4. 기초수식

1번

- 문제 1: $T(n) = T(n-1) + 1$

- `T(0) = 1`로 설정하고 진행

$T(n) = T(n-1) + 1$

$= (T(n-2) + 1) + 1$

$= (T(n-3) + 1) + 1 + 1$

$...$

$= T(n-k) + k$

$(k = n$인경우$)$

$= T(0) + n$

$(T(0) = 1$이므로$)$

$\therefore T(n) = 1 + n$

$\therefore T(n) = O(n)$

2 번

- 문제 2: $T(n) = T(n-1) + n$

- `T(0) = 1`로 설정하고 진행

$T(n) = T(n-1) + n$

$T(n-1) = T(n-2) + n - 1$

$T(n-2) = T(n-3) + n - 2$

....

$T(1) = T(0) + 1$

$\Rightarrow$

$T(n) + T(n-1)....T(1) = T(n-1) + T(n-2)... + T(1) + T(0) + n + (n-1)... + 2 + 1$

$\Rightarrow$

$T(n) = n + (n-1) + (n-2)...2 + 1 + T(0)$

$\Rightarrow$

$T(n) = n(n+1)/2 + 1$

$\therefore T(n) = O(n^2)$

4번

- 문제 4: $T(n) = T\left(\frac{n}{2}\right) + 1$

- `T(1) = 1`로 설정하고 진행

$$
\begin{aligned}
T(n) &= T\left(\tfrac{n}{2}\right) + 1 \\
&= T\left(\tfrac{n}{2^2}\right) + 1 = T\left(\tfrac{n}{2^2}\right) + 2 \\
&= T\left(\tfrac{n}{2^3}\right) + 1 = T\left(\tfrac{n}{2^3}\right) + 3 \\
&\ \ \vdots \\
&= T\left(\tfrac{n}{2^k}\right) + k
\end{aligned}
$$

$2^k = n$인 경우 $\rightarrow k = \log_2 n$

$$
\begin{aligned}
&= T(1) + \log n \\
&= 1 + \log n \\
&\therefore \boxed{O(\log n)}
\end{aligned}
$$

6번

- 문제 6: $T(n) = 2T\left(\frac{n}{2}\right) + n$

$T(n) = 2T(\frac{n}{2}) + n$

$= 2(2T(\frac{n}{2^2}) + \frac{n}{2}) + n$

$= 2(2(2T(\frac{n}{2^3}) + \frac{n}{2^2}) + \frac{n}{2}) + n$

...

$= 2^k T(\frac{n}{2^k}) + n(k-1)$

$2^k = n, k = log(n), T(1) = 1$이므로

$$\therefore n + n(logn - 1)$$
$$= nlog(n)$$

$$\therefore T(n) = O(nlog(n))$$

8번

- 문제 8: $T(n) = T(n-1) + \frac{1}{n}$

$$T(n) = T(n-1) + \frac{1}{n}$$
$$= T(n-2) + \frac{1}{n-1} + \frac{1}{n}$$
$$= T(n-3) + \frac{1}{n-2} + \frac{1}{n-1} + \frac{1}{n}$$
$$= T(n-k) + \frac{1}{n-k+1} + ... + \frac{1}{n-2} + \frac{1}{n-1} + \frac{1}{n}$$

$T(1) = 1, k = n - 1$이므로
$$\therefore T(n) = T(1) + \frac{1}{n-(n-2)} + \frac{1}{n-(n-3)} + ... + \frac{1}{n}$$
$$= 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + ... + \frac{1}{n}$$
$$= log(n)$$

$$\therefore T(n) = O(log(n))$$

- 참고

  https://gateoverflow.in/41209/Find-oder-of-this-algorithm-t-n-t-n-1-1-n-if-n-1

Big-O 표기법에 대한 이해

1. **빅오 표기법의 정의**

어떤 알고리즘을 수행하는 데 걸리는 시간을 설명하는 계산 복잡도를 의미하며, 계산 복잡도를 표기하는 대표적인 방법이 빅오 표기법인 것이다.

2. **빅오 표기법의 특징**
    - 빅오로 시간 복잡도를 표현할때는 **최고차항만 표기힌다.**
    - **O(1)**
      : 입력값에 상관없이 일정한 실행시간을 최고!의 알고리즘이라 할 수 있다. 하지만 상수 시간에 실행된다 해도 상수값이 상상 이상으로 클 경우 사실상 일정한 시간의 의미가 없다. 최고의 알고리즘이 될 수 있지만 그만큼 신중해야 한다.
    - **O(log n)**
      : 로그는 매우 큰 입력값에서도 크게 영향을 받지 않는 편이다. 매우 견고한 알고리즘으로 이진 탐색의 경우가 이에 해당한다.
    - **O(n)**
      : 알고리즘을 수행하는데 걸리는 시간은 입력값에 비례한다. 이러한 알고리즘을 선형 시간 알고리즘이라 부른다. 정렬되지 않은리스트에서 최대 또는 최솟값을 찾는 경우가 해당되며 **모든 입력값을 적어도 한 번 이상은 살펴봐야 한다.**
    - **O (n log n)**
      : 병합 정렬등의 **대부분 효율이 좋은 알고리즘이 이에 해당 한다. 아무리 좋은 알고리즘이라 할지라도 n log n 보다 빠를 수 없다.**
      입력값이 **최선**일 경우, 비교를 건너 뛰어 O(n)이 될 수 있다.
    - **O(n^2)**
      : 버블 정렬 같은 비효율적인 정렬 알고리즘이 이에 해당 한다.
    - **O(2^n)**
      : 피보나치의 수를 재귀로 계산하는 알고리즘이 이에 해당 한다. n^2와 혼동되는 경우가 있는데 2^n이 훨씬 더 크다.
    - **O(n!)**
      : 가장 느린 알고리즘으로 입력값이 조금만 커져도 계산이 어렵다.
3. `f(n) = 3n^2 + 2n + 5`**를 빅오 표기법으로 표시하기**

$\therefore O(n^2)$

참고 자료

https://www.radford.edu/~nokie/classes/360/recurrence.eqns.revised.html