:]
(1, 2, 3, 4)
```

튜플 자료형 (3)

- 삭제 연산과 치환 연산을 지원하지 않음

```
>>> f = (1, 2, 3)
>>> del f[0]
Traceback (most recent call last):
File "<pyshell#112>", line 1, in <module>
del f[0]
TypeError: 'tuple' object doesn't support item deletion
>>> f[0] = 4
Traceback (most recent call last):
File "<pyshell#113>", line 1, in <module>
f[0] = 4
TypeError: 'tuple' object does not support item assignment
```

튜플 자료형 (4)

- 함수에서 1개 이상의 데이터를 반환할 때 사용하면 편리

```
def sum_average(a, b, c):
    sum_int = a + b + c
    average_int = sum_int / 3
    return (sum_int, average_int)

a = int(input('a = '))
b = int(input('b = '))
c = int(input('c = '))
result = sum_average(a, b, c)
print('합 = %d, 평균 = %f'%(result[0], result[1]))
```

프로그래밍 연습 (2)

- 1부터 N 사이의 정수 난수 10개를 발생시켜 리스트에 삽입한 다음, 이 리스트에서 최대값과 최소값을 반환하는 함수 maxMin() 작성
- math 모듈에 있는 max(), min() 함수를 사용하지 않음
- 프로그램의 실행 예

```
>>>
N = 30
리스트 :  [1, 12, 27, 15, 3, 15, 18, 25, 18, 4]
최대값 = 27, 최소값 = 1
>>>
N = 50
리스트 :  [9, 27, 30, 24, 46, 23, 14, 10, 42, 11]
최대값 = 46, 최소값 = 9
>>>
```

프로그램 작성

- 난수의 개수를 입력 받은 다음 생성된 난수를 리스트에 추가함
- 이렇게 만들어진 리스트를 집합에 저장한 다음, 최장 연속 순차의 길이를 구하는 프로그램 작성

```
>>>
난수의 개수 입력 : 10
리스트 : [8, 3, 1, 8, 9, 3, 8, 2, 4, 1]
집합 : {1, 2, 3, 4, 8, 9}
최장 연속 순차의 길이 : 4
>>>
난수의 개수 입력 : 15
리스트 : [8, 1, 15, 13, 13, 3, 15, 6, 5, 6, 12, 5, 9, 7, 4]
집합 : {1, 3, 4, 5, 6, 7, 8, 9, 12, 13, 15}
최장 연속 순차의 길이 : 7
>>>
```

프로그래밍 과제 (1)

- 정렬되지 않은 리스트에 대해 최장 연속 순차의 길이와 최장 연속 순차를 함께 출력하는 프로그램 작성
  - 두 개의 데이터를 반환할 때 튜플 자료형 사용

```
>>>
난수의 개수 입력 : 10
리스트 : [7, 10, 1, 6, 8, 3, 2, 6, 6, 4]
집합 : {1, 2, 3, 4, 6, 7, 8, 10}
최장 연속 순차의 길이 : 4
최장 연속 순차 : [1, 2, 3, 4]
>>>
난수의 개수 입력 : 10
리스트 : [6, 8, 9, 7, 2, 8, 7, 1, 4, 8]
집합 : {1, 2, 4, 6, 7, 8, 9}
최장 연속 순차의 길이 : 4
최장 연속 순차 : [6, 7, 8, 9]
>>>
```

프로그래밍 과제 (2- 1)

- 리스트가 정렬되어 있을 때 최장 연속 순차의 길이를 구하는 프로그램 작성

```
>>>
난수의 개수 입력 : 10
리스트 : [2, 3, 6, 7, 8, 10]
최장 연속 순차의 길이 : 3
>>>
난수의 개수 입력 : 15
리스트 : [2, 3, 4, 5, 6, 8, 9, 12, 14]
최장 연속 순차의 길이 : 5
>>>
```

프로그래밍 과제 (2- 2)

- 원소의 중복이 없는 정렬된 리스트를 생성하는 프로그램

```
import random
n = int(input('난수의 개수 입력 : '))
numbers = []
for i in range(n):
    x = random.randint(1,n)
    if numbers.count(x) == 0:
        numbers.append(x)
numbers.sort()
print('리스트 : ', numbers)
```

3.4 정렬된 리스트 합병

프로그램 개요

- 자연수를 원소로 가지는 두 개의 정렬된 리스트를 하나의 정렬된 리스트로 합병하는 프로그램 작성
  - 첫 번째 리스트의 원소 개수와 두 번째 리스트의 원소 개수를 입력으로 받은 다음, 리스트의 원소 개수만큼 난수를 발생시켜 초기 리스트 생성
  - 이러한 초기 리스트를 런(run)이라고 부르는데, sort() 메소드를 사용하여 정렬된 두 개의 런을 하나의 런으로 합병하는 프로그램 작성

- 입력 : 두 개의 리스트 r1, r2
- 출력 : 하나의 정렬된 리스트

프로그램 작성

- 두 개의 정렬된 리스트를 하나의 정렬된 리스트로 합병하는 프로그램 작성

```
>>>
첫 번째 리스트의 원소 개수 입력 : 4
두 번째 리스트의 원소 개수 입력 : 5
정렬된 첫 번째 리스트 : [17, 37, 57, 78]
정렬된 두 번째 리스트 : [21, 27, 46, 75, 96]
합병된 리스트 : [17, 21, 27, 37, 46, 57, 75, 78, 96]
>>>
첫 번째 리스트의 원소 개수 입력 : 7
두 번째 리스트의 원소 개수 입력 : 6
정렬된 첫 번째 리스트 : [7, 43, 57, 60, 78, 78, 98]
정렬된 두 번째 리스트 : [10, 10, 47, 53, 57, 63]
합병된 리스트 : [7, 10, 10, 43, 47, 53, 57, 57, 60, 63, 78, 78, 98]
>>>
```

프로그래밍 과제 (1)

- 하나의 리스트를 입력 받아 이것을 두 개로 나눈 다음, 나누어진 두 개의 리스트를 합병하는 프로그램 작성
  - 원소의 개수가 홀수이면 두 번째 리스트의 원소 개수가 첫 번째 리스트의 원소 개수보다 한 개 적어지도록 나눔

```
>>>
리스트의 원소 개수 입력 : 10
최초 리스트 : [37, 44, 27, 98, 43, 97, 37, 76, 64, 10]
정렬된 첫 번째 리스트 : [27, 37, 43, 44, 98]
정렬된 두 번째 리스트 : [10, 37, 64, 76, 97]
합병된 리스트 : [10, 27, 37, 37, 43, 44, 64, 76, 97, 98]
>>>
리스트의 원소 개수 입력 : 11
최초 리스트 : [43, 3, 56, 9, 68, 44, 29, 84, 25, 6, 45]
정렬된 첫 번째 리스트 : [3, 9, 43, 56, 68]
정렬된 두 번째 리스트 : [6, 25, 29, 44, 45, 84]
합병된 리스트 : [3, 6, 9, 25, 29, 43, 44, 45, 56, 68, 84]
>>>
```

프로그래밍 과제 (2)

- 자연수로 이루어진 리스트에서 이미 정렬이 되어 있는 부분을 찾아서 출력해 주는 프로그램 작성

```
>>>
리스트의 원소 개수 입력 : 20
[46, 91, 52, 48, 92, 95, 10, 57, 29, 100, 61, 82, 33, 94, 78, 96, 33, 59, 15, 4]
run1 : 46 91
run2 : 52
run3 : 48 92 95
run4 : 10 57
run5 : 29 100
run6 : 61 82
run7 : 33 94
run8 : 78 96
run9 : 33 59
run10 : 15
run11 : 4
>>>
```

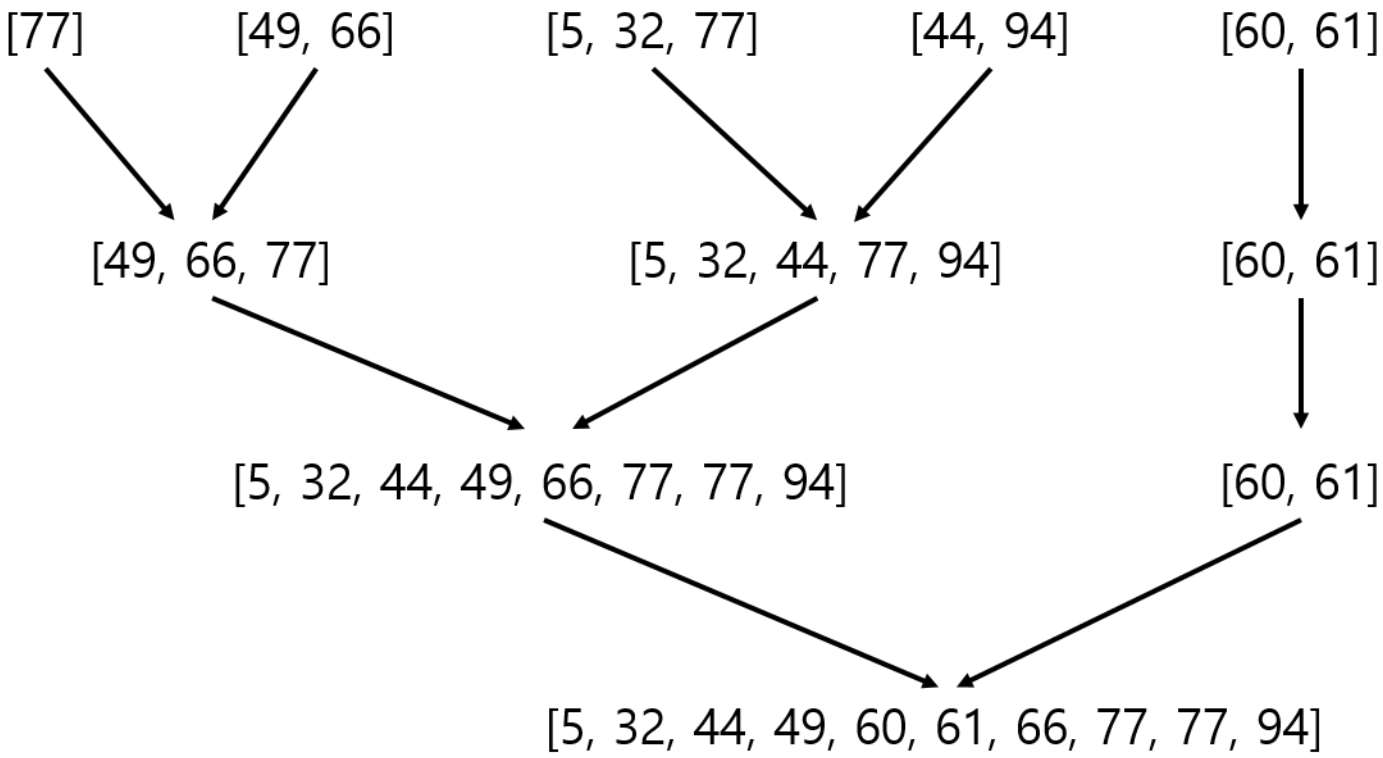
프로그래밍 과제 (3- 1)

- 자연 합병 정렬(Natural Merge Sort)을 수행하는 프로그램을 작성
  - 자연수로 이루어진 리스트에서 이미 정렬이 되어 있는 런을 부분 리스트로 저장한 다음, 두 개의 리스트씩 merge(r1, r2) 함수를 사용하여 차례로 합병하게 되면 최종적으로 한 개의 정렬된 리스트를 만들 수 있음

```
>>>
리스트의 원소 개수 입력 : 10
최초 리스트 : [77, 49, 66, 5, 32, 77, 44, 94, 60, 61]
초기 런 생성 : [[77], [49, 66], [5, 32, 77], [44, 94], [60, 61]]
런 합병 단계 : [[49, 66, 77], [5, 32, 44, 77, 94], [60, 61]]
런 합병 단계 : [[5, 32, 44, 49, 66, 77, 77, 94], [60, 61]]
런 합병 단계 : [[5, 32, 44, 49, 60, 61, 66, 77, 77, 94]]
정렬된 리스트 : [5, 32, 44, 49, 60, 61, 66, 77, 77, 94]
>>>
```

프로그래밍 과제 (3- 2)

- 자연 합병 정렬의 수행 과정



3.5 이진 탐색

프로그램 개요

- 정렬 리스트에서 탐색 키와 동일한 원소가 있는지 비교

> - 입력 : 데이터의 개수와 탐색 키
> - 출력 : 탐색 키와 동일한 원소가 있으면 이 원소의 인덱스를 출력하고,
>
> 없으면 탐색 키와 동일한 원소가 없다고 출력

- 이진 탐색(binary search)
  - 탐색 키와 리스트의 원소를 비교할 때, 리스트의 중간에 있는 원소와 제일 먼저 비교
  - 중간 원소가 탐색 키보다 크면 왼쪽 절반에 대해 이진 탐색을 수행하고, 중간 원소가 탐색 키보다 작으면 오른쪽 절반에 대해 이진 탐색을 수행함

프로그램 작성 (1)

- 중복된 원소 없이 오름차순으로 정렬된 값을 가지는 N개의 난수로 이루어진 리스트에 대해 이진 탐색을 수행하는 프로그램을 작성

```
>>>
데이터의 개수 입력 : 10
리스트 : [4, 7, 8, 11, 14, 18, 19, 23, 28, 29]
탐색 키 입력 : 28
리스트의 9 번째 위치에서 탐색 키와 동일한 원소 발견
>>>
데이터의 개수 입력 : 15
리스트 : [3, 5, 8, 13, 18, 21, 26, 30, 35, 40, 41, 45, 49, 51, 56]
탐색 키 입력 : 50
탐색 키와 동일한 원소가 없습니다.
>>>
```

프로그램 작성 (2)

- 중복된 원소 없이 오름차순으로 정렬된 값을 가지는 N개의 난수로 이루어진 리스트를 생성하는 코드

```
import random
N = int(input('데이터의 개수 입력 : '))
data = 0
numbers = []
for i in range(N):
    data += random.randint(1, 5)
    numbers.append(data)
print('리스트 : ', numbers)
```

프로그래밍 과제 (1)

- 보간 탐색(interpolation search)
  - 데이터의 크기에 따라 처음 탐색하는 위치를 조정해 주는 탐색 기법
  - 일반적으로 사전이나 전화번호부에서 단어나 이름을 찾을 때는 이진 탐색과 같이 중간부터 찾을 수도 있지만 알파벳의 순서에 따라 처음부터 찾는 것이 빠를 수도 있고, 뒤에서부터 찾는 것이 빠를 수도 있음
  - 예를 들어, 영어 사전에서 'zoo'를 찾을 때는 사전의 중간을 먼저 펼치는 것보다는 사전의 뒤쪽을 먼저 탐색하는 것이 효과적임
- 보간 탐색 프로그램은 mid의 값을 구하는 다음 수식을 제외하고는 이진 탐색과 동일함

$$mid = low + (high - low) \times (key - nums[low])/(nums[high] - nums[low])$$

- mid의 값을 구하는 부분을 위의 수식으로 변경하여 보간 탐색을 수행하는 프로그램 작성

프로그래밍 과제 (2)

- 리스트 : [3, 6, 10, 14, 18, 23, 25, 29]
- 탐색 키 : 10

low	high-low	key-nums[low]	nums[high]-nums[low]	a	mid
0	7	7	26	1	1
2	5	4	23	0	2

※ $a = (high - low) \times (key - nums[low])/(nums[high] - nums[low])$

3.6 파스칼의 삼각형

프로그램 개요 (1)

- 높이 h를 입력 받아 높이가 h인 파스칼의 삼각형을 출력하는 프로그램 작성

- 입력 : 높이 h
- 출력 : 높이가 h인 파스칼의 삼각형

프로그램 개요 (2)

- 파스칼의 삼각형(Pascal's triangle)
  - 이항 계수(binomial coefficient)를 삼각형 모양의 기하학적 형태로 배열한 것
  - 이항 계수는 이항식인 $(a+b)^n$에 $n = 0, 1, 2, 3, \cdots$을 대입하여 전개했을 때의 계수를 나타냄
  - $(a+b)^1 = a+b$, $(a+b)^2 = a^2 + 2ab + b^2$, $(a+b)^3 = a^3 + 3a^2b + 3ab^2 + b^3$이 되는데, 이렇게 전개한 식의 계수를 다음 그림과 같이 나타낸 것

$n = 0$ 1

$n = 1$ 1 1

$n = 2$ 1 2 1

$n = 3$ 1 3 3 1

$n = 4$ 1 4 6 4 1

… …

프로그램 개요 (3)

- 파스칼의 삼각형을 만드는 방법
  - 먼저 첫 번째 줄에는 1을 씀
  - 그 다음 줄을 만들려면, 바로 위의 왼쪽 숫자와 오른쪽 숫자를 더함. 예를 들어, 네 번째 줄의 숫자 1과 3을 더하여 다섯 번째 줄의 4가 만들어짐
- 파스칼의 법칙
  - n 번째 줄의 k 번째 값을 a_{nk}라고 하면

$$a_{n1} = 1$$

$$a_{nn} = 1$$

$$a_{nk} = a_{n-1k-1} + a_{n-1k} \; (n, k \geq 0)$$

프로그램 작성

- 높이 h를 입력 받아 높이가 h인 파스칼의 삼각형을 출력하는 프로그램 작성

```
>>>
높이 h 입력 : 10
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
>>>
```

프로그래밍 과제

- 행 번호를 입력 받아 파스칼의 삼각형에서 입력 받은 행만을 출력해 주는 프로그램 작성

```
>>>
행 번호 입력 : 5
1 4 6 4 1
>>>
행 번호 입력 : 10
1 9 36 84 126 126 84 36 9 1
>>>
```