1) 알고리즘 기초

매 강의 강의자료 시작에 PDF파일을 올려두었어요!

▼ PDF 파일

[수업 목표]

- 알고리즘이 무엇인지, 그리고 왜 필요한지에 대해 배웁니다.
- 시간 복잡도와 공간 복잡도에 대해 배우고, 이를 계산하는 방법을 학습합니다.

[목차]

모든 토글을 열고 닫는 단축키
Windows : `Ctrl` + `alt` + `t`
Mac : `⌘` + `⌥` + `t`

---

01. 알고리즘 (Algorithm)

✔️ 문제를 해결하기 위한 단계적인 방법

▼ **1) 알고리즘 개념**

- 알고리즘은 문제를 해결하기 위한 명확한 절차나 방법입니다.
- 알고리즘은 **입력을 받아 원하는 출력을 생성하기 위한 절차**입니다.
- 알고리즘은 입력, 출력, 명확한 단계, 실행 가능성의 특성을 갖습니다.
- 알고리즘은 주어진 입력에 대해 **정확하고 일관된 결과**를 제공해야 합니다.
- 알고리즘은 컴퓨터 프로그래밍뿐만 아니라 다양한 분야에서 사용됩니다.

▼ **2) 알고리즘의 중요성**

- 효율적인 알고리즘은 컴퓨터 프로그래밍에서 매우 중요합니다.
- 같은 문제를 해결하는 두 알고리즘이 있다면, 효율적인 알고리즘은 덜 효율적인 알고리즘보다 더 적은 컴퓨팅 자원(시간, 메모리 등)을 사용합니다.
- 따라서, 가능한 한 효율적인 알고리즘을 사용하는 것이 중요합니다.

02. Big O 표기법

✔️ 알고리즘 전투력 측정기!

▼ **1) Big O 표기법**

- Big O 표기법은 알고리즘의 효율성을 나타내는 표기법입니다.
- 특히, Big O 표기법은 입력의 크기에 따라 알고리즘이 얼마나 많은 시간이나 공간을 필요로 하는지를 설명합니다.
- Big O 표기법은 알고리즘의 최악의 경우 성능을 나타내므로 알고리즘의 효율성을 과장하지 않습니다.

▼ **2) Big O 표기법의 예**

- O(1): 상수 시간. 입력의 크기에 상관없이 항상 일정한 시간이 걸립니다.
- O(n): 선형 시간. 입력의 크기에 비례하여 시간이 걸립니다.
- O(n^2): 이차 시간. 입력의 크기의 제곱에 비례하여 시간이 걸립니다.

- O(log n): 로그 시간. 입력의 크기의 로그에 비례하여 시간이 걸립니다.

▼ **3) 빅오 표기법 계산**

1. **상수 버리기**
   알고리즘의 시간 복잡도에서 가장 큰 영향을 주는 항목을 중심으로 살펴봅니다. 상수 항목이나 낮은 차수의 항목은 빅오 표기법에서 중요하지 않으므로 버립니다. 예를 들어, O(2n), O(3n^2)와 같은 경우 O(n), O(n^2)로 간소화할 수 있습니다.
2. **최고 차수 항목만 남기기**
   빅오 표기법에서는 가장 빠르게 증가하는 항목에 초점을 맞춥니다. 따라서 가장 큰 차수의 항목만을 남기고 나머지는 버립니다. 예를 들어, O(n^2 + n)의 경우 O(n^2)로 간소화할 수 있습니다.
3. **다항식의 경우 최고 차수 항목만 고려**
   다항식의 경우 가장 빠르게 증가하는 항목에 초점을 맞춥니다. 상수항이나 낮은 차수의 항목은 무시합니다. 예를 들어, O(3n^3 + 5n^2 + 2n + 7)의 경우 O(n^3)로 간소화할 수 있습니다.
4. **연산자 상수 무시**
   빅오 표기법에서는 연산자나 상수 값을 무시합니다. 예를 들어, O(3n^2 + 4n + 2)의 경우 O(n^2)로 간소화할 수 있습니다.

03. 시간 복잡도 (Time Complexity)

✔️ **How Long?**

▼ **1) 시간 복잡도란?**

- 시간 복잡도란 알고리즘이 문제를 해결하는데 걸리는 시간을 나타내는 척도입니다.
- 코드의 실행 시간을 실제 시간(초)으로 측정하는 것이 아니라, 입력 크기에 대한 연산 횟수로 측정합니다.
- Big-O 표기법을 사용하여 표시함

▼ **2) 활용 예제**

▼ **예제 1**

```
int Sum(int n)
{
```

```
    int sum = 0;
    for (int i = 0; i <= n; i++)
    {
        sum += i;
    }
    return sum;
}
```

💡 **해답**

for 루프가 0부터 n까지 순회하며 합계를 계산합니다. 따라서 n회의 연산이 필요하며, 시간 복잡도는 O(n)입니다.

▼ 예제 2

```
void PrintPairs(int n)
{
    for (int i = 0; i <= n; i++)
    {
        for (int j = 0; j <= n; j++)
        {
            Console.WriteLine(i + ", " + j);
        }
    }
}
```

💡 **해답**

두 개의 중첩된 for 루프를 포함하고 있습니다. 각 루프는 0부터 n까지 순회하므로, 전체 연산 횟수는 n^2이며, 시간 복잡도는 O(n^2)입니다.

▼ 예제 3

```
int Fibonacci(int n)
{
    if (n <= 1)
    {
        return n;
    }
    else
    {
        return Fibonacci(n - 1) + Fibonacci(n - 2);
    }
}
```

💡 **해답**
재귀적으로 피보나치 수열을 계산하는 함수입니다. 이 함수는 각 호출마다 두 번의 재귀 호출을 수행하므로, 시간 복잡도는 O(2^n)입니다.
이는 매우 비효율적인 방법으로, 실제 문제 해결에서는 동적 프로그래밍 등의 기법을 사용해 효율성을 높이는 것이 중요합니다.

04. 공간 복잡도 (Space Complexity)

✔️ How Many?

▼ 1) 공간 복잡도란?

- 코드의 메모리 사용량을 실제 메모리 크기(바이트)로 측정하는 것이 아니라, 입력 크기에 따라 필요한 저장 공간의 양을 측정하는 이유를 설명합니다.
- Big-O 표기법을 사용하여 표시함

▼ 2) 활용 예제

```
// 시간 복잡도: O(n)
int FindMax(int[] arr)
{
    int max = arr[0];

    for (int i = 1; i < arr.Length; i++)
    {
        if (arr[i] > max)
        {
            max = arr[i];
        }
    }

    return max;
}

// 시간 복잡도: O(n^2)
int FindMax2(int[] arr)
{
    for (int i = 0; i < arr.Length; i++)
    {
        bool isMax = true;

        for (int j = 0; j < arr.Length; j++)
        {
```

```
            if (arr[j] > arr[i])
            {
                isMax = false;
                break;
            }
        }

        if (isMax)
        {
            return arr[i];
        }
    }

    return -1;
}

int[] arr = new int[] { 1, 2, 3, 4, 5 };

// FindMax 메서드의 시간 복잡도: O(n), 공간 복잡도: O(1)
int max = FindMax(arr);

// 배열의 크기에 비례하여 실행 시간이 증가하므로 O(n)이라 할 수 있습니다. 또한 추가적인 메모리 공간을 사용하지 않기 때문에 공간 복잡도는 O(1)이라 할 수 있습니다.

// FindMax2 메서드의 시간 복잡도: O(n^2), 공간 복잡도: O(1)
int max2 = FindMax2(arr);

// 이중 반복문을 사용하기 때문에 배열의 크기에 제곱에 비례하여 실행 시간이 증가하므로 O(n^2)이라 할 수 있습니다. 이 알고리즘은 추가적인 메모리 공간을 사용하지 않기 때문에 공간 복잡도는 O(1)이라 할 수 있습니다.
```

05. 예제 문제

이 문제들을 풀면서 각각의 메서드의 시간 복잡도와 공간 복잡도를 계산해보세요.

▼ 문제 1

다음은 주어진 배열에서 특정 숫자를 찾는 메서드 `FindNumber` 입니다.
배열의 크기는 n이라고 가정합니다. 해당 숫자가 배열에 존재하면 true를 반환하고, 존재하지 않으면 false를 반환해야 합니다. 이때, 시간 복잡도와 공간 복잡도를 계산하세요.

```
public static bool FindNumber(int[] array, int target)
{
    for (int i = 0; i < array.Length; i++)
    {
        if (array[i] == target)
        {
            return true;
        }
    }
```

```
    return false;
}
```

▼ 시간 복잡도 와 공간 복잡도

- 시간 복잡도: O(n)
- 공간 복잡도: O(1)

▼ 문제 2

다음은 n개의 숫자로 이루어진 배열에서 최대값을 찾는 메서드 `FindMax` 입니다.
배열의 크기는 n이라고 가정합니다. 이때, 시간 복잡도와 공간 복잡도를 계산하세요.

```
public static int FindMax(int[] array)
{
    int max = int.MinValue;

    for (int i = 0; i < array.Length; i++)
    {
        if (array[i] > max)
        {
            max = array[i];
        }
    }

    return max;
}
```

▼ 시간 복잡도 와 공간 복잡도

- 시간 복잡도: O(n)
- 공간 복잡도: O(1)

▼ 문제 3

다음은 n개의 숫자로 이루어진 배열에서 중복된 숫자를 제거하는 메서드 `RemoveDuplicates` 입니다.
배열의 크기는 n이라고 가정합니다. 중복된 숫자가 제거된 새로운 배열을 반환해야 합니다. 이때, 시간 복잡도와 공간 복잡도를 계산하세요.

```
public static int[] RemoveDuplicates(int[] array)
{
    List<int> distinctList = new List<int>();

    foreach (int num in array)
    {
        if (!distinctList.Contains(num))
        {
```

```
            distinctList.Add(num);
        }
    }

    return distinctList.ToArray();
}
```

▼ 시간 복잡도 와 공간 복잡도

- 시간 복잡도: O(n)
- 공간 복잡도: O(n)

06. 알고리즘 시각화 참고 사이트

visualising data structures and algorithms through animation - VisuAlgo

VisuAlgo was conceptualised in 2011 by Dr Steven Halim as a tool to help his students better understand data structures and algorithms, by allowing them to learn the basics on their own and at their own pace. Together with his students from the National University of Singapore, a series of visualizations were

https://visualgo.net/en

이전 강의

14강 고급 자료형 및 기능

다음 강의

16강 정렬 알고리즘