병합정렬이란 하나의 리스트를 두 개의 균등한 크기로 분할하고 분할된 부분 리스트를 정렬한 다음, 두 리스트를 합하여 전체가 정렬된 리스트를 만드는 방법으로 분할 정복 기법과 재귀 알고리즘을 이용한 정렬 알고리즘이다. 퀵 정렬 알고리즘은 하나의 리스트를 피벗을 기준으로 두 개의 비균등한 크기로 분할하고 분할된 부분 리스트를 정렬한 다음, 두 리스트의 정렬된 부분을 합하여 전체 리스트가 정렬되게 하는 방법이며 분할 정복 기법과 재귀 알고리즘을 이용한 정렬 알고리즘이다.

<병합정렬과 퀵 정렬을 이용해 birthday.in을 정렬한 결과 비교>

Birthday.in의 데이터가 Yeji 1201, Suji 0804, Sujeong 0905라고 한다면,

병합정렬의 경우: Yeji 1201, Suji 0804 와 Sujeong 0905의 2개의 부분 배열로 나뉘게 된다. 두 부분 배열을 정렬하여 Suji 0804, Yeji 1201과 Sujeong 0905를 얻게 되고 부분 배열을 통합하여 Suji 0804, Sujeong 0905, Yeji 1201를 얻게 된다.

퀵 정렬의 경우: Yeji 1201를 피벗으로 선택하고 low는 Suji 0804, high는 Sujeong 0905가 된다. Low를 피벗보다 큰 항목까지 이동하고 high를 피벗보다 작은 항목까지 이동하게 되는데 현재는 바로 low와 high가 엇갈리기 때문에 high와 피벗의 값을 교환해 Suji 0804, Yeji 1201, Sujeong 0905의 배열을 얻는다. 이후 나눠진 두 배열에 대해 재귀적으로 퀵 정렬을 수행하면 Suji 0804, Sujeong 0905, Yeji 1201를 얻게 된다.

결론: 따라서 병합정렬과 퀵 정렬의 정렬 결과가 같다.

<소스코드에 사용한 정렬 알고리즘과 선택 이유>

퀵 정렬 사용하였다. 퀵 정렬은 정말 특이한 경우가 아닌 이상 평균적으로 매우 빠른 수행 속도를 자랑하는 정렬 방법이기 때문이다.

<프로그래밍 없이 "같은 생일자를 찾는 알고리즘"을 정렬하기 전과 후 비교>

정렬되지 않은 생일 목록을 사용한 알고리즘 :생일 목록에서 맨 처음에 있는 사람을 선택해 선택한 사람과 다른 모든 사람의 생일을 비교하며 동일한 생일을 가진 사람이 있는지 확인한다. 만약 동일한 생일을 가진 사람이 있다면 두 사람의 이름과 생일을 반환한다. 그러나 존재하지 않는다면 두번째 사람을 선택하여 위 과정을 반복하며 같은 생일을 가진 사람을 찾는다. 본 과정을 같은 생일을 가진 사람이 나타날 때까지 반복한다.

정렬된 생일 목록을 사용한 알고리즘 :생일이 오름차순으로 정렬되어 있다고 가정한다면, 생일 목록을 순차적으로 탐색하며 동일한 생일을 가진 사람을 찾고 두 사람의 이름과 생일을 반환한다.

<correctness & efficiency>

1.(정렬 전 데이터에 대해) 같은 생일자를 찾는 알고리즘

correctness: 입력 배열이 n개의 요소로 구성되어 있고 이 알고리즘이 n개의 요소에 대해 정확하게 작동한다고 가정한다면 배열에 새로운 값을 추가할 때 해당 값과 이전에 선택된 값을 비교해 동일한 생일을 가진 사람이 있는지 확인한다. 따라서 새로운 값을 추가한 배열에서도 정확성이 유지된다.

Efficiency: (pseudocode 사용)

```
for from i=0 to arr_size-1:
        for from j=i+1 to arr_size:
            if arr[i].birthday==arr[j],birthday:
                print arr[i], arr[j] ->이중 for문이 사용되기 때문에 시간 복잡도는 O(n^2)이다.
```

2.(정렬 후 데이터에 대해) 같은 생일자를 찾는 알고리즘

Correctness:입력배열의 크기가 n개이고 이 알고리즘이 n개의 값에 정확하게 작동된다면, 배열을 정렬 후 인접한 값을 비교해 같은 생일자를 확인한다. 정렬된 배열에선 같은 생일자는 인접하게 되므로 정확성이 유지된다.

Efficiency: (pseudocode사용)

```
Arr=quicksort(arr)
For from i=0 to arr_size-2:
    If(arr[i].birthday==arr[i+1].birthday:
        Print arr[i],arr[i+1] ->본 알고리즘의 시간복잡도는 정렬 알고리즘이 무엇이냐에 따라 다르겠
```

지만 만약 입력 데이터 자체가 정렬이 되어 있다면 O(n)의 시간복잡도를 갖게 된다.

3.병합정렬

Correctness: 귀납법을 사용하여 증명한다.

Base: P(1)은 참이다. -> 원소가 1개이면 항상 정렬되어 있는 상태이다.

Step: P(n-1) -> P(n)이 True라면, 다시말해 n/2범위에서 병합정렬함수를 호출하고 정렬이 성공한다면 P(n)일 때 병합정렬은 성공한다.

Result: P(n)은 모든 자연수 n에 대해서 참이다.

따라서 병합정렬은 정확하다.

Efficiency: (pseudocode사용)

```
MergeSort(list, left, right)
If left < right
  mid = (left+right)/2;
  MergeSort(list, left, mid);
  MergeSort(list, mid+1, right);   -> 따라서 T(n)=2T(n/2)+n이고 이를 계산하면
  Merge(list, left, mid, right);     -> O(nlogn)의 시간복잡도를 갖게 된다.
```

4.퀵정렬

```
  quicksort(list, left, right)     -> Correctness: 피벗을 기준으로 배열을 분할하고 각
    if(left < right)                -> 부분 배열에 대해 재귀적으로 정렬을 수행하여 정확성 유지.
        q=partition(list, left, right);
        quicksort(list, left, q-1);
        quicksort(list, q+1, right);
  partition(list, left, right)
      x=list[right];
      i=left-1;
      for(j=left to right-1)
        if(list[j] <= x)
            i=i+1;
            exchange list[i] with list[j];     -> Efficiency:
       exchange list[i+1] with list[right]; ->pseudocode를 통한 평균적인 시간복잡도는 O(nlogn)이
```

다.