

과제 1-1: 비트행렬에서 1의 수 세기

- ◆ $n \times n$ 배열 A 의 각 행은 1과 0으로만 구성되며, A 의 어느 행에서나 1은 해당 행의 0보다 앞서 나온다고 가정하자
- ◆ 이에 더하여, $i = 0, 1, 2, \dots, n - 2$ 에 대해, i 행의 1의 개수는 $i + 1$ 행의 1의 개수보다 작지 않다고 가정하자
- ◆ A 가 이미 주기억장치에 존재한다고 가정하고, 알고리즘 `countOnesButSlow(A, n)`는 $O(n^2)$ 시간에 배열 A 에 포함된 1의 개수를 모두 센다
- ◆ 예: 8×8 배열 A
 - A 에 포함된 1은 총 37개

	0	1	2	3	4	5	6	7
0	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	0	0
2	1	1	1	1	1	1	0	0
3	1	1	1	1	1	0	0	0
4	1	1	1	1	1	0	0	0
5	1	1	1	1	1	0	0	0
6	1	1	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0

A

과제 1-1: 비트행렬에서 1의 수세기 (conti.)

Alg *countOnesButSlow*(A, n)
input bit matrix $A[n \times n]$
output the total number of 1's

```
1.  $c \leftarrow 0$ 
2. for  $i \leftarrow 0$  to  $n - 1$ 
     $j \leftarrow 0$ 
    while  $((j < n) \ \& \ (A[i, j] = 1))$ 
         $c \leftarrow c + 1$ 
         $j \leftarrow j + 1$ 
3. return  $c$ 
```

◆ 주의: 알고리즘 *countOnesButSlow*는 A 에 포함된 1의 개수를 세며 실행시간은 $O(n^2)$ 이다

	0	1	2	3	4	5	6	7
0	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	0	0
2	1	1	1	1	1	1	0	0
3	1	1	1	1	1	0	0	0
4	1	1	1	1	1	0	0	0
5	1	1	1	1	1	0	0	0
6	1	1	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0

A

과제 1-1: 비트행렬에서 1의 수세기 (conti.)

- ◆ 알고리즘 `countOnesButSlow`를 개선하여 같은 계산을 $O(n)$ 에 수행하는 알고리즘 `countOnes(A, n)`을 설계하라
- ◆ 두 알고리즘을 각각 **C** 함수로 구현하라 – 각 함수의 명세는 다음과 같다
 - **인자:** 비트 행렬 A , 정수 n
 - **반환값:** 정수 (1의 개수)
- ◆ 주함수는 배열을 입력 받아 느린 버전 함수, 빠른 버전 함수 순서로 호출한 결과를 인쇄한다
- ◆ **입력:** $1 + n$ 개의 라인
 - 첫 번째 라인: 정수 n ($n \times n$ 행렬에서 n 값, 단 $0 < n \leq 100$)
 - 두 번째 이후 라인: $n \times n$ 행렬 원소들 (행우선 순서)
- ◆ **출력:** 2개의 라인
 - 1의 개수 (느린 버전의 수행 결과)
 - 1의 개수 (빠른 버전의 수행 결과)
- ◆ **실행예:** p.1 배열을 그대로 사용하여 제출할 것

과제 1-2: 비트행렬에서 1의 수세기 + 실행시간 측정

- ◆ 앞 문제의 주함수를 다음과 같이 수정하여 느린 버전과 빠른 버전 각 함수의 **실행시간**을 출력하라
- ◆ 주함수는 미리 정해진(즉, 사용자 입력이 아님) n 값에 대해 **난수** 함수를 이용하여 $n \times n$ 배열을 다음과 같이 초기화한다
 - 배열의 0행부터 $n - 1$ 행까지 순서대로 각 i 행의 1의 개수 k_i 값으로 **비오름차순** 난수를 발생시켜 사용한다
 - 이때 k_{i+1} 이 k_i 의 90% 이상이어야 한다, 즉:
 - ◆ 0행에서: $0.9n \leq k_0 \leq n$
 - ◆ 1행에서: $0.9k_0 \leq k_1 \leq k_0$
 - ◆ 2행에서: $0.9k_1 \leq k_2 \leq k_1$
 - ◆ 이런 식으로, $0 < i \leq n - 1$ 행에서: $0.9k_{i-1} \leq k_i \leq k_{i-1}$
 - 이 과정에서 배열 내 1의 총수, 즉 k_i ($0 < i \leq n - 1$) 값의 총합을 $kTotal$ 변수에 누적한다

과제 1-2: 비트행렬에서 1의 수세기 + 실행시간 측정 (conti.)

- ◆ 주함수는 아래 세 가지 n 값에 대해 느린 버전, 빠른 버전 함수 쌍을 각각 호출하고 그 결과를 인쇄한다
- ◆ 입력: 없음
- ◆ 출력: 6개의 라인
 - $kTotal, ones, cputime$ ($n = 30,000$ 에 대한 빠른 버전의 수행 결과)
 - $kTotal, ones, cputime$ ($n = 10,000$ 에 대한 빠른 버전의 수행 결과)
 - $kTotal, ones, cputime$ ($n = 20,000$ 에 대한 빠른 버전의 수행 결과)
 - $kTotal, ones, cputime$ ($n = 30,000$ 에 대한 느린 버전의 수행 결과)
 - $kTotal, ones, cputime$ ($n = 10,000$ 에 대한 느린 버전의 수행 결과)
 - $kTotal, ones, cputime$ ($n = 20,000$ 에 대한 느린 버전의 수행 결과)
- ※ 참고: $kTotal$ = 실제 1의 총수, $ones$ = 프로그램이 계산한 1의 총수
- ◆ 주의
 - 배열 초기화 작업은 각 함수 호출 전이므로 실행시간에서 제외
 - 실행시간 0초는 허용하지 않음
 - PC 성능 문제로 불가피한 경우에 한해, 세 개의 n 값을 “일제히” 10배 또는 1/10배로 올리거나 낮추는 것을 허용함