최장 증가 부분수열 LIS

Longest Increasing Subsequence

LIS

LIS란 ?

증가 부분수열

증가 부분수열

증가 부분수열: [10, 20], [10, 20, 40, 50, 70], [20, 40, 50], [10, 20, 25, 30, 70] …

증가 부분수열

증가 부분수열: [10, 20], [10, 20, 40, 50, 70], [20, 40, 50], [10, 20, 25, 30, 70] …

수열을 이루는 원소가 실제 배열 내에서 인접해있지 않아도 된다!

최장 증가 부분수열

최장 증가 부분수열: [10, 20, 25, 30, 70]

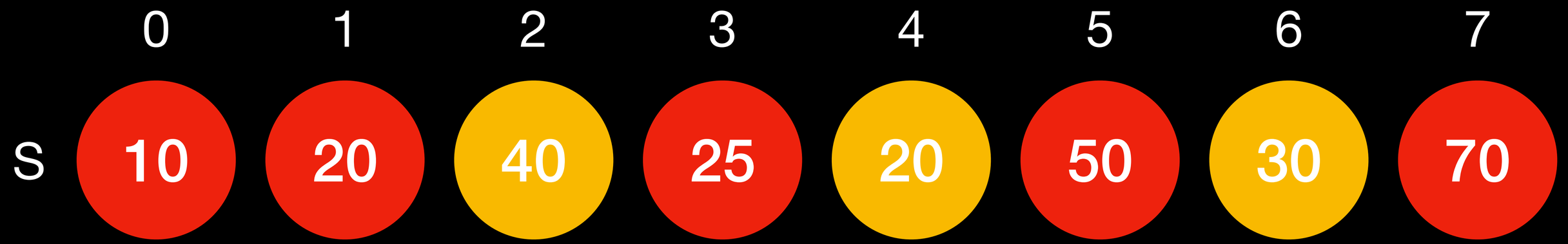
최장 증가 부분수열의 길이: 5

최장 증가 부분수열

최장 증가 부분수열: [10, 20, 25, 50, 70]

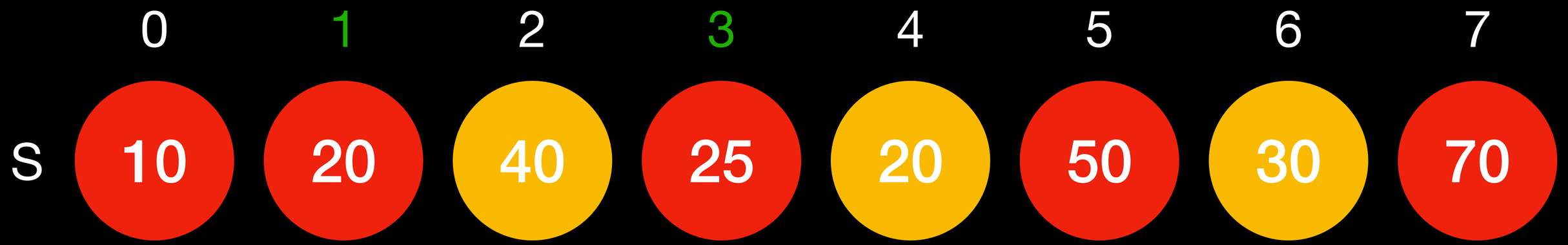
최장 증가 부분수열의 길이: 5

최장 증가 부분수열



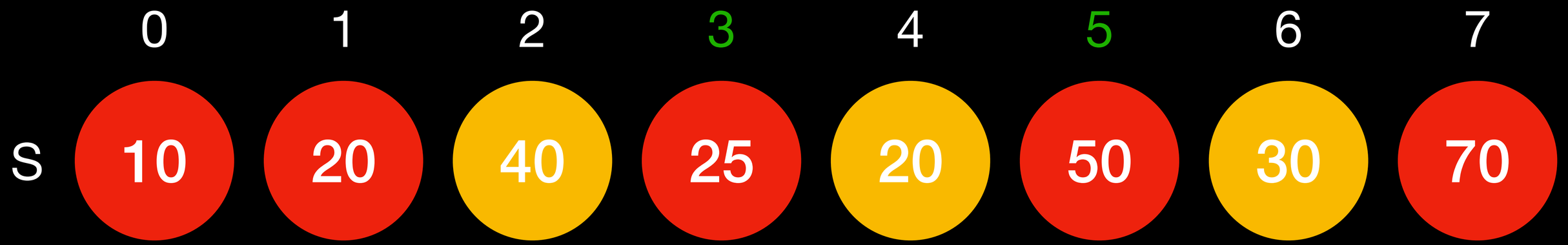
0 <= i, j <= | S | 인 i,j에 대하여 ***i < j*** 이면, ***S[i] < S[j]*** 를 만족한다.

최장 증가 부분수열



i = 1, j = 3 일 때, S[1](=20) < S[3](=25)

최장 증가 부분수열



i = 3, j = 5 일 때, S[3](=25) < S[5](=50)

구현

3가지 방법

1.
완전탐색

```
int lis(vector<int> arr) {

  if(arr.empty()) return 0;

  int len = arr.size(), answer = 1;

  for(int i=0; i < len; i++) {

    vector<int> next;

    for(int j=i+1; j < len; j++) {

      if(arr[j] > arr[i]) next.push_back(arr[j]);

    }

    answer = max(answer, 1 + lis(next));

  }

  return answer;

}
```

1.
완전탐색

```
int lis(vector<int> arr) {

  if(arr.empty()) return 0;

  int len = arr.size(), answer = 1;

  for(int i=0; i < len; i++) {

    vector<int> next;

    for(int j=i+1; j < len; j++) {

      if(arr[j] > arr[i]) next.push_back(arr[j]);

    }

    answer = max(answer, 1 + lis(next));

  }

  return answer;

}
```

1.
완전탐색

“i < j 일 때, arr[i] < arr[j] 라면 증가수열의 원소가 될 수 있음”

```
int lis(vector<int> arr) {

  if(arr.empty()) return 0;

  int len = arr.size(), answer = 1;

  for(int i=0; i < len; i++) {

    vector<int> next;

    for(int j=i+1; j < len; j++) {

      if(arr[j] > arr[i]) next.push_back(arr[j]);

    }

    answer = max(answer, 1 + lis(next));

  }

  return answer;

}
```

1.
완전탐색

“재귀호출을 통해 증가수열의 길이 계산 ”

```
int lis(vector<int> arr) {

  if(arr.empty()) return 0;

  int len = arr.size(), answer = 1;

  for(int i=0; i < len; i++) {

    vector<int> next;

    for(int j=i+1; j < len; j++) {

      if(arr[j] > arr[i]) next.push_back(arr[j]);

    }

    answer = max(answer, 1 + lis(next));

  }

  return answer;

}
```

1.
완전탐색

시간복잡도 : O(2^N)

BOJ에 이 방법으로 구현해서 제출하면 시간초과

조금 더 효율적인 방법이 필요하다!

2.
동적계획법(DP)

동적계획법(DP)

DP[i] : 수열의 i번째 값을 마지막 원소로 갖는 최장증가수열의 길이

1. DP배열 초기화

동적계획법(DP)

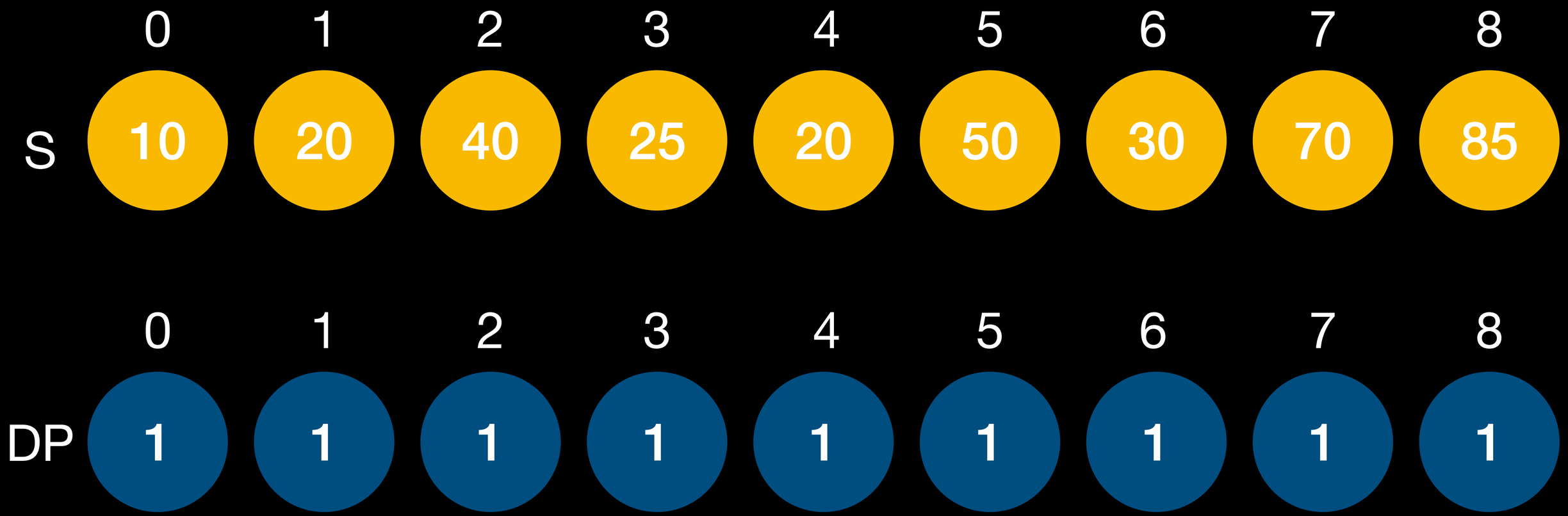
DP[i] : i번째 값을 마지막 원소로 갖는 최장증가수열의 길이

DP[i] = *(DP[j] + 1 > DP[i] && S[j] < S[i]) ? DP[j] + 1 : DP[i]*

```
for(int i=1;i<=n;i++) {
  dp[i] = 1;
  for(int j=1;j<=i;j++) {
    if(arr[j] < arr[i] && dp[j]+1 > dp[i]) {
      dp[i] = dp[j]+1;
    }
  }
}
```

2. DP배열 점화식

동적계획법(DP)

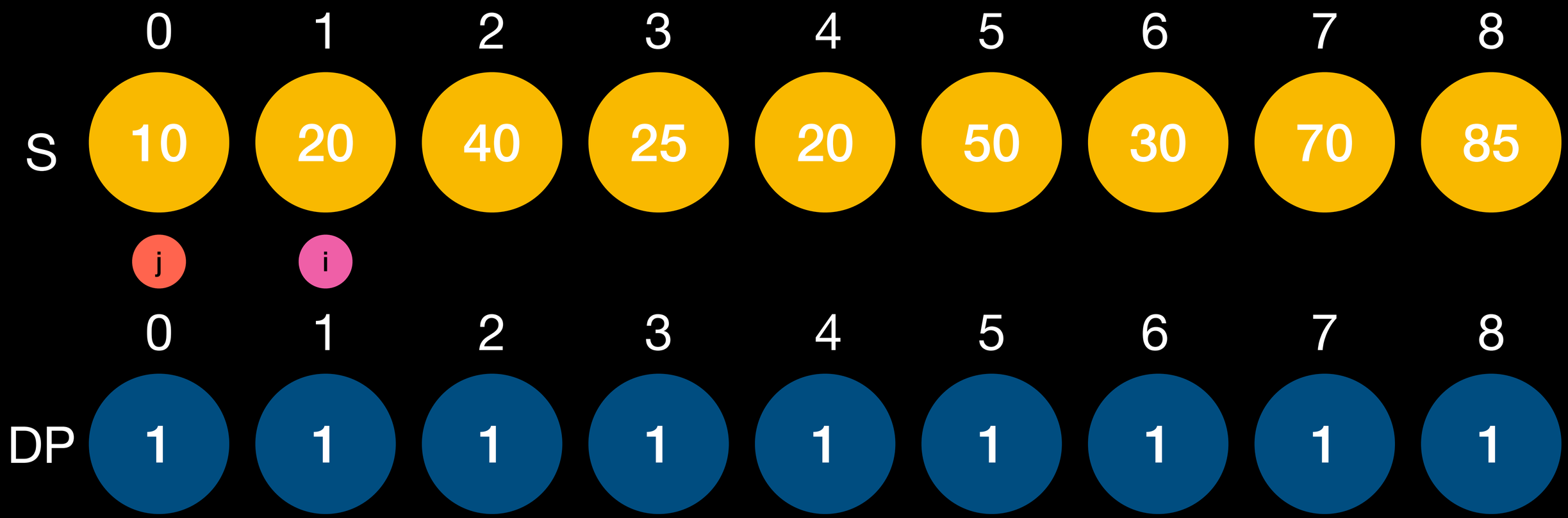


3. DP배열 채우기

동적계획법(DP)

DP[1] = *(DP[0] + 1 > DP[1] && S[0] < S[1]) ? DP[0] + 1 : DP[1]*

조건을 만족하므로, ***DP[1] = 2*** 갱신

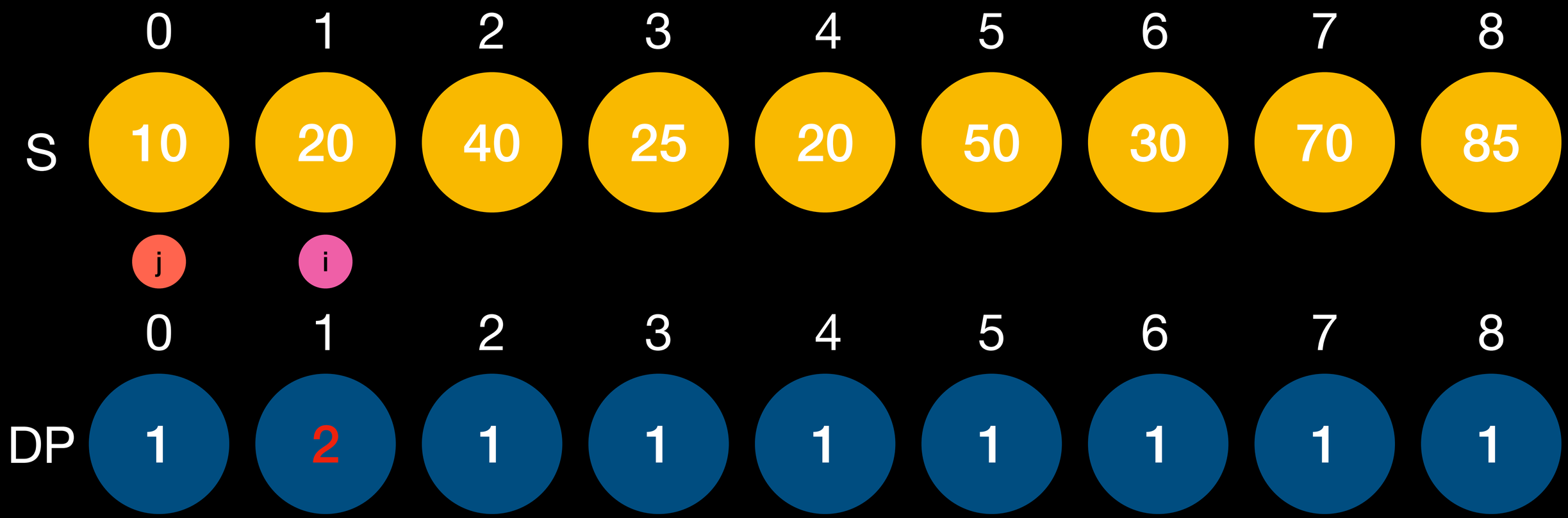


3. DP배열 채우기

동적계획법(DP)

DP[1] = *(DP[0] + 1 > DP[1] && S[0] < S[1]) ? DP[0] + 1 : DP[1]*

조건을 만족하므로, ***DP[1] = 2*** 갱신

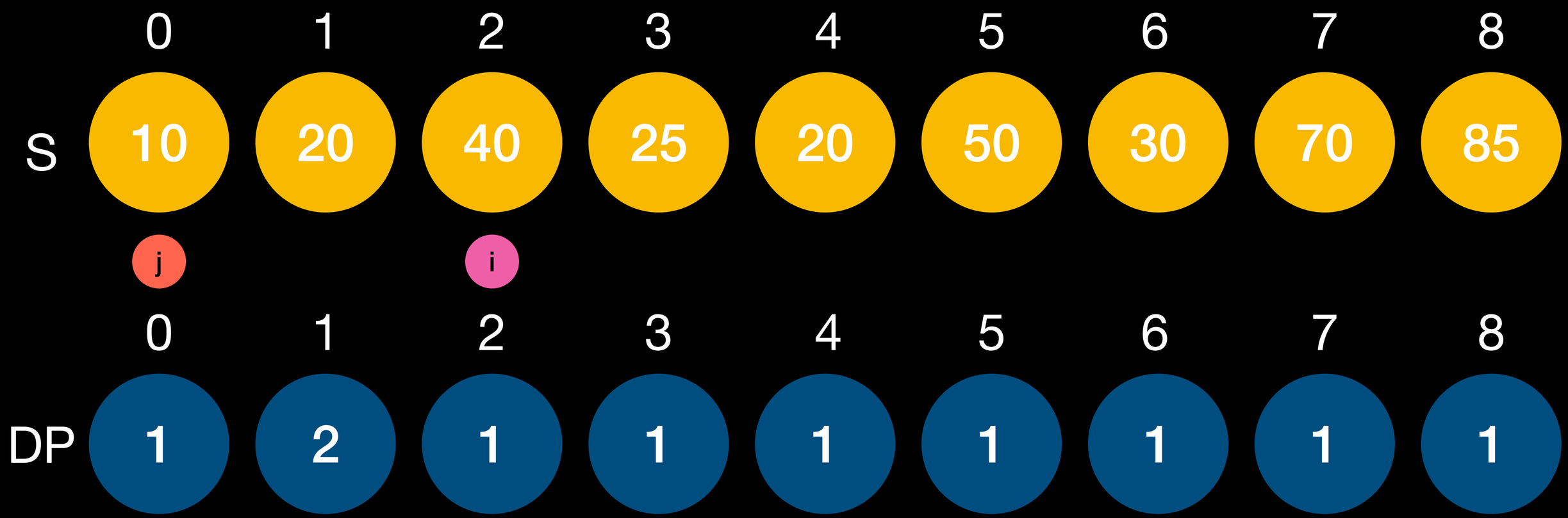


3. DP배열 채우기

동적계획법(DP)

DP[2] = *(DP[0] + 1 > DP[2] && S[0] < S[2]) ? DP[0] + 1 : DP[2]*

조건을 만족하므로, ***DP[2] = 2*** 갱신

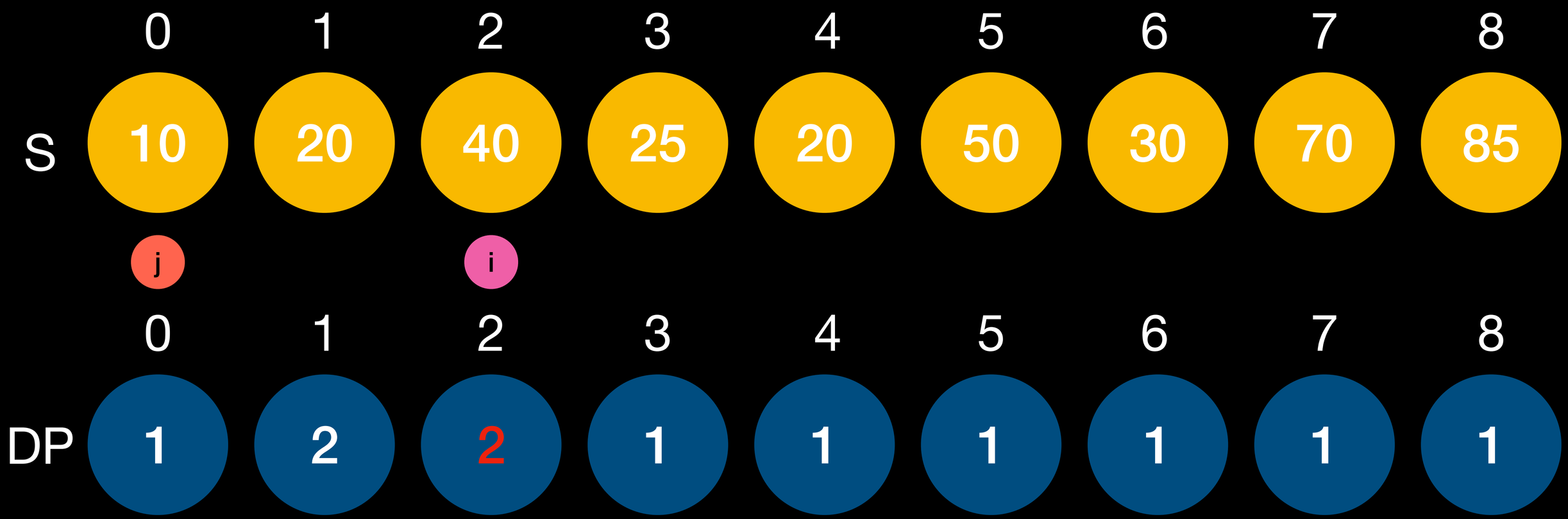


3. DP배열 채우기

동적계획법(DP)

DP[2] = *(DP[0] + 1 > DP[2] && S[0] < S[2]) ? DP[0] + 1 : DP[2]*

조건을 만족하므로, ***DP[2] = 2*** 갱신

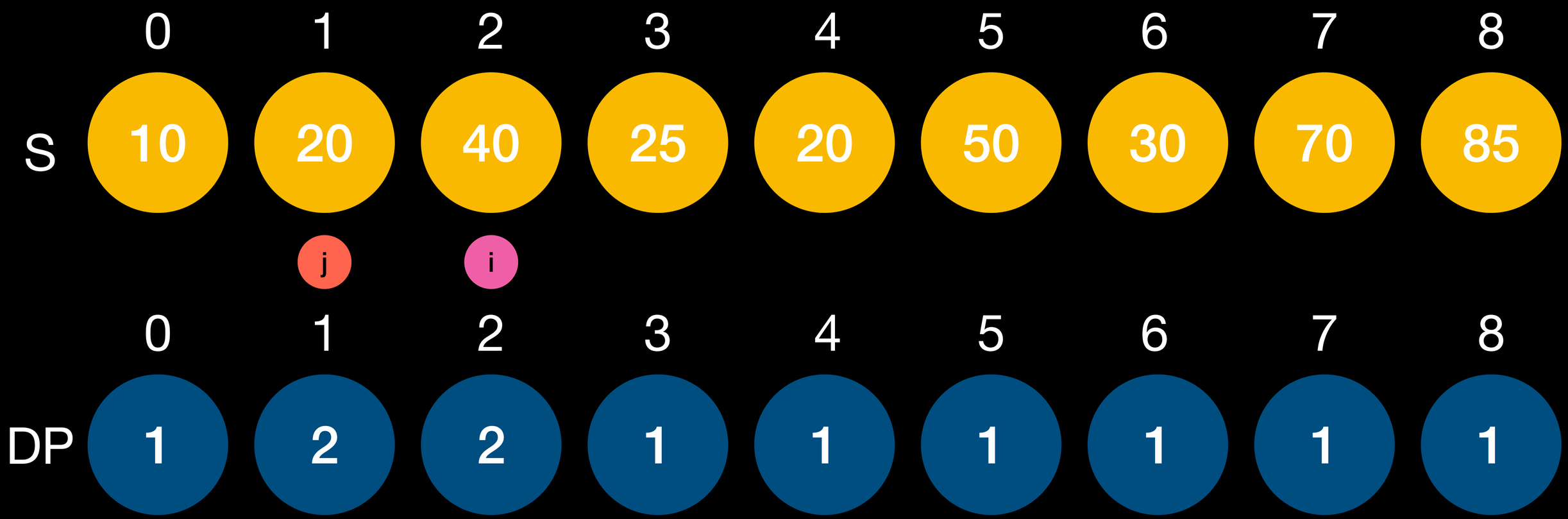


3. DP배열 채우기

동적계획법(DP)

DP[2] = *(DP[1] + 1 > DP[2] && S[1] < S[2]) ? DP[1] + 1 : DP[2]*

조건을 만족하므로, ***DP[2] = 3*** 갱신

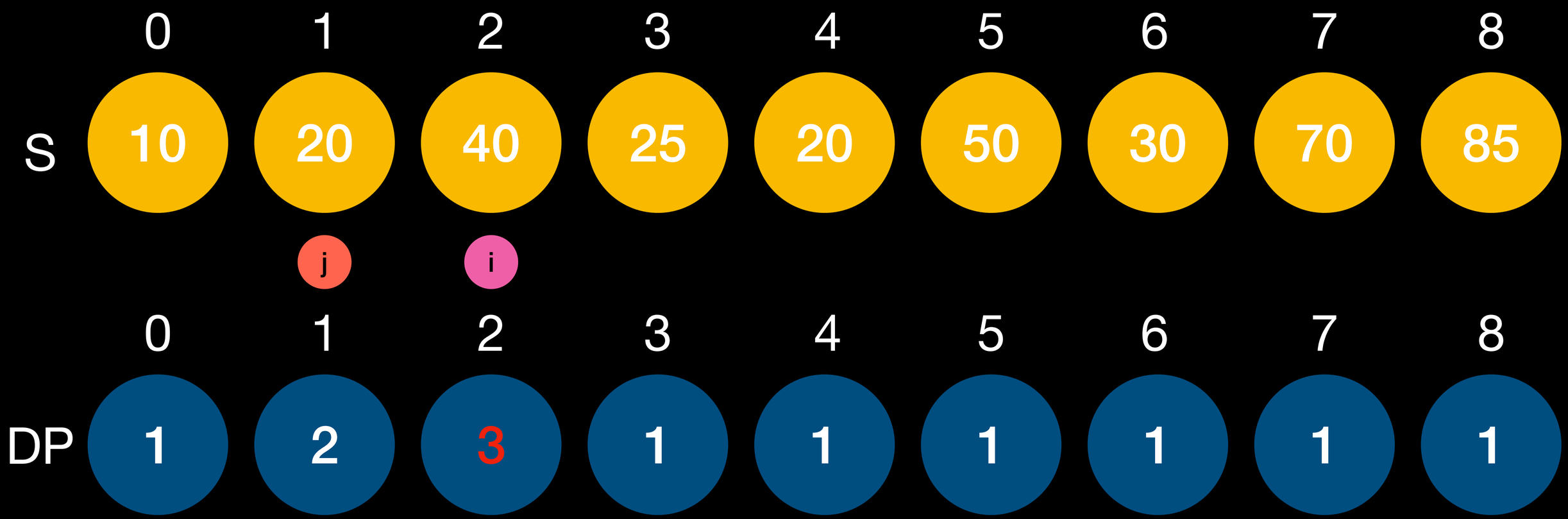


3. DP배열 채우기

동적계획법(DP)

DP[2] = *(DP[1] + 1 > DP[2] && S[1] < S[2]) ? DP[1] + 1 : DP[2]*

조건을 만족하므로, ***DP[2] = 3*** 갱신

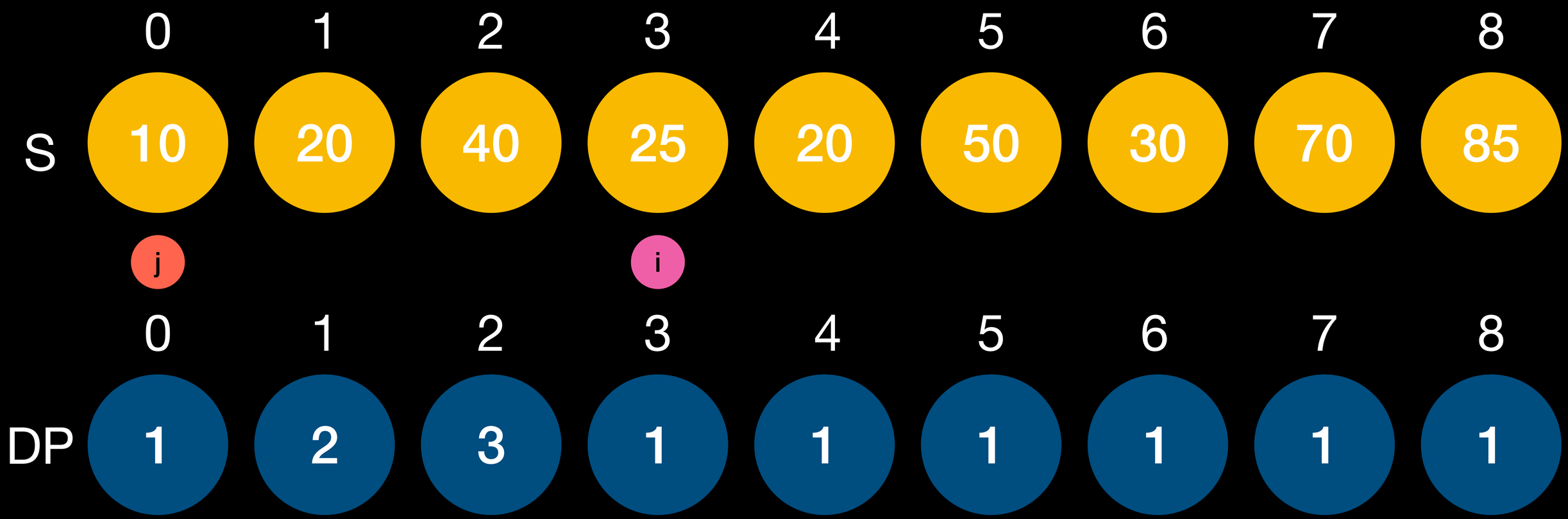


3. DP배열 채우기

동적계획법(DP)

DP[3] = *(DP[0] + 1 > DP[3] && S[0] < S[3]) ? DP[0] + 1 : DP[3]*

조건을 만족하므로, ***DP[3] = 2*** 갱신

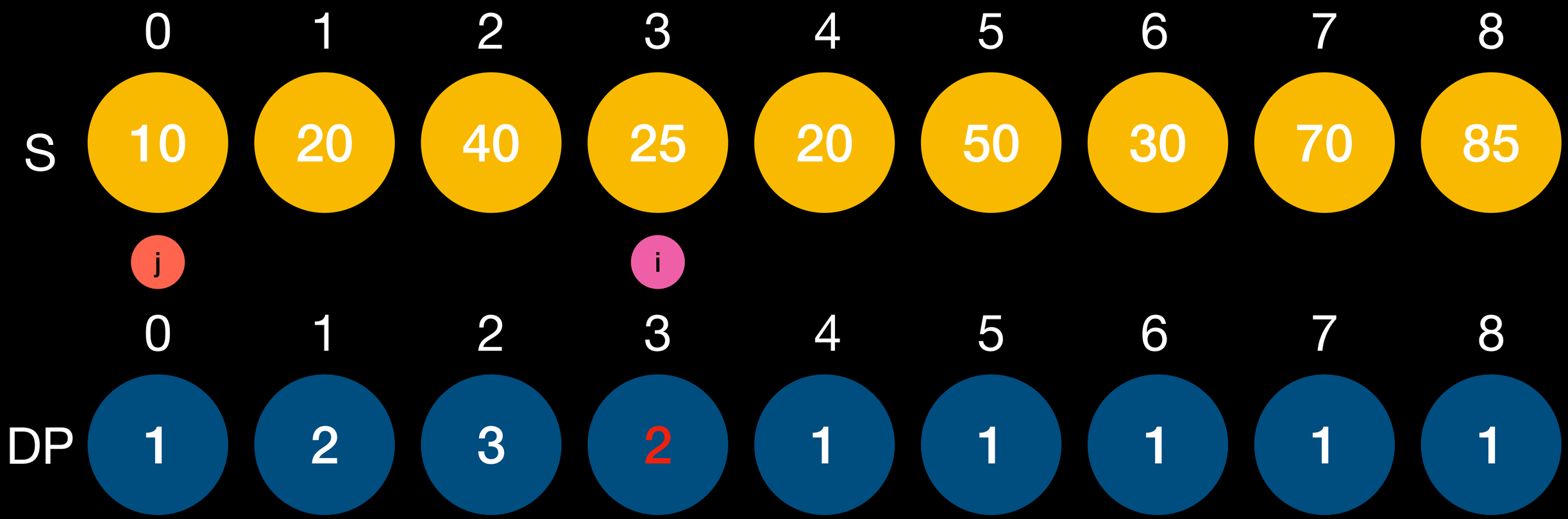


3. DP배열 채우기

동적계획법(DP)

DP[3] = *(DP[0] + 1 > DP[3] && S[0] < S[3]) ? DP[0] + 1 : DP[3]*

조건을 만족하므로, ***DP[3] = 2*** 갱신

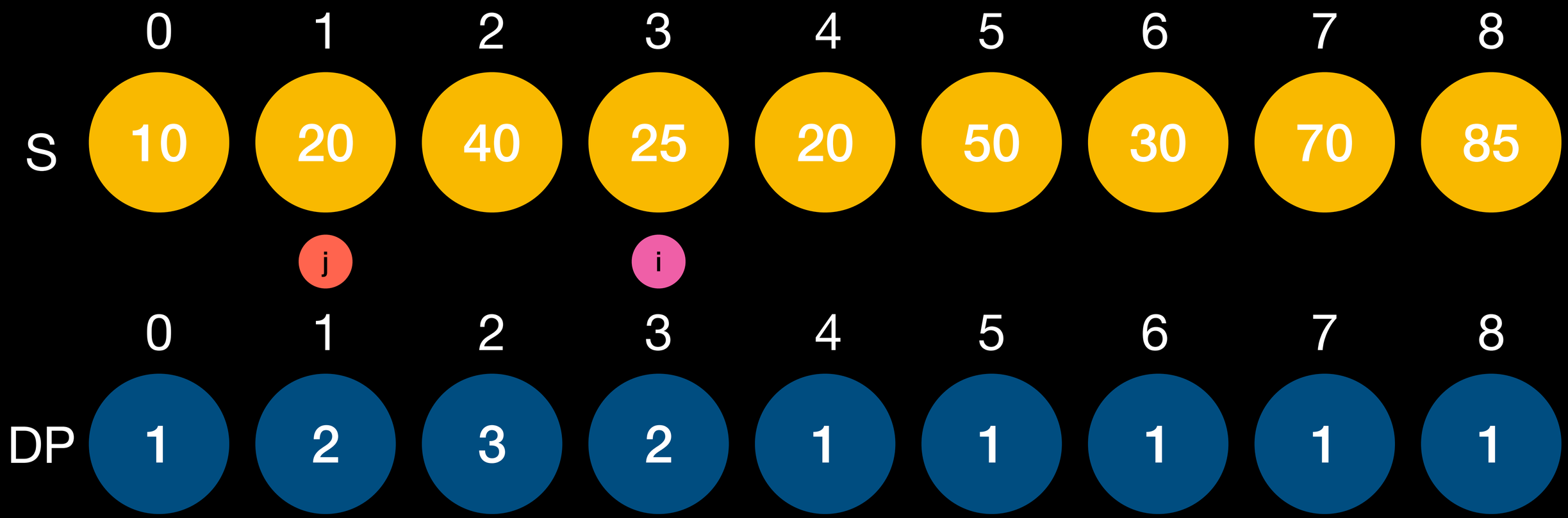


3. DP배열 채우기

동적계획법(DP)

DP[3] = *(DP[1] + 1 > DP[3] && S[1] < S[3]) ? DP[1] + 1 : DP[3]*

조건을 만족하므로, ***DP[3] = 3*** 갱신

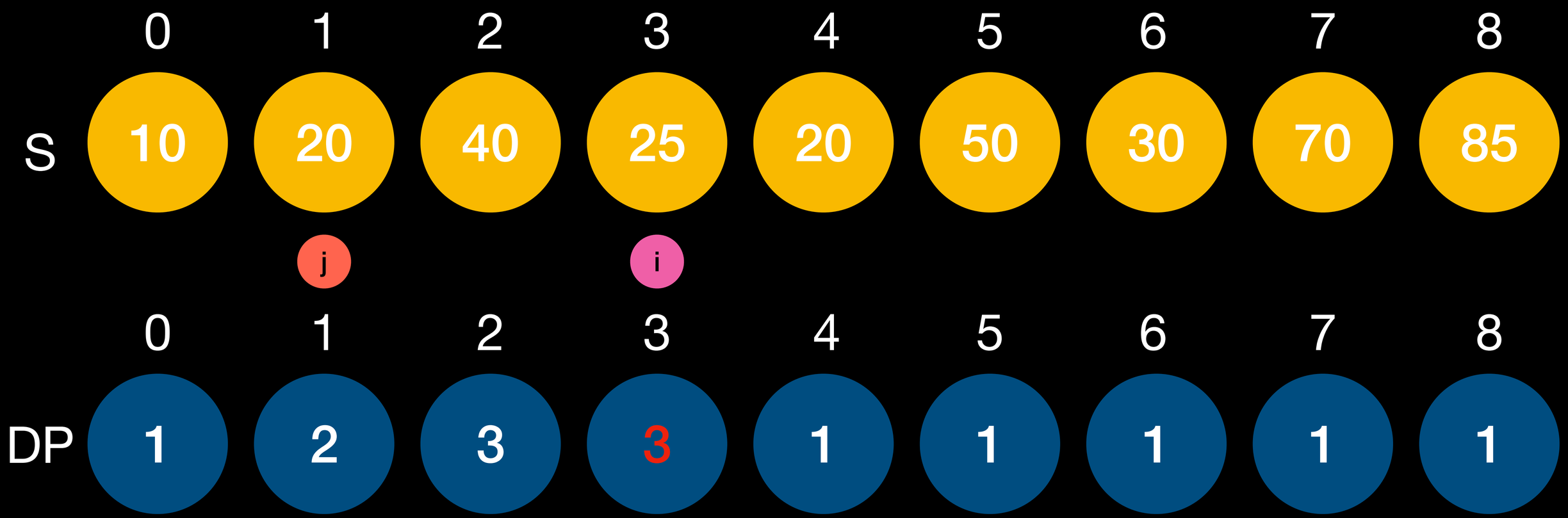


3. DP배열 채우기

동적계획법(DP)

DP[3] = *(DP[1] + 1 > DP[3] && S[1] < S[3]) ? DP[1] + 1 : DP[3]*

조건을 만족하므로, ***DP[3] = 3*** 갱신

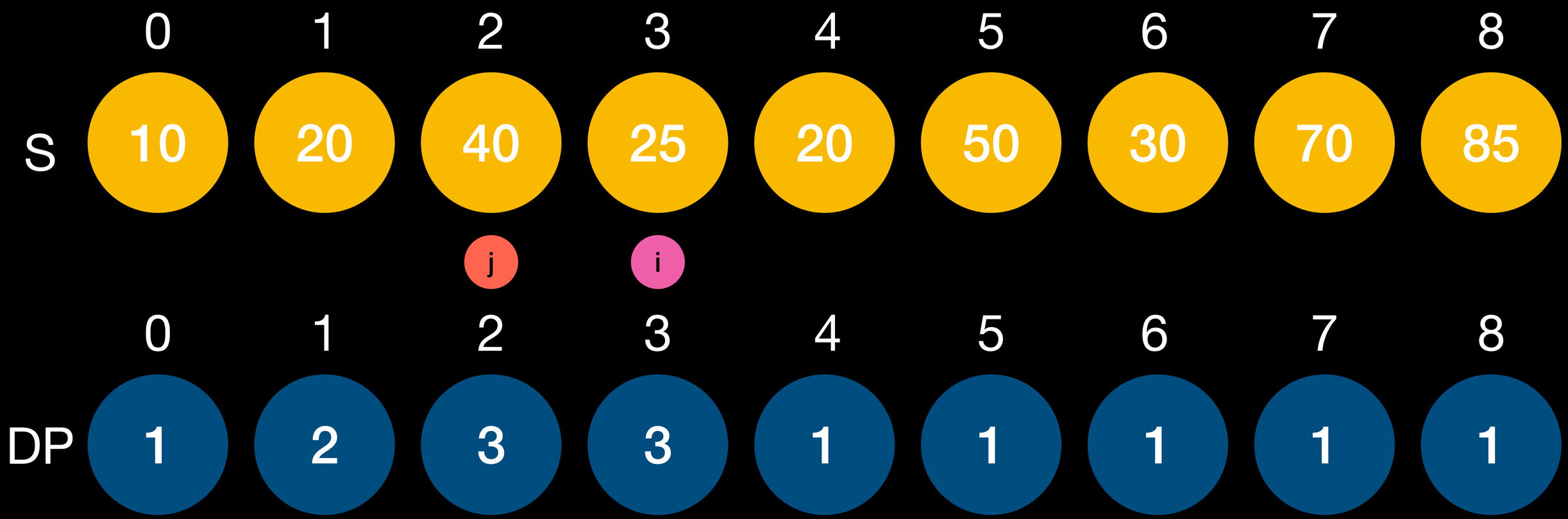


3. DP배열 채우기

동적계획법(DP)

DP[3] = *(DP[2] + 1 > DP[3] && S[2] < S[3]) ? DP[2] + 1 : DP[3]*

*S[2] < S[3]*을 만족하지 않으므로, 갱신하지 않음.

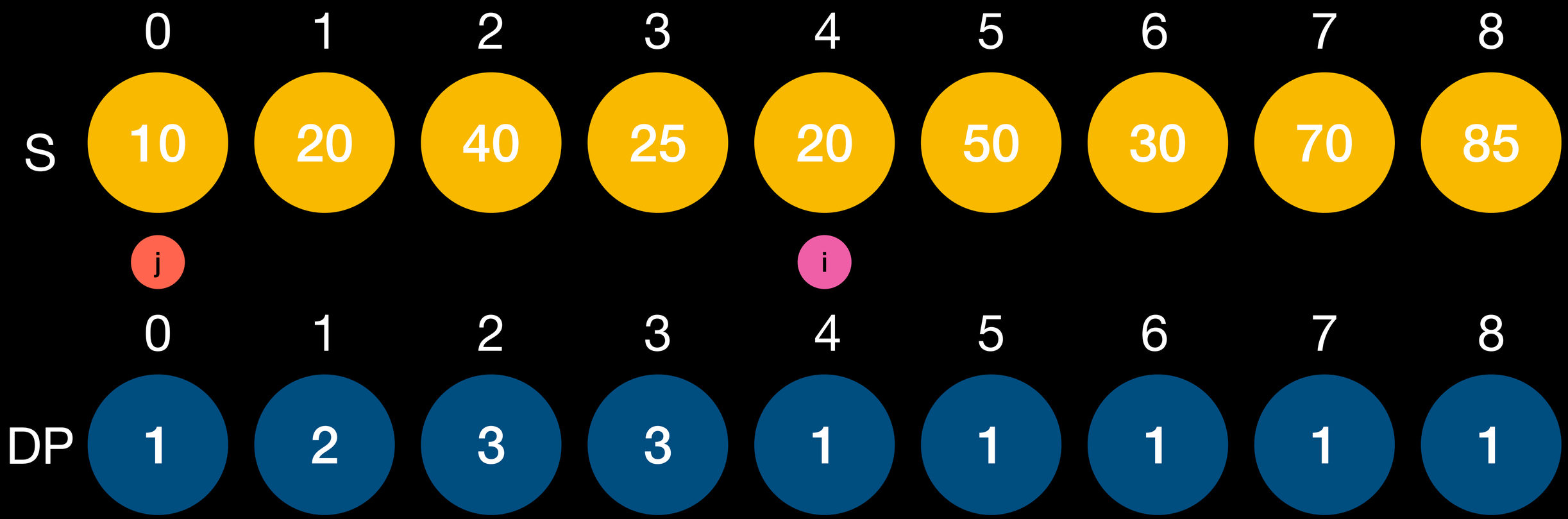


3. DP배열 채우기

동적계획법(DP)

DP[4] = *(DP[0] + 1 > DP[4] && S[0] < S[4]) ? DP[0] + 1 : DP[4]*

조건을 만족하므로, ***DP[4] = 2*** 갱신

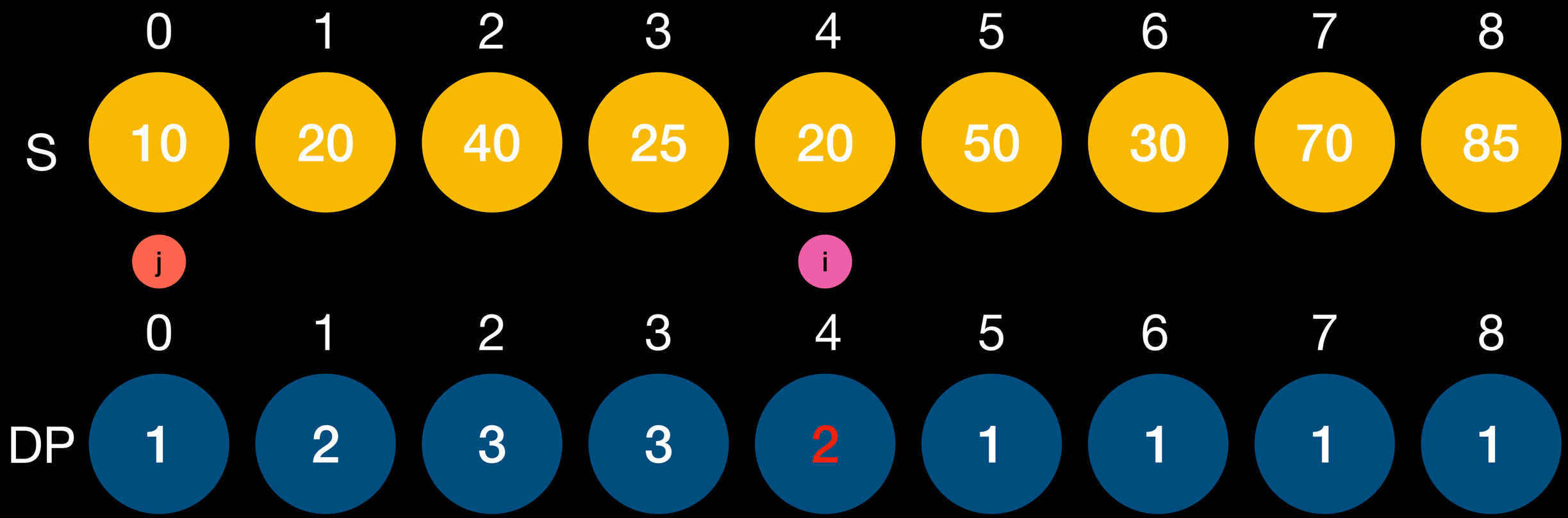


3. DP배열 채우기

동적계획법(DP)

DP[4] = *(DP[0] + 1 > DP[4] && S[0] < S[4]) ? DP[0] + 1 : DP[4]*

조건을 만족하므로, ***DP[4] = 2*** 갱신

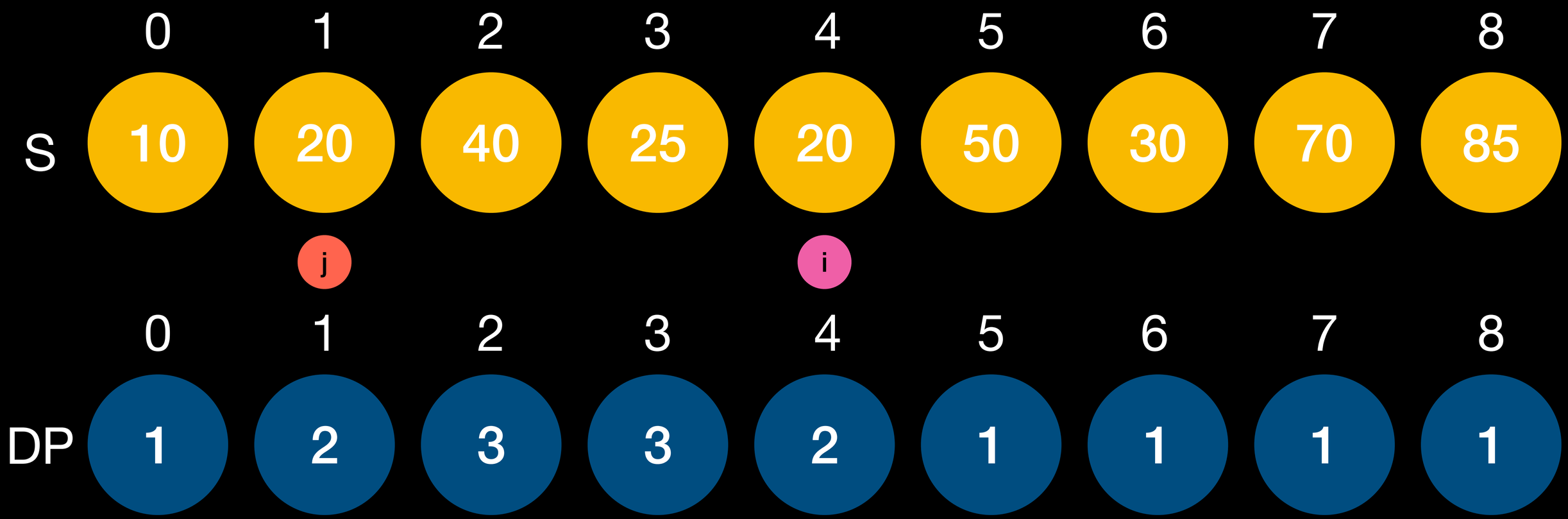


3. DP배열 채우기

동적계획법(DP)

DP[4] = *(DP[1] + 1 > DP[4] && S[1] < S[4]) ? DP[1] + 1 : DP[4]*

*S[1] < S[4]*를 만족하지 않으므로 갱신 *X, j*가 *2~3*일 때도 동일.

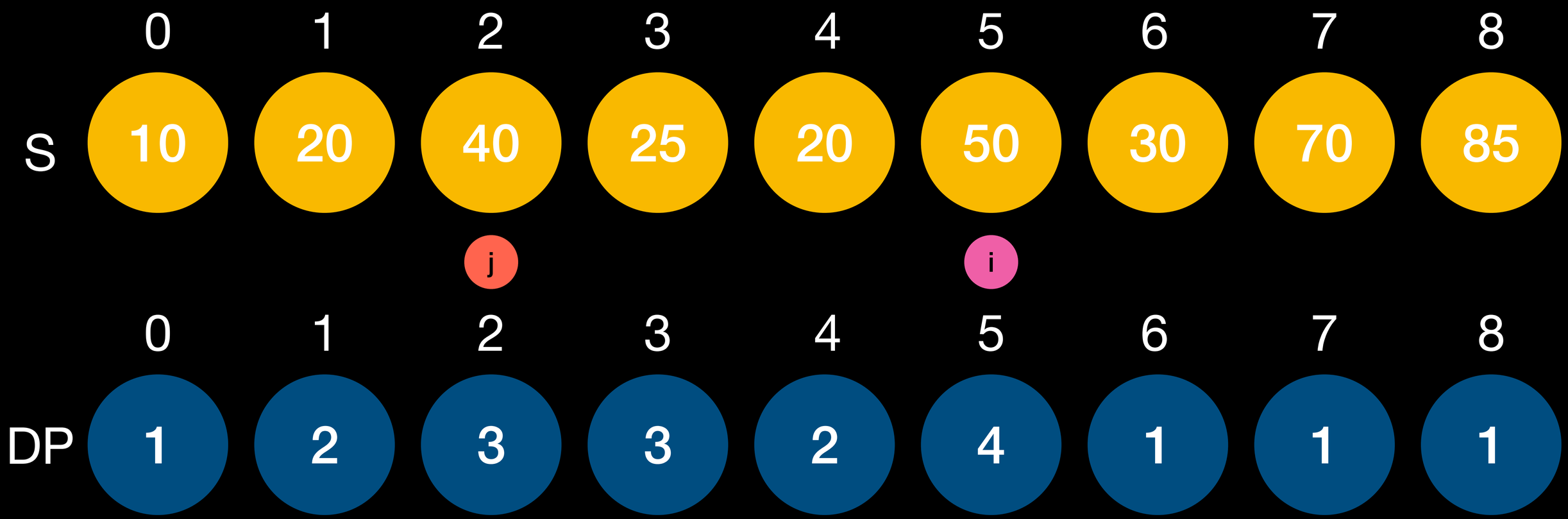


3. DP배열 채우기

동적계획법(DP)

DP[5] = *(DP[2] + 1 > DP[5] && S[2] < S[5]) ? DP[2] + 1 : DP[5]*

같은 방식으로 *j=2*까지 진행했을 때, *DP[5]*는 *4*까지 업데이트됨

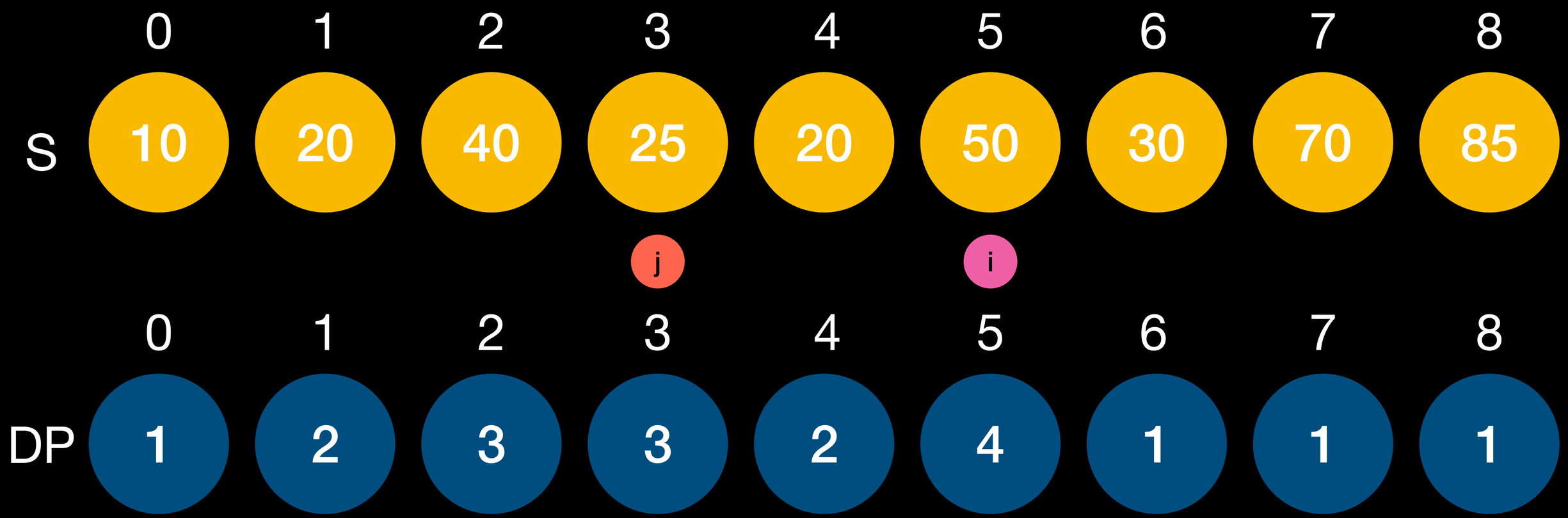


3. DP배열 채우기

동적계획법(DP)

DP[5] = (*DP[3] + 1* > *DP[5]* && *S[3]* < *S[5]*) ? *DP[3] + 1* : *DP[5]*

DP[3] + 1 > DP[5] 조건을 만족하지 않으므로 갱신 *X*, *j*가 *4*일때도 마찬가지

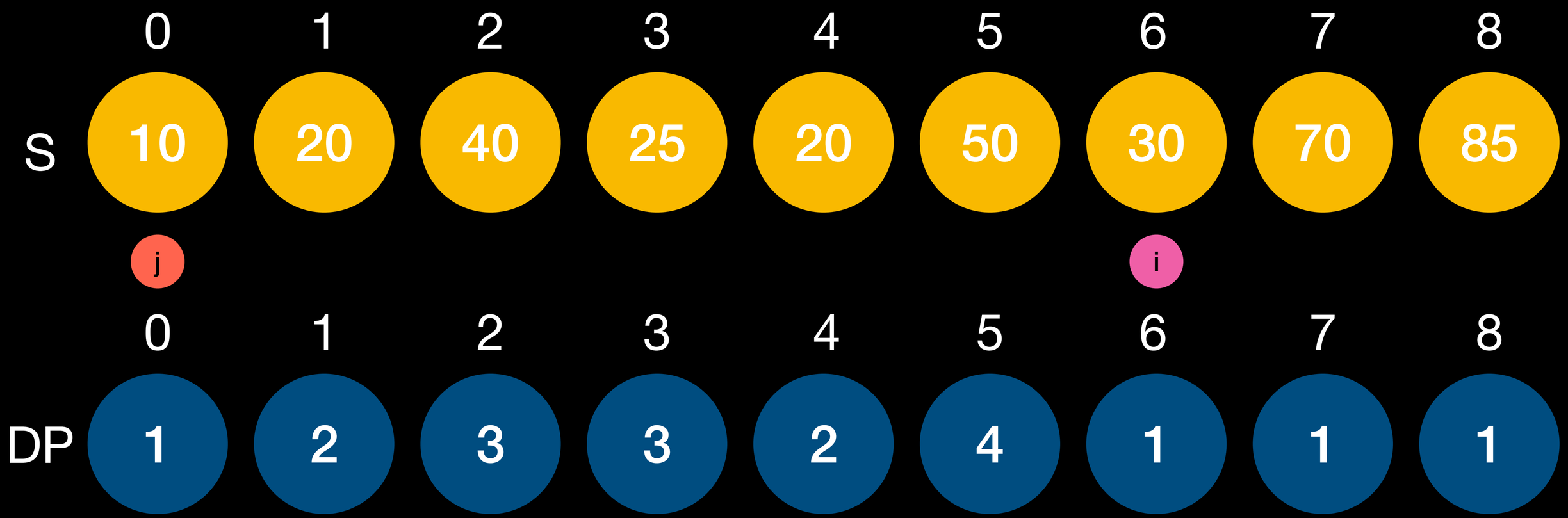


3. DP배열 채우기

동적계획법(DP)

DP[5] = (*DP[3] + 1* > *DP[5]* && *S[3]* < *S[5]*) ? *DP[3] + 1* : *DP[5]*

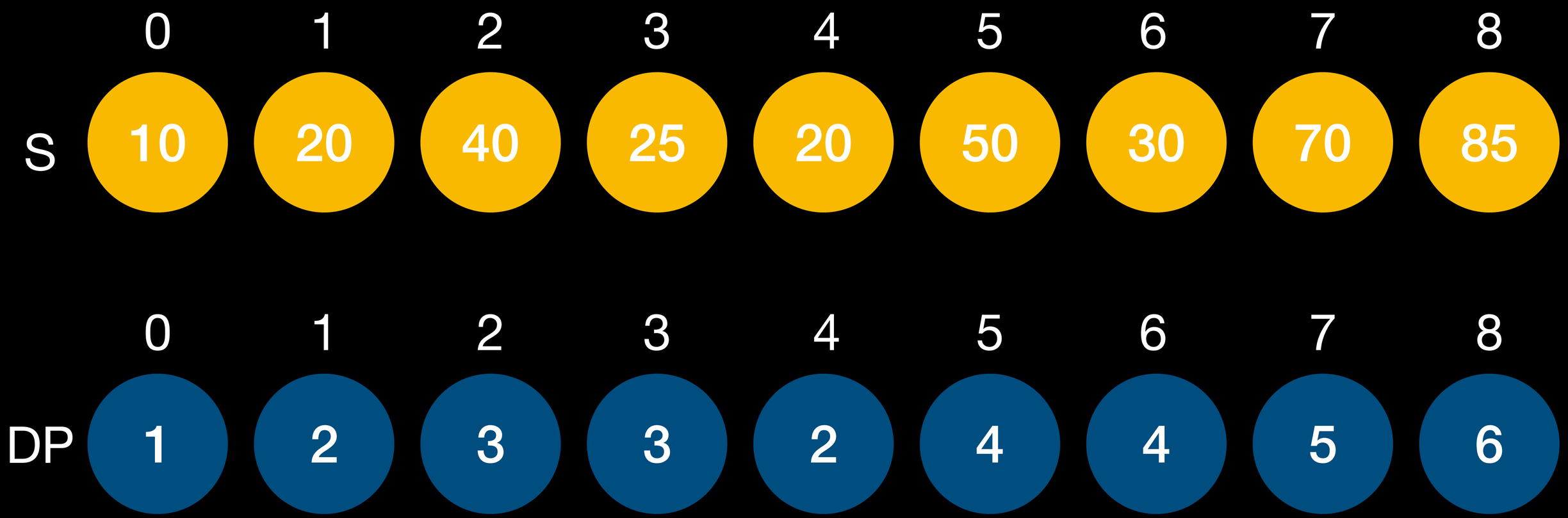
DP[3] + 1 > DP[5] 조건을 만족하지 않으므로 갱신 *X*, *j*가 *4*일때도 마찬가지



3. DP배열 채우기

동적계획법(DP)

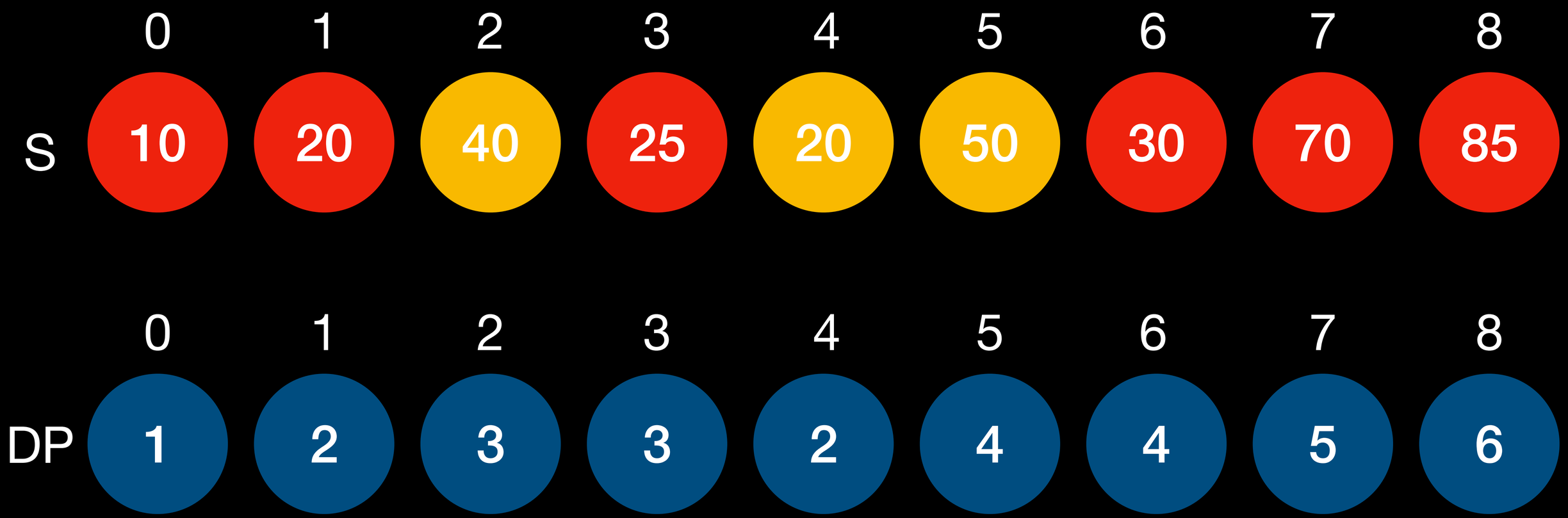
이 후 과정은 생략한다...
완성된 DP배열 내에서 최댓값이 바로 LIS값이 된다.



3. DP배열 채우기

동적계획법(DP)

이 후 과정은 생략한다...
완성된 DP배열 내에서 최댓값이 바로 LIS값이 된다.



3. DP배열 채우기

12015	시간 초과

```
for(int i=1;i<=n;i++) {
  dp[i] = 1;
  for(int j=1;j<=i;j++) {
    if(arr[j] < arr[i] && dp[j]+1 > dp[i]) {
      dp[i] = dp[j]+1;
    }
  }
}
```

동적계획법(DP)

시간복잡도 : O(N^2)

N의 값이 10만보다 커지는 경우에는 사용하기 어려움

조금 더 … 더 …효율적인 방법이 필요하다!

3.
이분탐색
(lower_bound)

3.
이분탐색
(lower_bound)

동적계획법을 더 최적화 한 방식

O(N log N)의 시간복잡도를 가짐

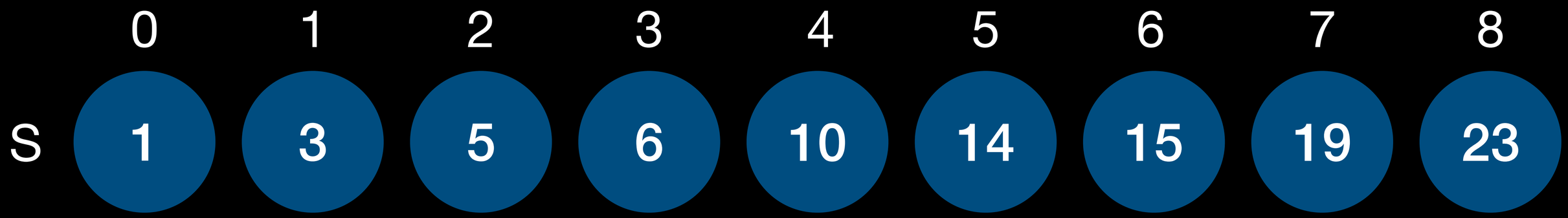
Lower bound란?

이분탐색을 기반으로 한 것.

Lowerbound(k) : 정렬된 배열에서 k이상인 값이 처음으로 등장하는 인덱스

Lowerbound(4) = ?

Lowerbound(12) = ?



1. lower bound

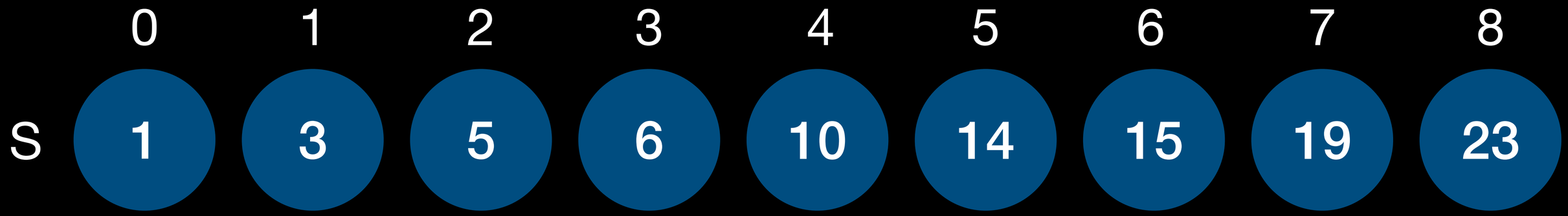
Lower bound란?

이분탐색을 기반으로 한 것.

Lowerbound(k) : 정렬된 배열에서 k이상인 값이 처음으로 등장하는 인덱스

Lowerbound(4) = 2

Lowerbound(12) = 5



LIS (with lowerbound)

아이디어

LIS의 마지막 원소가 작을수록 더 긴 LIS를 생성할 수 있다!

구현

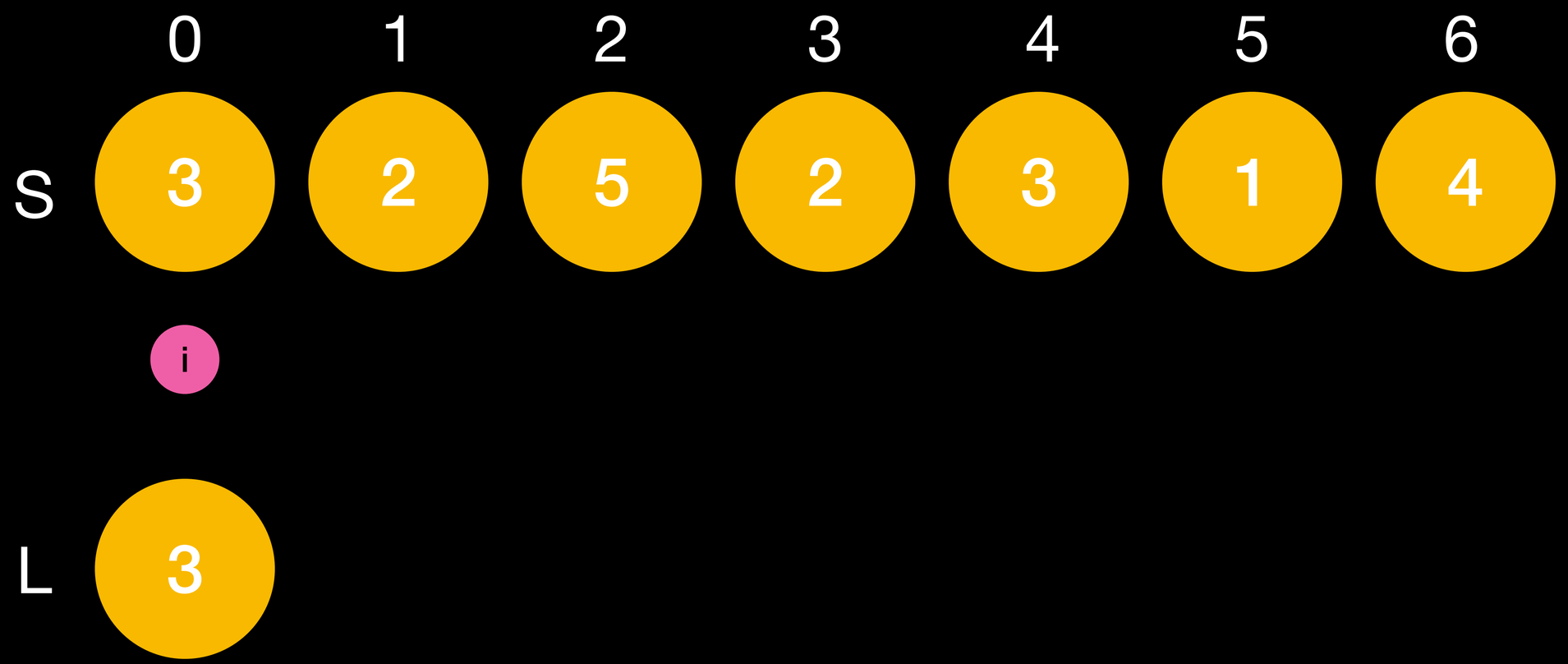
L[i] : 길이가 i인 LIS 중 마지막 원소 값이 가장 작은 LIS의 마지막 원소값 -> L 배열 완성하기.
L은 처음에 비어있는 배열임.

1. 기존 수열을 탐색하며 L의 마지막 원소보다 크다면 L의 뒤에 삽입한다.
2. 크지 않다면 해당 값의 Lowerbound인 곳에 삽입한다.

2. algorithm

LIS (with lowerbound)

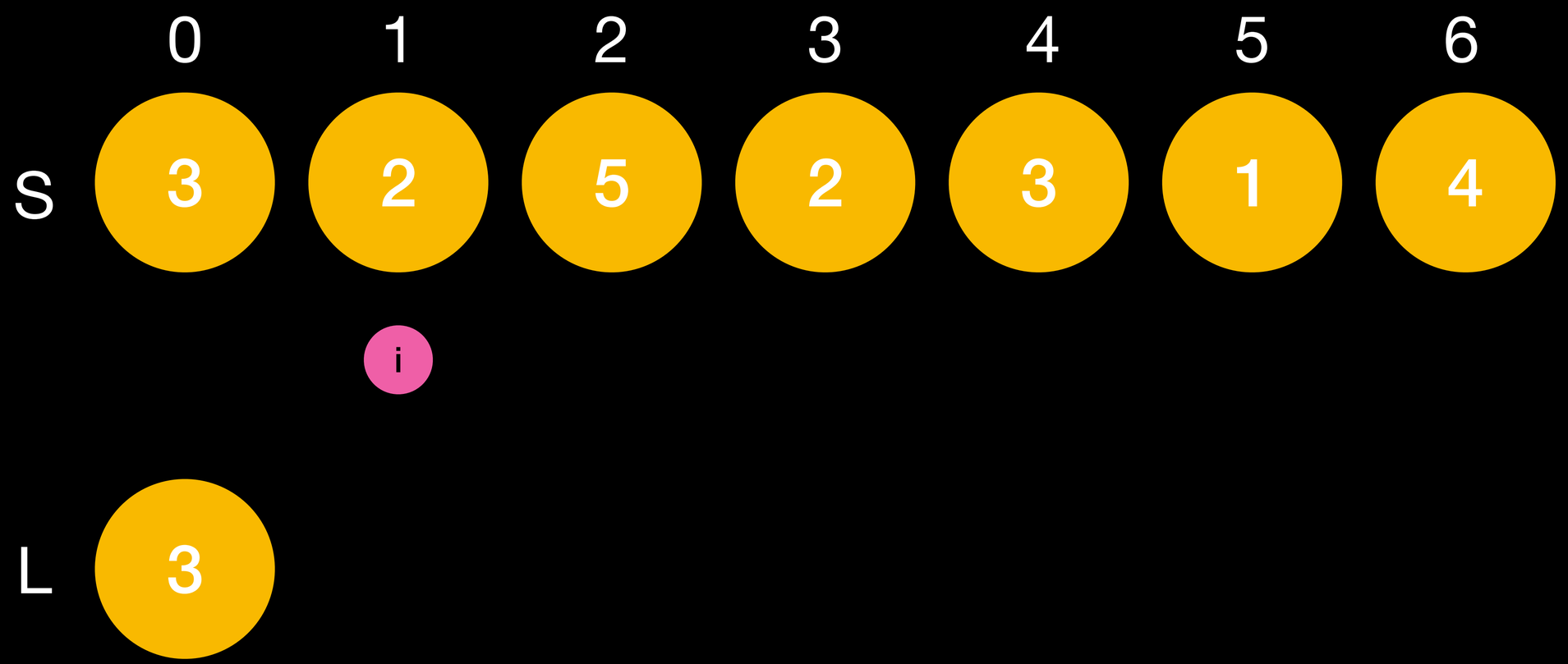
1. 기존 수열을 순회하며 L의 마지막 원소보다 크다면 L의 뒤에 삽입한다.
2. 크지 않다면 해당 값의 Lowerbound인 곳에 삽입한다.



3. L 배열 채우기

LIS (with lowerbound)

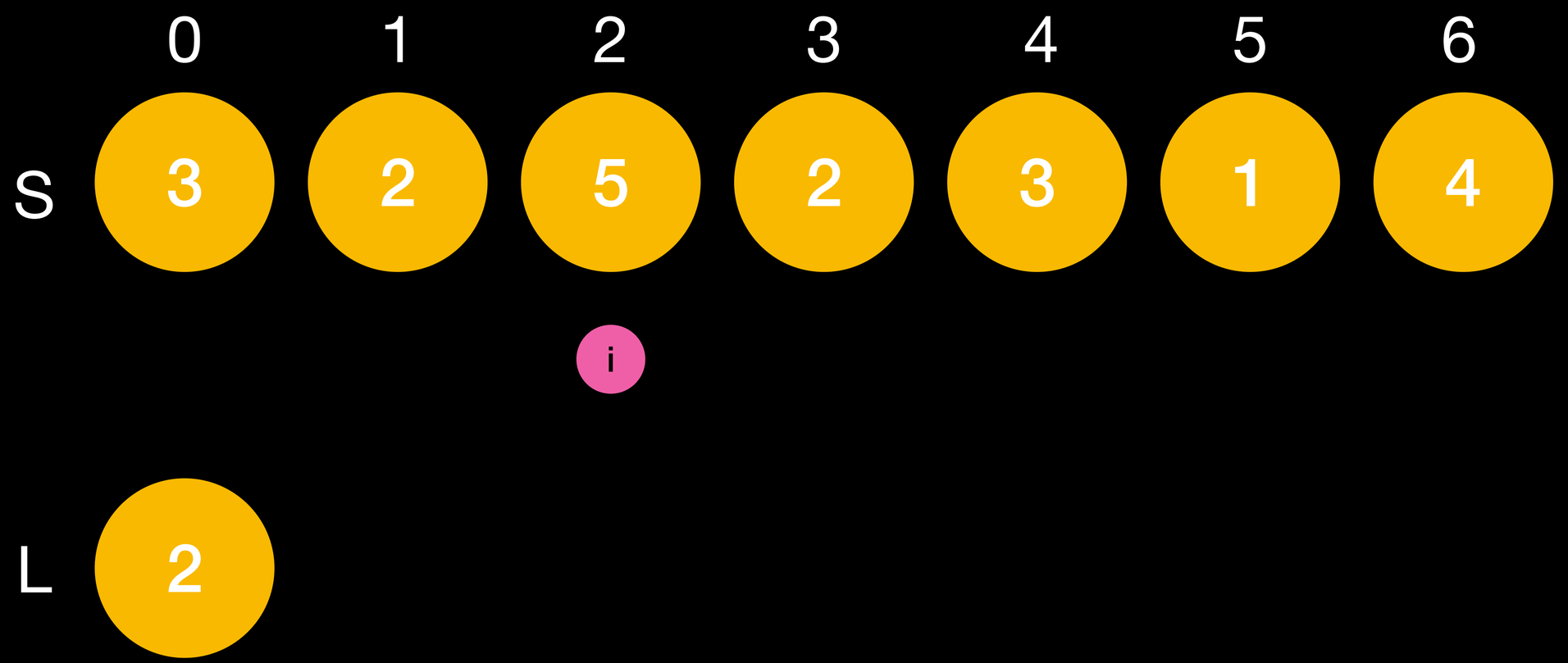
1. 기존 수열을 순회하며 L의 마지막 원소보다 크다면 L의 뒤에 삽입한다.
2. 크지 않다면 해당 값의 Lowerbound인 곳에 삽입한다.



3. L 배열 채우기

LIS (with lowerbound)

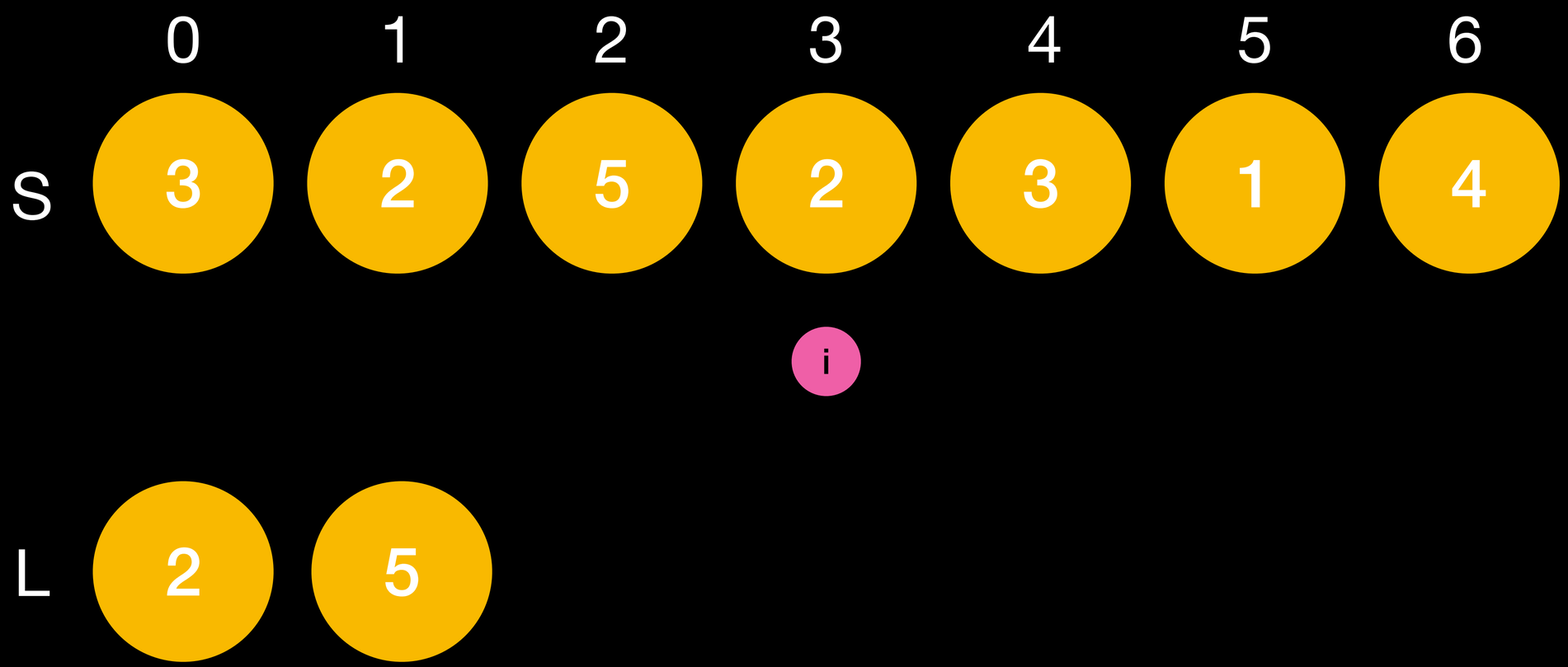
1. 기존 수열을 순회하며 L의 마지막 원소보다 크다면 L의 뒤에 삽입한다.
2. 크지 않다면 해당 값의 Lowerbound인 곳에 삽입한다.



3. L 배열 채우기

LIS (with lowerbound)

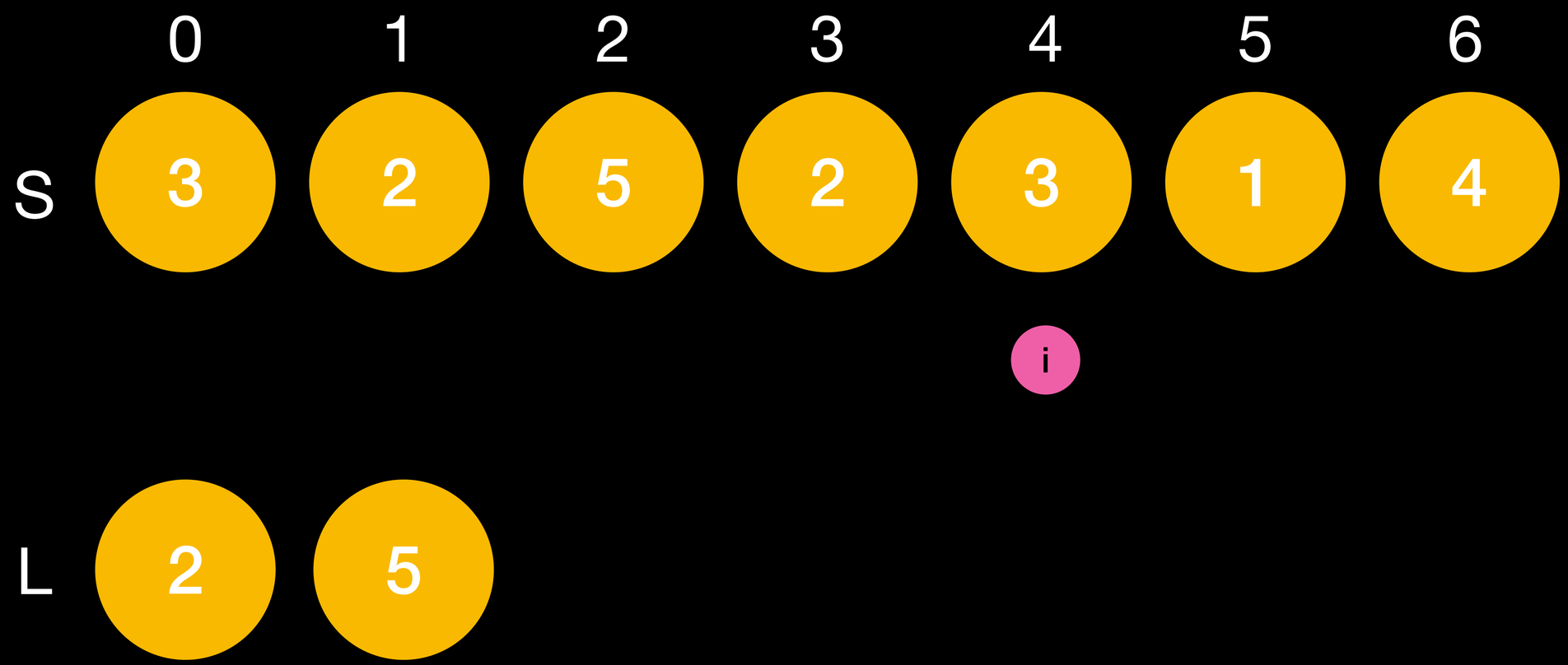
1. 기존 수열을 순회하며 L의 마지막 원소보다 크다면 L의 뒤에 삽입한다.
2. 크지 않다면 해당 값의 Lowerbound인 곳에 삽입한다.



3. L 배열 채우기

LIS (with lowerbound)

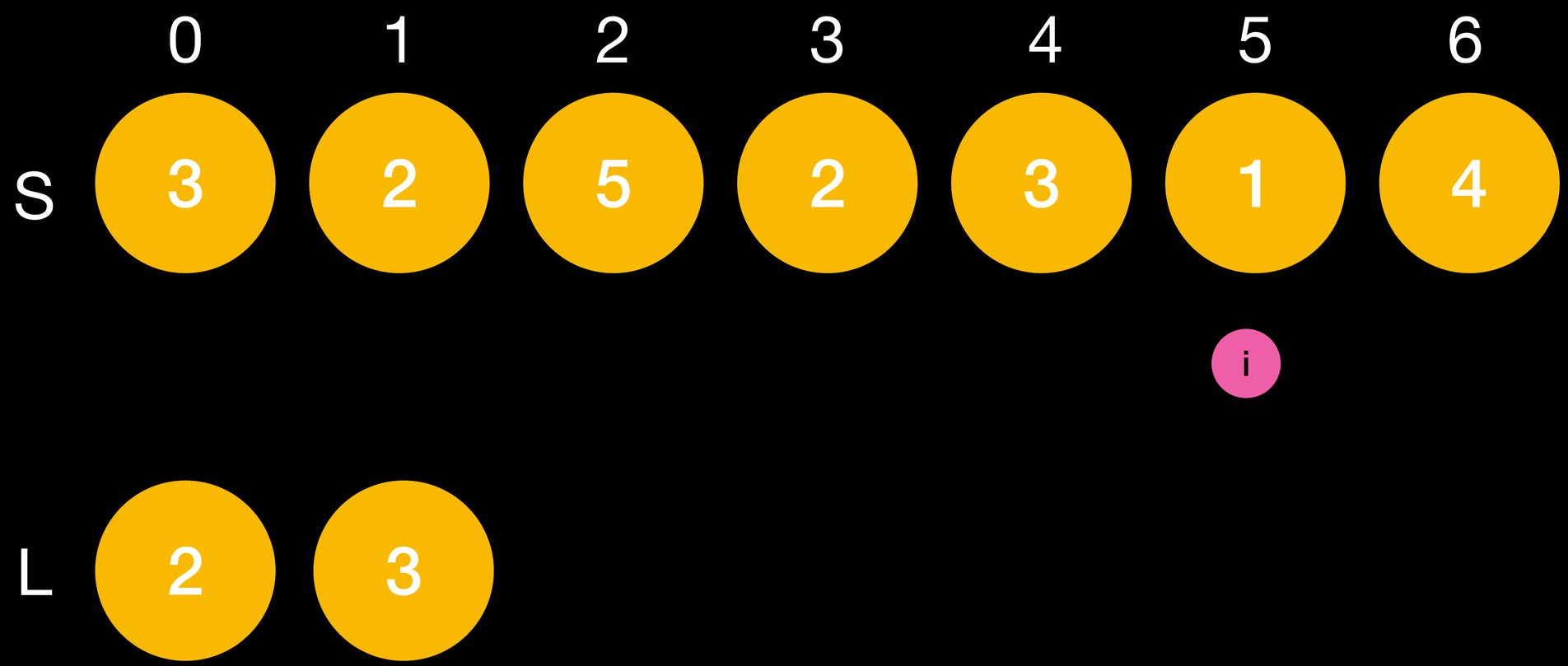
1. 기존 수열을 순회하며 L의 마지막 원소보다 크다면 L의 뒤에 삽입한다.
2. 크지 않다면 해당 값의 Lowerbound인 곳에 삽입한다.



3. L 배열 채우기

LIS (with lowerbound)

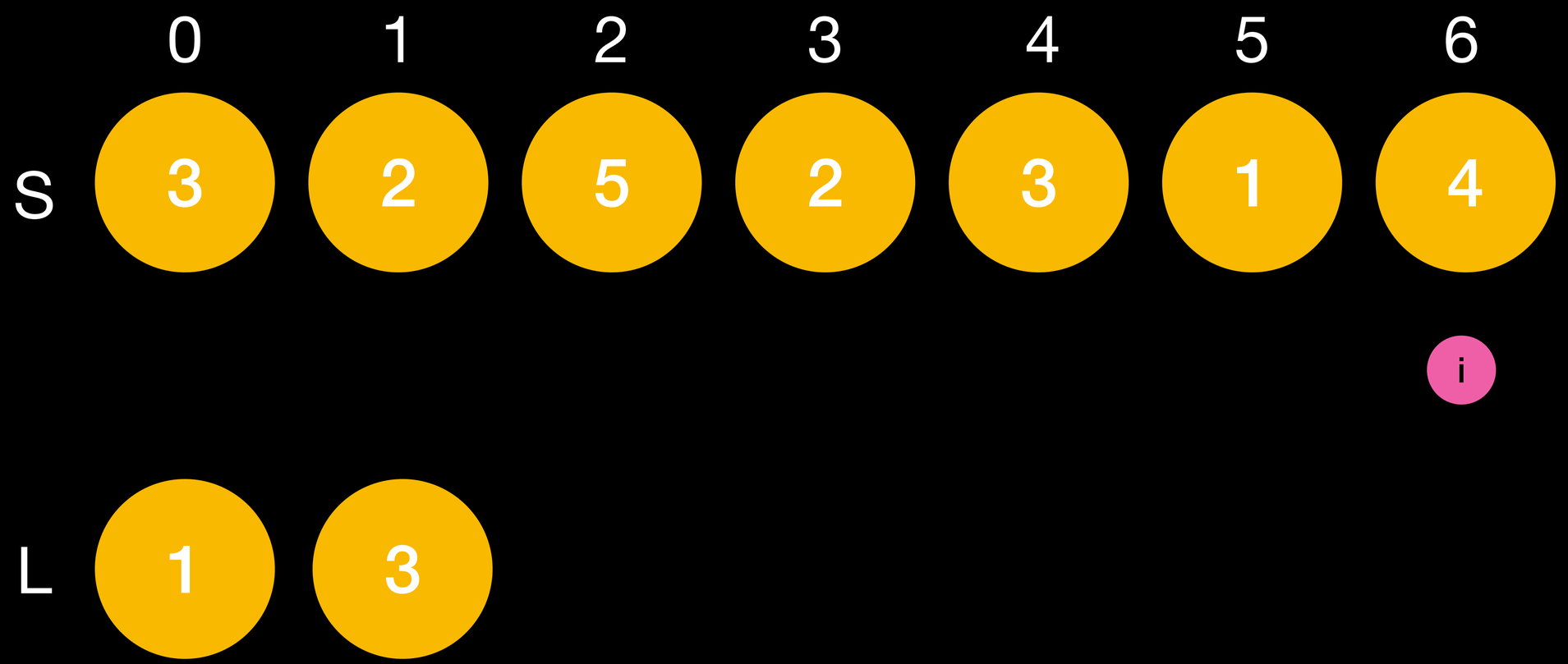
1. 기존 수열을 순회하며 L의 마지막 원소보다 크다면 L의 뒤에 삽입한다.
2. 크지 않다면 해당 값의 Lowerbound인 곳에 삽입한다.



3. L 배열 채우기

LIS (with lowerbound)

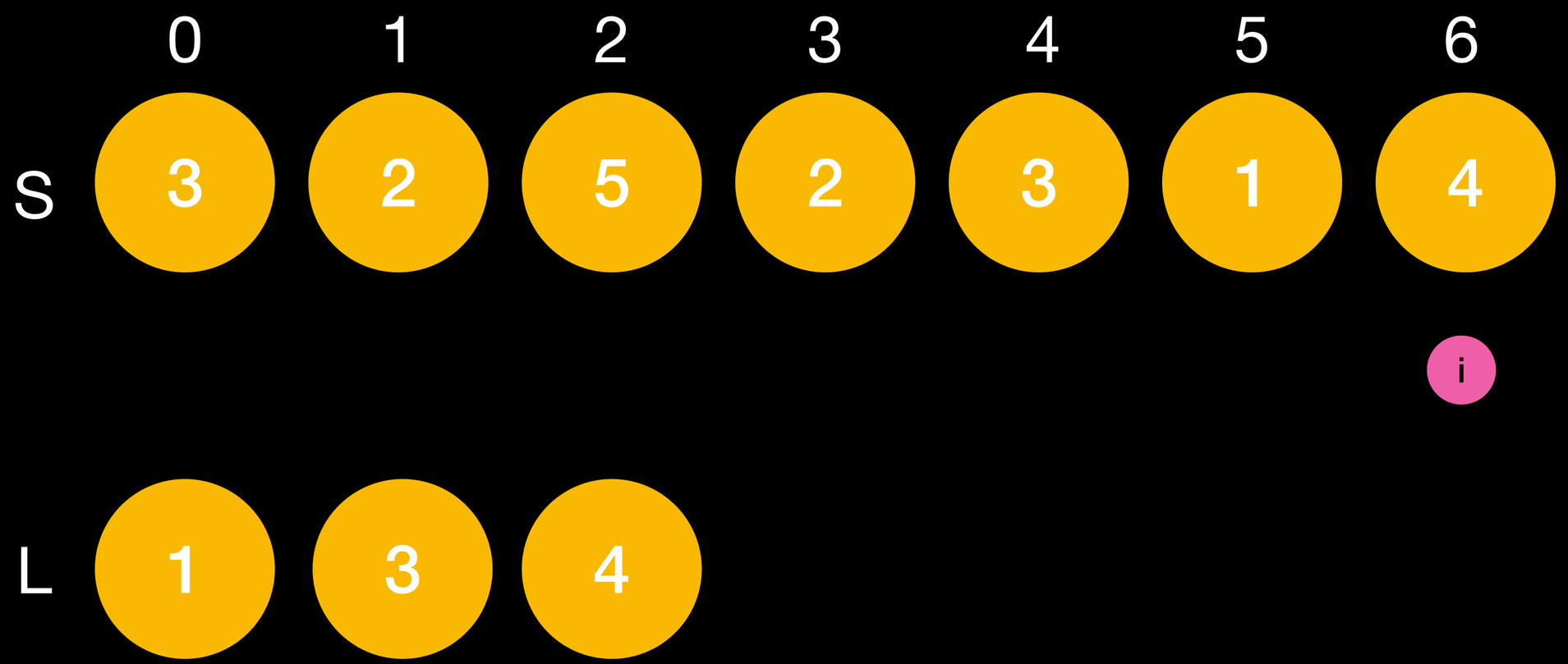
1. 기존 수열을 순회하며 L의 마지막 원소보다 크다면 L의 뒤에 삽입한다.
2. 크지 않다면 해당 값의 Lowerbound인 곳에 삽입한다.



3. L 배열 채우기

LIS (with lowerbound)

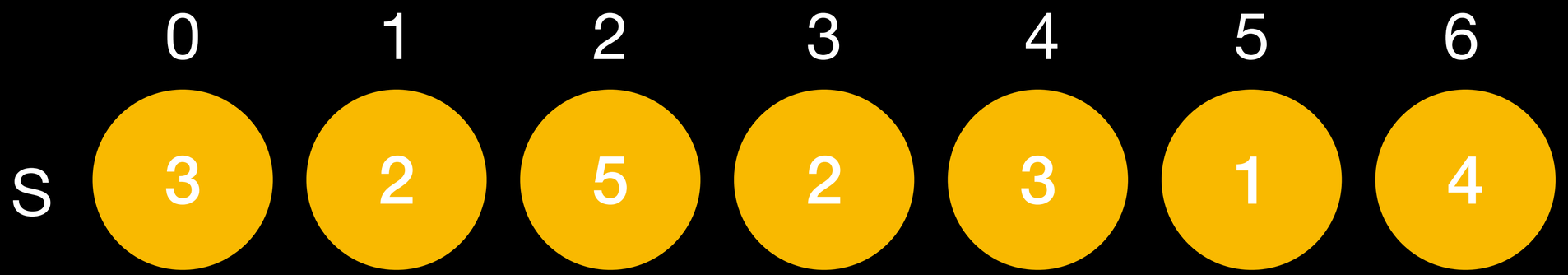
1. 기존 수열을 순회하며 L의 마지막 원소보다 크다면 L의 뒤에 삽입한다.
2. 크지 않다면 해당 값의 Lowerbound인 곳에 삽입한다.



3. L 배열 채우기

LIS (with lowerbound)

1. 기존 수열을 순회하며 L의 마지막 원소보다 크다면 L의 뒤에 삽입한다.
2. 크지 않다면 해당 값의 Lowerbound인 곳에 삽입한다.

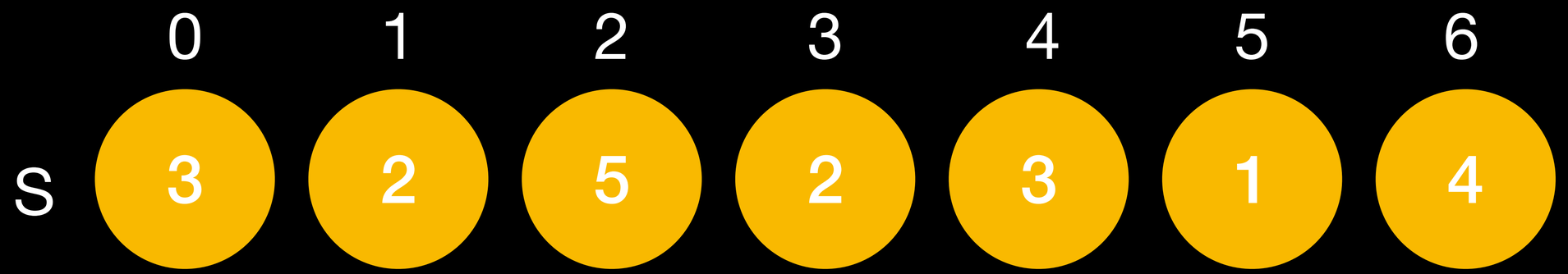


“L배열의 길이 = LIS의 길이”

3. L 배열 채우기

LIS (with lowerbound)

1. 기존 수열을 순회하며 L의 마지막 원소보다 크다면 L의 뒤에 삽입한다.
2. 크지 않다면 해당 값의 Lowerbound인 곳에 삽입한다.



“L배열의 길이 = LIS의 길이”

“L배열!= LIS”

3. L 배열 채우기

LIS (with lowerbound)

LIS의 길이와 LIS의 요소까지 모두 구하는 법

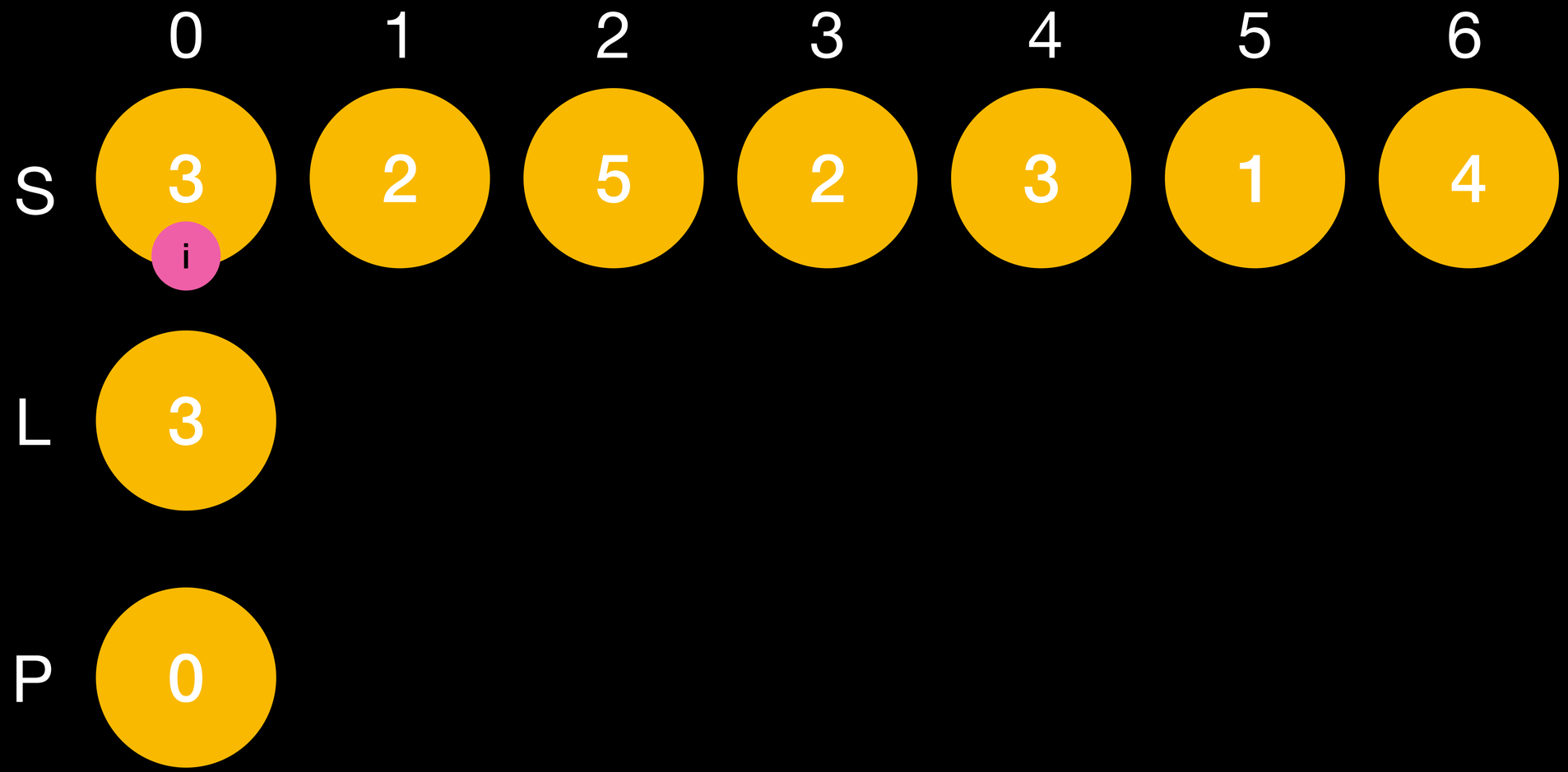
수열의 원소들이 L배열에 삽입될 때의 L배열에서 차지한 인덱스를 기억하는 P 배열을 생성한다.

1. L 배열과 완성된 P 배열을 통해 LIS의 길이와 그 요소까지 모두 구할 수 있다.
2. P 배열을 끝에서부터 탐색한다.
3. 자세한 건.. 그림으로 …

2. algorithm

1. 기존 수열을 순회하며 L의 마지막 원소보다 크다면 L의 뒤에 삽입한다.
2. 크지 않다면 해당 값의 Lowerbound인 곳에 삽입한다.

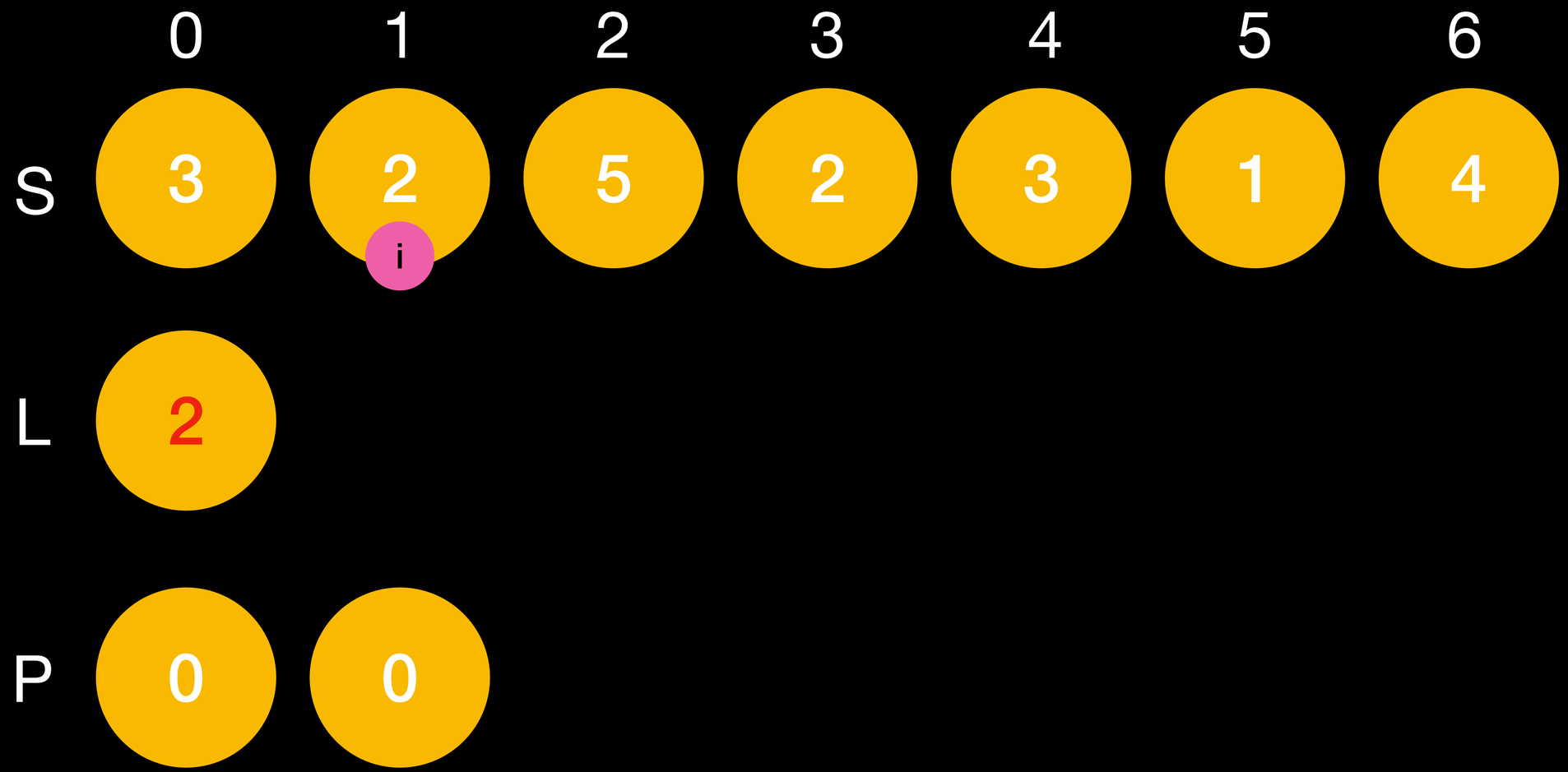
LIS (with lowerbound)



3. P 배열 채우기

LIS (with lowerbound)

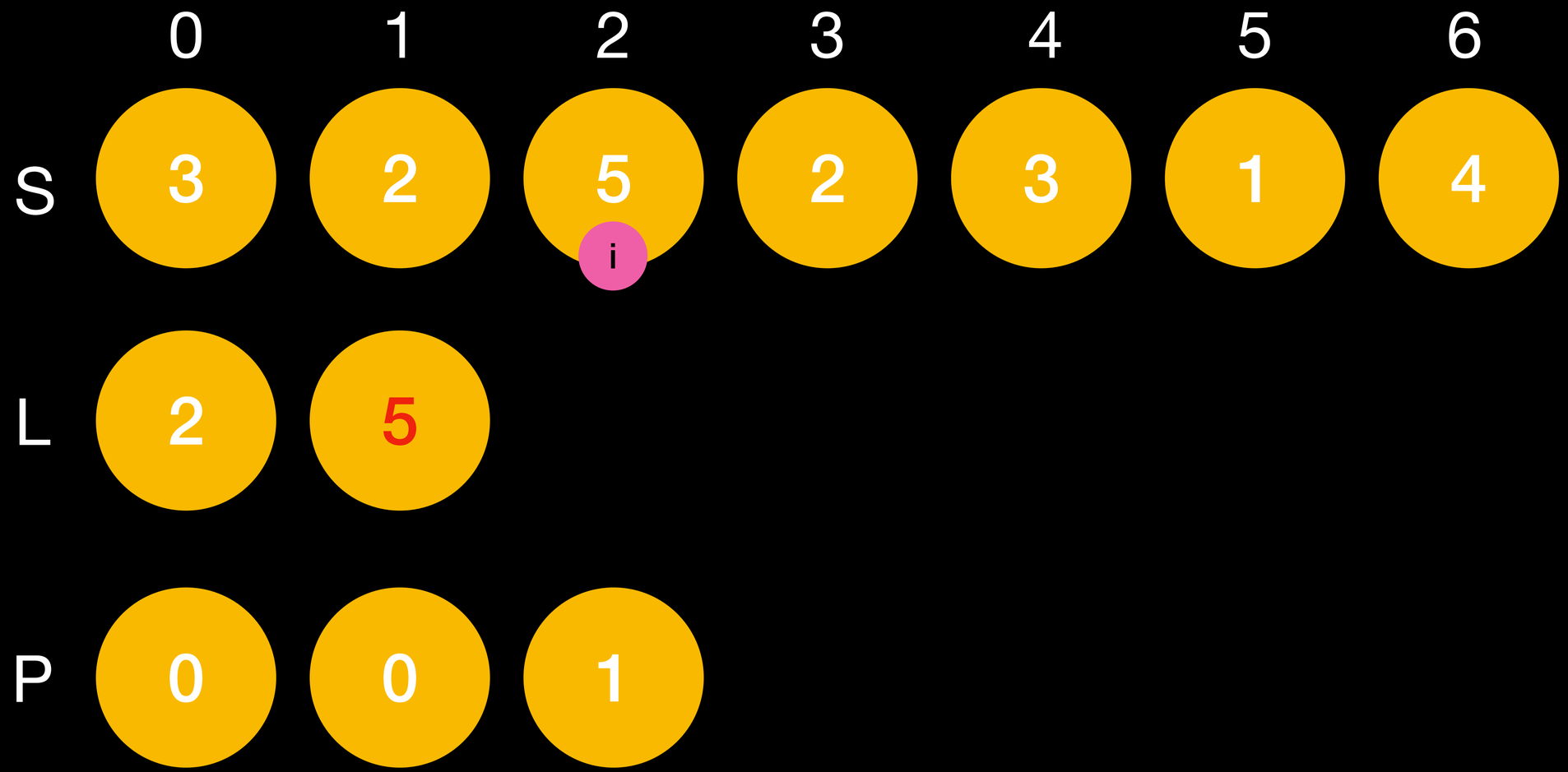
1. 기존 수열을 순회하며 L의 마지막 원소보다 크다면 L의 뒤에 삽입한다.
2. 크지 않다면 해당 값의 Lowerbound인 곳에 삽입한다.



3. P 배열 채우기

LIS (with lowerbound)

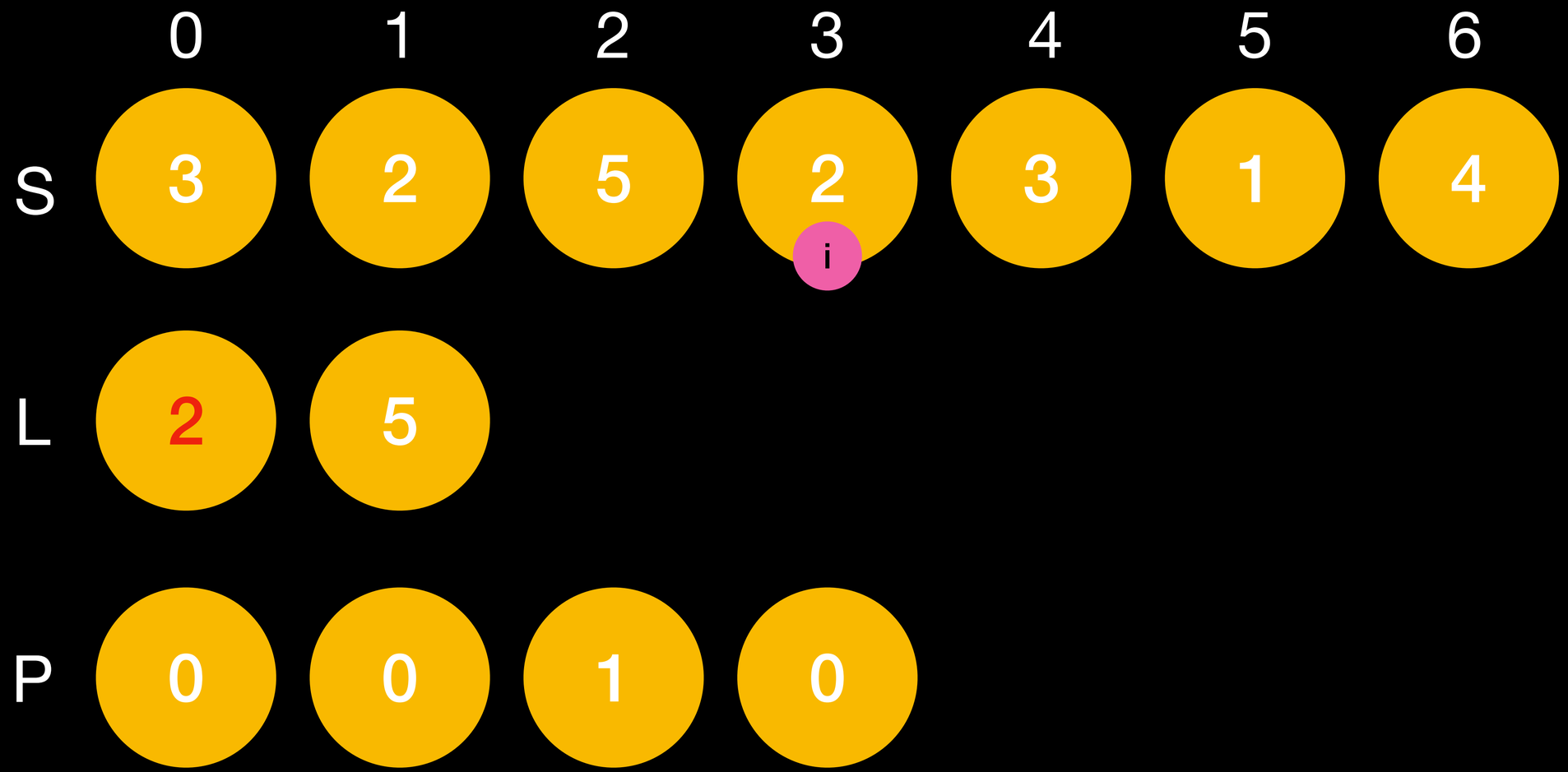
1. 기존 수열을 순회하며 L의 마지막 원소보다 크다면 L의 뒤에 삽입한다.
2. 크지 않다면 해당 값의 Lowerbound인 곳에 삽입한다.



3. P 배열 채우기

LIS (with lowerbound)

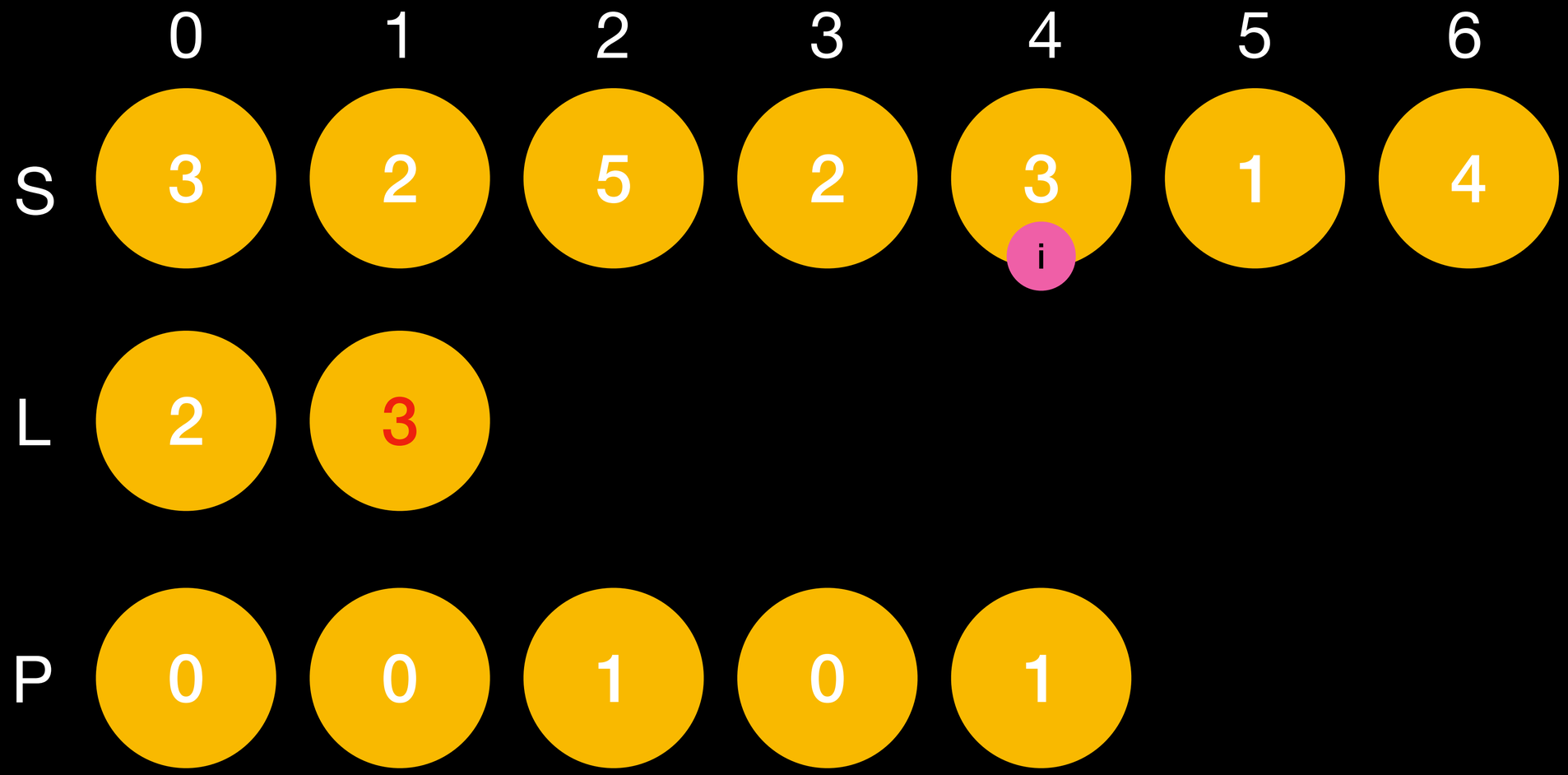
1. 기존 수열을 순회하며 L의 마지막 원소보다 크다면 L의 뒤에 삽입한다.
2. 크지 않다면 해당 값의 Lowerbound인 곳에 삽입한다.



3. P 배열 채우기

1. 기존 수열을 순회하며 L의 마지막 원소보다 크다면 L의 뒤에 삽입한다.
2. 크지 않다면 해당 값의 Lowerbound인 곳에 삽입한다.

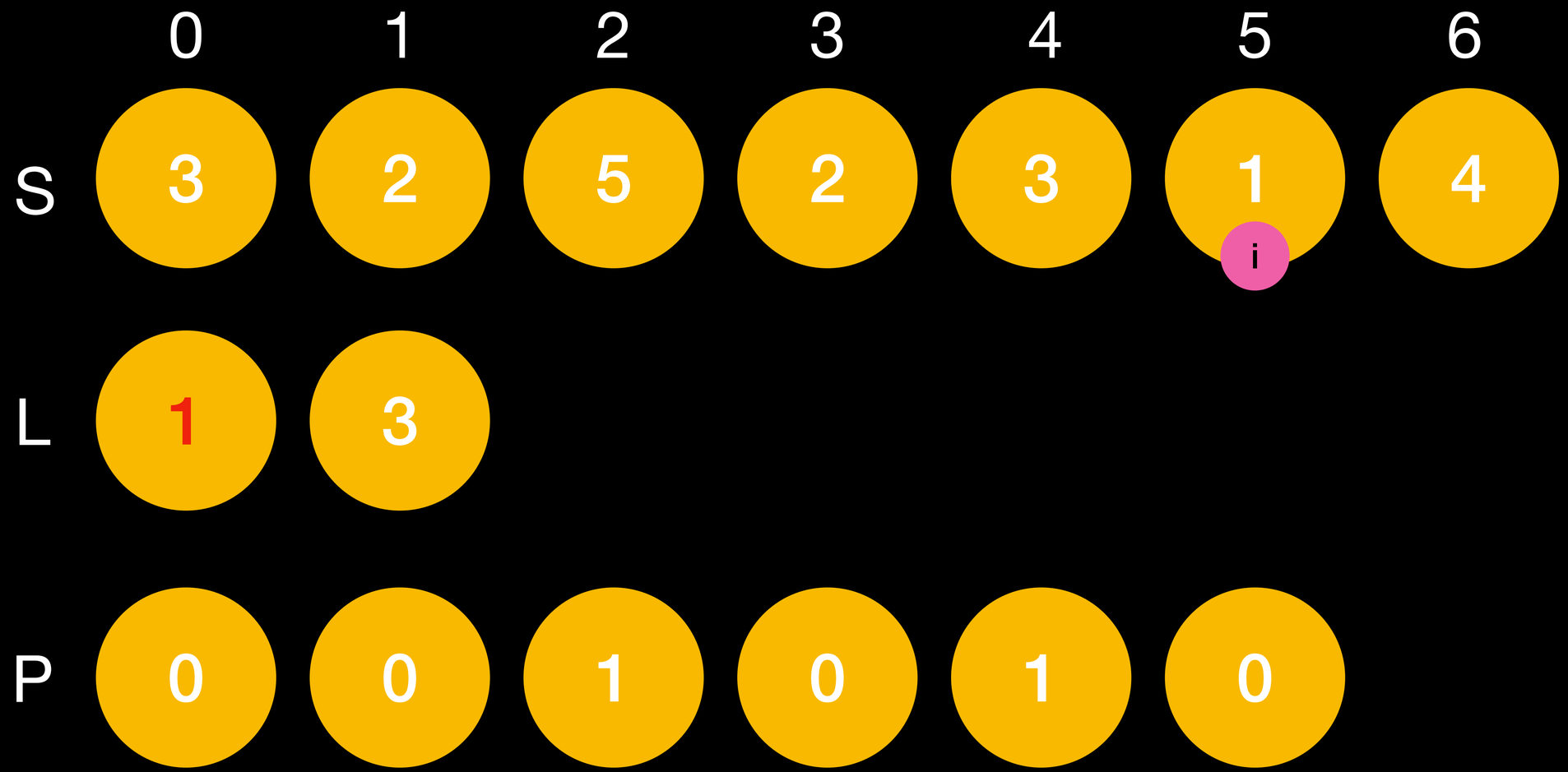
LIS (with lowerbound)



3. P 배열 채우기

LIS (with lowerbound)

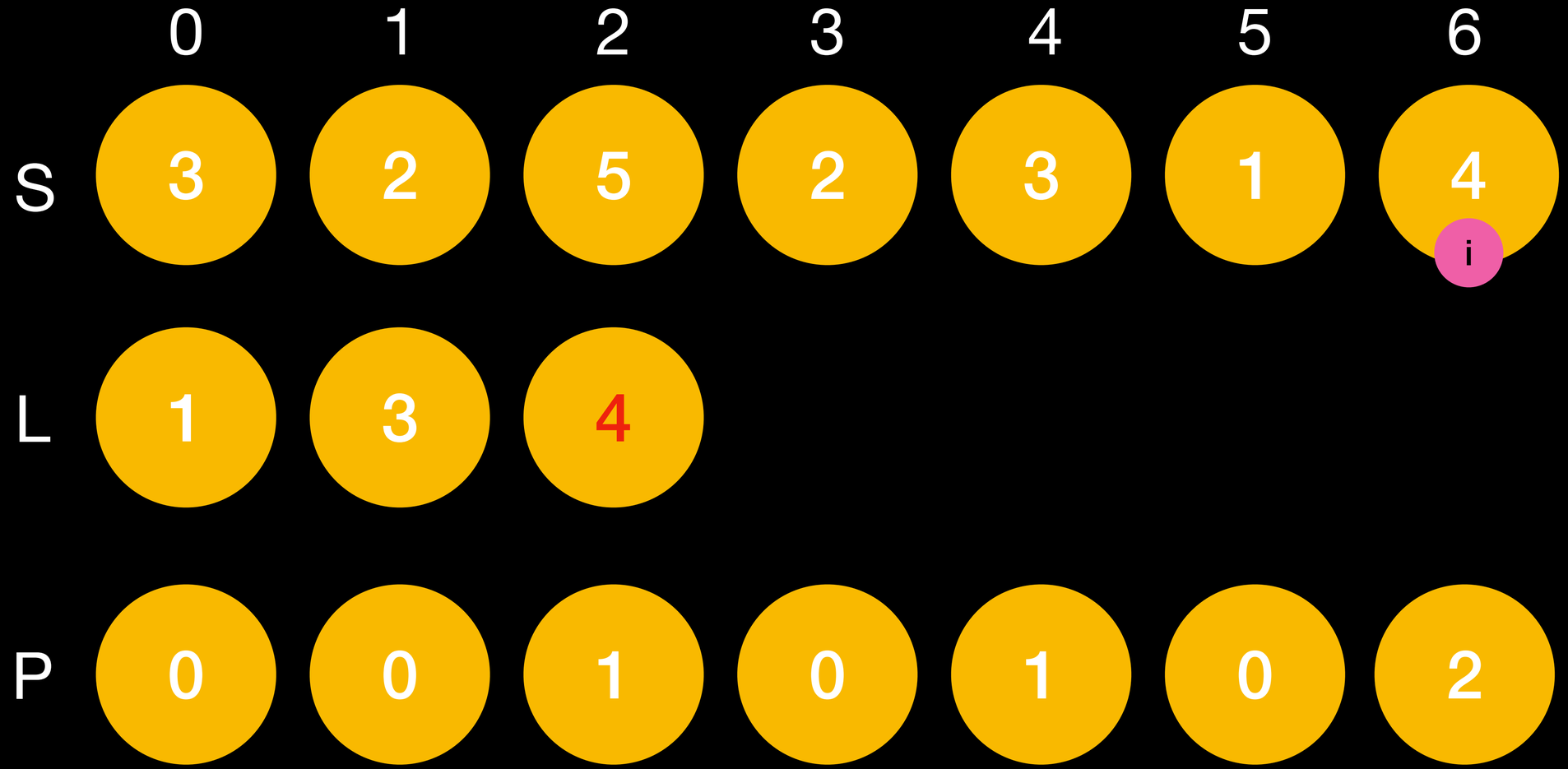
1. 기존 수열을 순회하며 L의 마지막 원소보다 크다면 L의 뒤에 삽입한다.
2. 크지 않다면 해당 값의 Lowerbound인 곳에 삽입한다.



3. P 배열 채우기

LIS (with lowerbound)

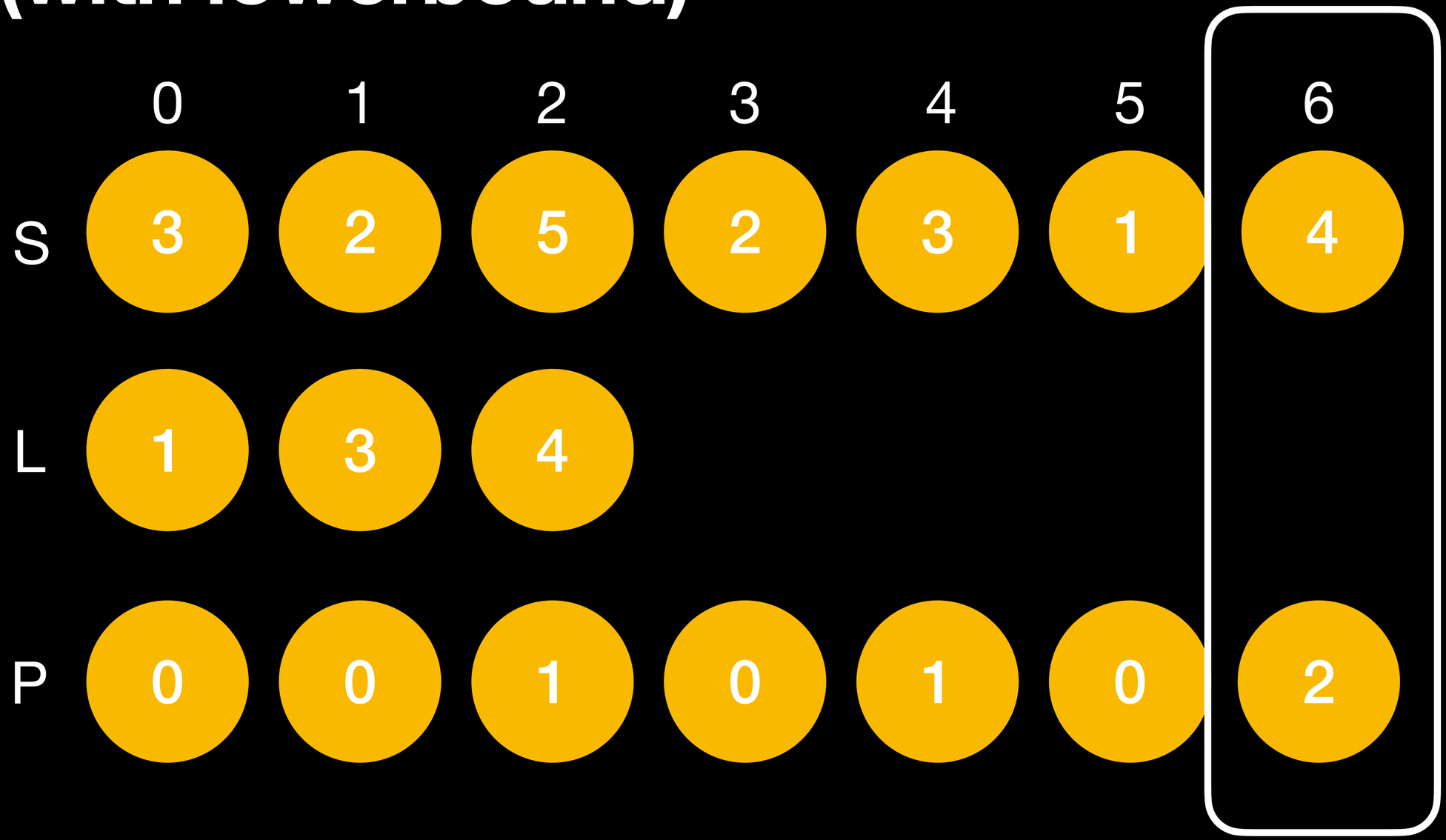
1. 기존 수열을 순회하며 L의 마지막 원소보다 크다면 L의 뒤에 삽입한다.
2. 크지 않다면 해당 값의 Lowerbound인 곳에 삽입한다.



3. P 배열 채우기

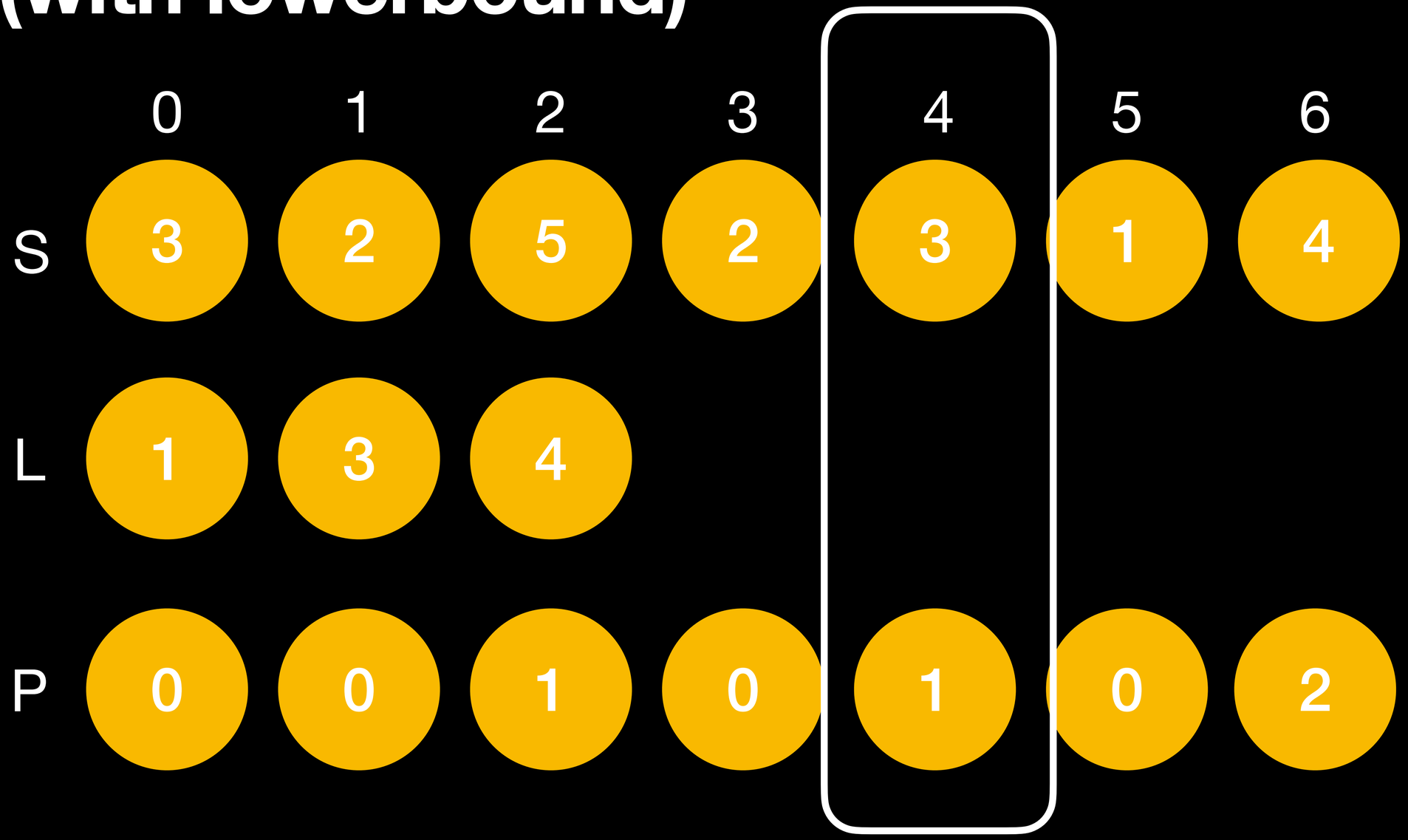
LIS = [4]

LIS (with lowerbound)



LIS = [3,4]

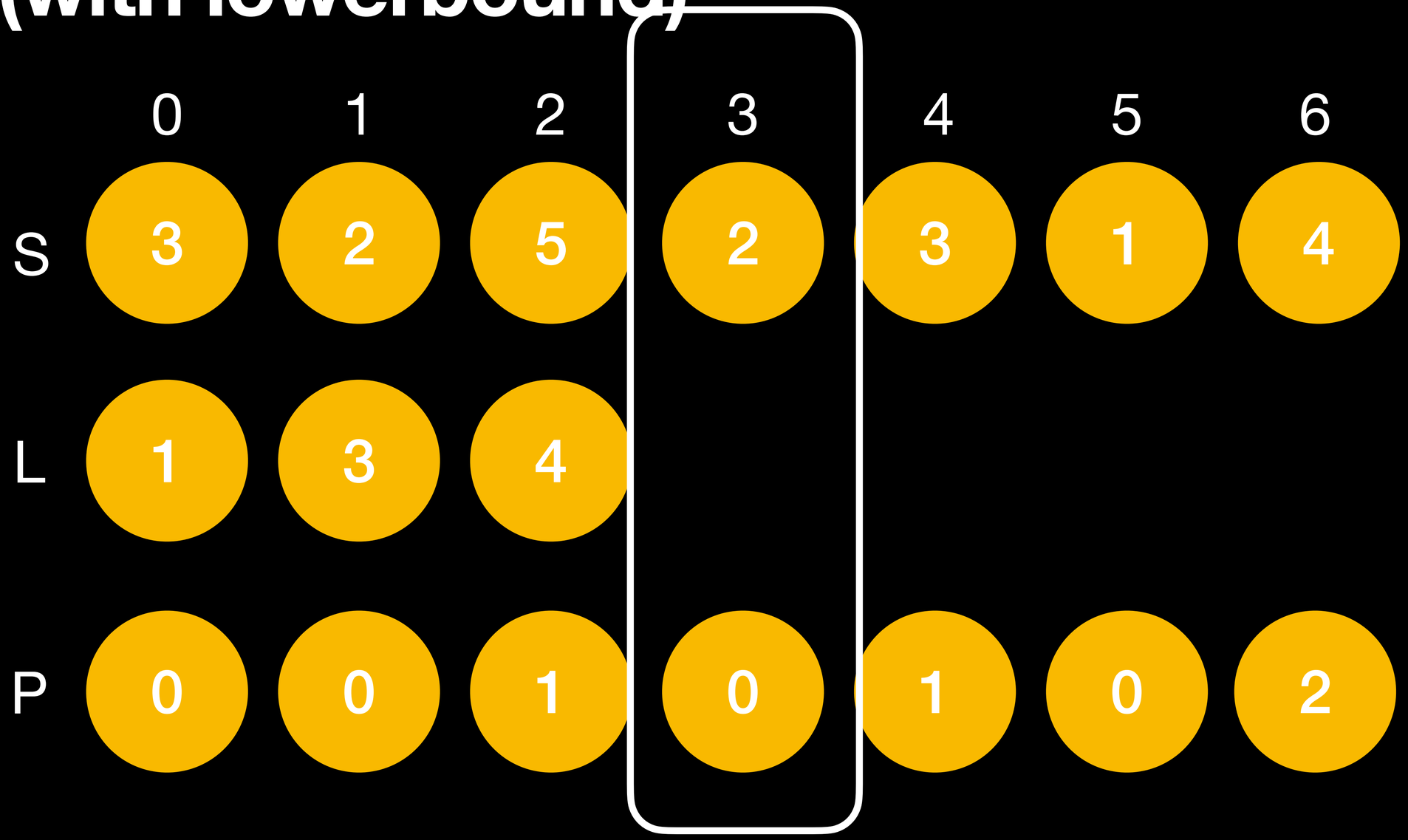
LIS (with lowerbound)



3. P 배열 채우기

LIS = [2,3,4]

LIS (with lowerbound)



3. P 배열 채우기

Bye

hungry