A1.

```
# 이진 검색 작동확인
A = Data(size=128)
print(A.data)
Loc, k = A.binary_search(10)
print(Loc, k)
```

```
[1, 1, 2, 2, 3, 6, 7, 8, 8, 8, 9, 9, 10, 11, 11, 12, 14, 15, 16, 16, 16, 20, 21, 21, 21, 23, 23, 25,
12 6
```

위 사진과 같이, 객체지향방법으로 구현한 이진검색이 제대로 동작하는 것을 확인할 수 있다.

(관찰)

```
############################## A1 ########################################

C = []  # n값에 따른 1000개의 문제 p의 평균 데이터 비교 횟수

for n in [128, 256, 512]:
    S = Data(size=n)
    c = 0

    for i in range(1000):
        x = randrange(1, n+1) # 하나의 문제 pi를 생성
        loc, ki = S.binary_search(x)    # x를 찾을때까지의 비교 횟수 ki를 계산
        c += ki

    c = int(c/1000) # 평균 데이터 비교횟수 c
    C.append(c)

print(C)    # n=128, n=256, n=512일때의 c값
```

```
[5, 6, 7]
```

1000개의 문제 p에 대한 평균 비교횟수 c는

n = 128일 때 5

n = 256일 때 6

n = 512일 때 7이다.

n과 c와의 관계는 $n = 2^k$ ($k \geq 3$ *인 자연수*) 일 때 $c = 2^{k-2}$인 것을 관찰할 수 있다. 다만 n의 수가 작을 때는 생성되는 S의 데이터 분포에 따라 오차가 있을 수

있다.

```
############################## A1 ########################################

C = []  # n값에 따른 1000개의 문제 p의 평균 데이터 비교 횟수

for n in [8, 16, 32, 64, 128, 256, 512, 1024]:
    S = Data(size=n)
    c = 0

    for i in range(1000):
        x = randrange(1, n+1) # 하나의 문제 pi를 생성
        loc, ki = S.binary_search(x)    # x를 찾을때까지의 비교 횟수 ki를 계산
        c += ki

    c = int(c/1000) # 평균 데이터 비교횟수 c
    C.append(c)

print(C)    # n=128, n=256, n=512일때의 c값
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
```

n이 2^3부터 2^10까지의 실행결과

A2.

```
############################ A2 ##################################

# merge_sort 검증
data = [8, 3, 15, 2, 9, 1, 5, 7, 4, 16, 10, 11, 12, 13, 6, 14]
extra_mem_size = merge_sort(data)
print(data)
# 추가로 필요한 메모리의 크기
print(extra_mem_size)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
31
```

문제 2에서 주어진 데이터를 올바르게 정렬하고, 추가로 필요한 메모리의 크기(≒2n)까지 계산한 결과.

(공간복잡도가 2n이 되는 merge sort알고리즘을 사용함. py파일 참조)

두 번째 merge sort 검증 결과. 랜덤으로 생성한 리스트에 대해서도 비내림차순으로 정렬된 결과를 보여준다.