

수행시간 체험 - 다항식 계산

고경환

조건

- RAM 모델에서 실행된다고 가정한다.
- for문의 제어를 위해 필요한 기본연산의 횟수는 세지 않는다.

(이유: n의 크기에 상관없이 똑같이 반복되기 때문)

1. 두 함수의 수행시간을 분석해 Big-O로 표기할 것

<pre>def evaluate_n2(A, x): 대입 ① n = len(A) 대입 ② result = 0 for i in range(n): 대입 ③ power-term = 1 곱, 대입 2 * 1/2 (n^2 - n) = (n^2 - n) for j in range(i): power-term *= x — (*) 곱, 합, 대입 ③ result += A[i] * power-term return result</pre> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> $\Rightarrow T_2(n) = (n^2 - n) + n + 3n + 2$ $= n^2 + 3n + 2$ $= O(n^2)$ </div>	<pre>def evaluate_n(A, x): 대입 ① n = len(A) 대입 ② result = 0 대입 ③ power-term = x for i in range(n-1): 곱, 대입 2(n-1) A[i+1] = A[i+1] * power-term 곱, 대입 2(n-1) power-term *= x for j in range(n): 합, 대입 ② result += A[j] return result</pre> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> $\Rightarrow T_1(n) = 2(n-1) + 2(n-1) + 2n + 3$ $= 6n - 1$ $= O(n)$ </div>
--	---

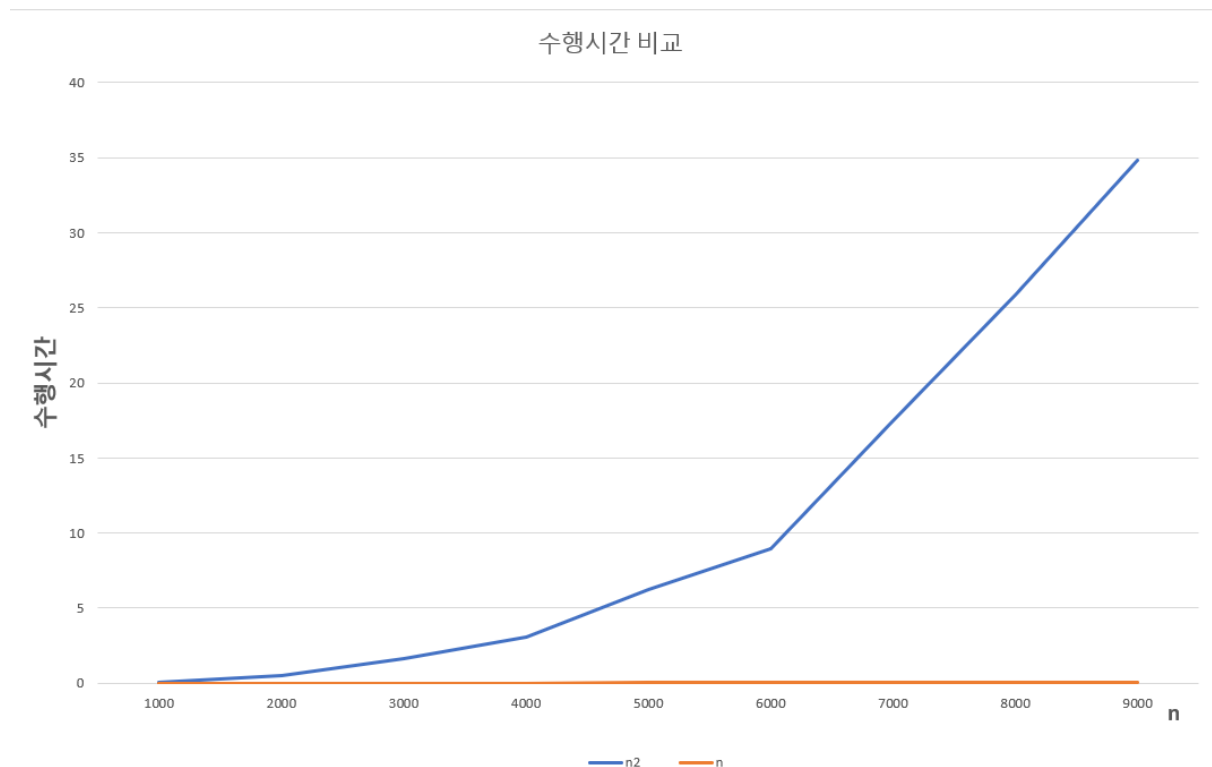
위 풀이에서 evaluate_n2함수의 두번째 for문에 * 부분은

$$\begin{array}{c|c}
 i & j \\
 \hline
 0 & 0 \\
 1 & 1 \\
 \vdots & \vdots \\
 n-1 & n-1
 \end{array}
 \rightarrow \overbrace{0+1+\dots+n-1}^{n\text{개}}$$

$$\begin{aligned}
 & \underbrace{1+2+\dots+n}_{\left(= \frac{n(n+1)}{2}\right)} - n \\
 &= \frac{n^2+n}{2} - \frac{2n}{2} \\
 &= \frac{n^2-n}{2}
 \end{aligned}$$

위와 같이 해당 명령어의 연산횟수는 곱셈, 대입 연산이 $(n^2 - n)/2$ 만큼 반복되므로 총 $n^2 - n$ 의 시간이 걸린다고 할 수 있다.

2. 다양한 n값에 따른 두 함수의 실제 실행 시간을 그래프 등의 시각적인 방법으로 나타내어 설명할 것



제한 수행시간(60초) 이내에서 출력결과물이 나오는 1000~9000까지 n을 다양하게 입력하며 이에 따른 evaluate_n2(파랑), evaluate_n(주황)함수의 수행시간을 알아보았습니다.

n이 1000~2000정도 일 때는 별차이 나지 않다가 점차 n이 증가함에 따라 evaluate_n2함수의 수행시간이 evaluate_n에 비해 급격하게 증가하는 양상을 볼 수 있습니다. 이를 통해 n이 훨씬 증가할 수록 evaluate_n2함수는 evaluate_n함수에 비해 훨씬 많은 시간이 소요되므로 evaluate_n함수가 evaluate_n2에 비해 훨씬 효율적이고 좋은 알고리즘이라고 할 수 있습니다.

3. 느낀점 작성

앞으로 알고리즘을 작성할 때 반복문을 중첩을 줄여 최대한 중첩없이 풀 수 있도록 고민하는 것이 좋겠다고 느꼈습니다.