Recursive 호출 알고리즘의 Time Complexity

2013210111 남세현

2가지 경우의 Recursive Algorithm의 시간복잡도를 직접 구해보겠습니다.

Factorial

F(n) = F(n-1) * n 이라고 정의를 합니다.

```
int F(int n)
{
        if( n == 0)
                return 1;
        return F(n-1) * n;
}
```

기본 연산이 상수시간 내에 계산이 된다고 전제를 하고, F(n)의 시간복잡도를 T(n)이라고 하면

T(n) = T(n-1) + 2, T(1) = 1 이라 할 수 있습니다.

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

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

항 (1), (2)에 의해

T(n) = T(n-2) + 2 * 2,

위 같은 연산을 계속하면

T(n) = T(n-3) + 3 * 2, 그리하여

T(n) = T(n-t) + t * 2 ... (3)꼴임이 됨을 알 수 있습니다.

T(1)의 값을 알고 있으므로, T(1) = T(n-t)을 만족하는 t는

t = n - 1 ... (4) 입니다.

항 (3)에 (4)를 대입하여 풀면

T(n) = 1 + (n-1) * 2 = 2n – 1

그러므로 Factorial의 Timecomplexity, Big O 표현으로는 O(n) 입니다.

Binary Search

정렬되어 있는 배열 안에 찾으려는 값의 index를 반환하는 함수입니다. (위키에서 가져옴)

```
BinarySearch(A[0..N-1], value, low, high) {
  if (high < low)
    return -1 // not found
  mid = (low + high) / 2
  if (A[mid] > value)
    return BinarySearch(A, value, low, mid-1)
  else if (A[mid] < value)
    return BinarySearch(A, value, mid+1, high)
  else
    return mid // found
}
```

배열 A의 Length는 N으로 고정되어 있고 찾으려는 value도 정해져 있다고 가정합니다. 초기의 low, high의 값이 0, N-1로 설정되어 있다고 합시다. Low와 high 를 그대로 표현하면 시간복잡도의 증명이 어려움으로, 'high – low'의 값을 BS 함수의 인자로 하겠습니다.

BS(L) = BS(L/2) + 1

BS(1) = 1

(이어서)

BS(L) = BS(L/2) + 1

BS(L/2) = BS(L/(2^2)) + 1

BS(L/2^2) = BS(L/(2^3)) + 1

위 세 식을 연립하여 풀면

BS(L) = BS(L/2^2) + 2

즉,

BS(L) = BS(L/2^t) + t ... (1) 꼴이 됨을 알 수 있습니다.

우리는 Big O, 즉 상한 점근을 알고 싶은 것임으로 L/2^t = 1이 되는 t를 구하면

T = log2(L). 이 값을 (1) 항에 대입하면

BS(L) = BS(1) + log2(L) = log2(L) + 1

즉, Big O로 O(logn)이 됨을 알 수 있습니다.

결론

상수적으로 값이 나오는 부분(T(1) = 1)과, 일반적인 부분(T(n) = T(n-1) + 1)을 조합하여 점화식 꼴로 만들어 문제를 해결하면 시간 복잡도를 구할 수 있습니다.