

1 longestIndexArray(T, left, right)

```
longestIndexArray(T, left, right)
1  if left < right
2      then m = ⌊(left + right)/2⌋
3      longestIndexArray(T, left, mid)
4      longestIndexArray(T, mid+1, right)
5      longestIndex(T, left, right)
```

주어진 data 배열을 배열의 인덱스를 이용하여 두 부분으로 나누는 메소드이다.
이후 나누어진 두 부분을 merge하게 된다.

2 longestIndex(T, left, right)

```
create stack TupleStack
longestIndex(T, left, right)
1  for i ← left to right
2      do if TupleStack empty
3          then TupleStack ← T[i]
4      else
5          t1 ← top data in TupleStack
6          t2 ← T[i]
7          if compareTupleData(t1, t2)
8              then TupleStack ← t2
```


주어진 data 배열을 merge하는 과정이다. 배열을 두 부분으로 나누었을때 왼쪽부분부터 비교하게 되므로 stack이 empty상태이면 첫번째 data는 stack에 push하게 된다. 이후 data는 stack의 가장 마지막 data와 비교하여 push를 결정한다.

3 compareTupleData(lt, rt)

```
compareTupleData(lt, rt)
1  y1 ← right data of Tuple lt
2  y2 ← right data of Tuple rt
3  if y1 < y2
4      x1 ← left data of Tuple lt
5      x2 ← left data of Tuple rt
6  |  if x2 - x1 == 1
7      then true;
8  return false;
```

두 data를 비교하는 메소드이다. data가 tuple의 형태로 저장했기 때문에 (index, 주어진 data)의 형태에서 data를 비교하여 stack에 있는 data값보다 큰 경우 index까지 비교하게 된다. 이때 index차이가 1보다 크다면 연속된 증가가 아니므로 stack에 push하지 않는다.

4 결과



```
<terminated> Mainjava (3) [Java Application] C:\Program Files\Java\jre1.8.0_301\bin\javaw.exe (2021. 10. 1. 오후 3:43:53)
6
2
0
```

주어진 data의 크기를 n 이라고 하자. 전체 n 크기를 푸는데 걸리는 시간 $T(n)$ 이라하자.

1번 메소드에서 두 부분으로 나누게 되므로 $T(n/2)$ 가 2개가 된다. 2번 메소드에서 나눈 2부분을 합쳐서 풀게 되고 반복문에서 n 크기 만큼 시간이 걸리므로 $\Theta(n)$ 이라 할 수 있다.

따라서 $T(n) = 2T(n/2) + \Theta(n) = 2T(n/2) + n$ 이다. 이를 이용해서 시간복잡도를 구하면 다음과 같다.

$$\begin{aligned}
 T(n) &= 2T(n/2) + n && \text{Assume } n = 2^k \\
 &= 4T(n/4) + 2n \\
 &= 8T(n/8) + 3n \\
 &\quad \vdots \\
 &= 2^k T(n/2^k) + kn \\
 &= nT(1) + n\log n \in \Theta(n\log n)
 \end{aligned}$$