알고리즘 스터디

ToBig's 11기 이영전

Ch07. 배열(Array)

- 자료구조 기본
- 배열
- 동적 배열
- 문제풀이!

Contents

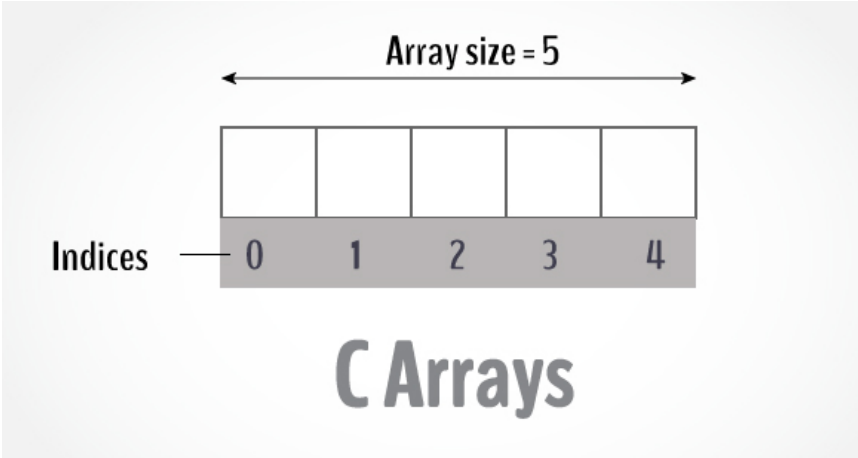
Unit 01 | Overview

- **자료구조**
  1. 메모리 공간 기반의 연속(Contiguous)방식
  2. 포인터 기반의 연결(Link) 방식

- 메모리 기반 연속(Contiguous)방식에는 배열(Array)이 있고
- 포인터 기반 연결 방식에는 연결 리스트(Linked List)가 있다

- 이번 시간에는 **메모리 기반 연속(Contiguous)방식**의 기본이 되는 배열(Array)에 대해 공부해보도록 한다!

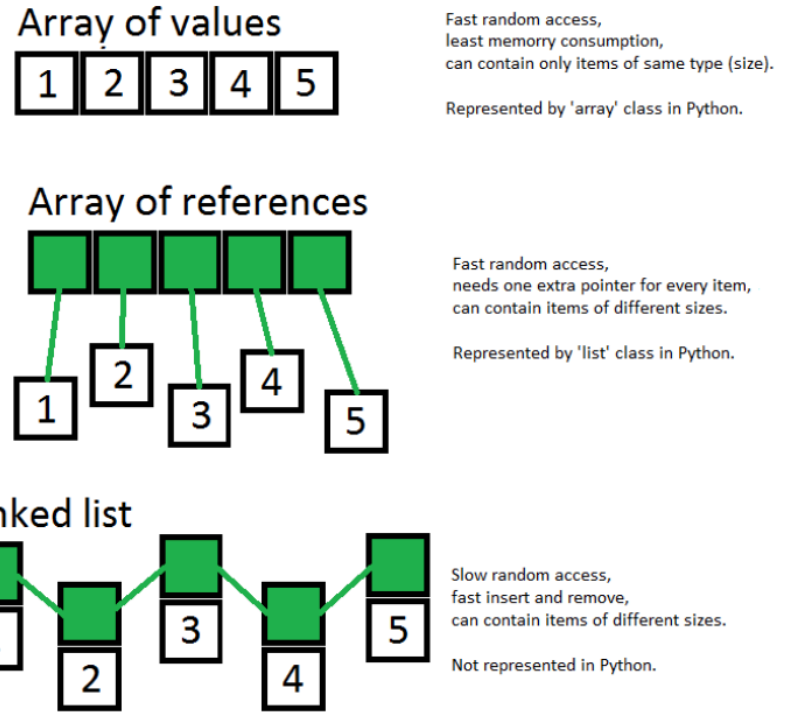
"배열은 값 또는 변수 엘레멘트의 집합으로 구성된 **자료구조**로, 하나 이상의 인덱스 또는 키로 식별된다."



https://www.programiz.com/c-programming/c-arrays

Unit 01 | Overview

- Python Array, Array of Reference and Linked List 비교
  - Array of values: python 'array' class
  - Array of references: python 'list' class
  - Linked List: Not represented in python



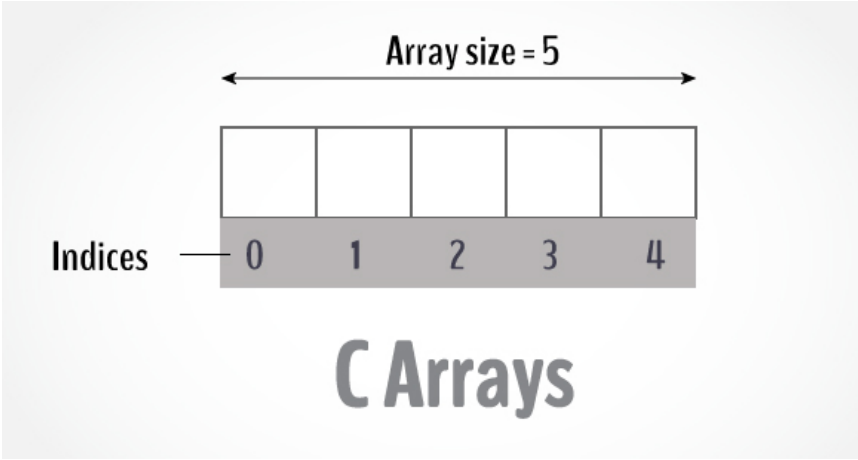
https://stackoverflow.com/questions/1514553/how-to-declare-an-array-in-python

Unit 02 | Data Structure & Array

- C언어를 기준으로 배열에 대해 알아보자 1
  (C는 자료구조의 가장 기본적인 형태를 소개하기 좋다!)

- 배열이란?
  - 크기를 지정하고 (고정된 크기)
  - 해당 크기 만큼의 연속된 메모리 공간을 할당 받는 작업을 수행하는 자료형

- C언어의 배열
  - 크기가 고정(크기를 지정한다)
  - 한 번 생성한 배열은 크기를 변경할 수 없다




```c
// C의 배열
// 다음 코드는 정수형 요소 5개로 이뤄진 배열을 생성함
// C

int arr[5] = {4, 7, 29, 0, 1};
```

https://www.programiz.com/c-programming/c-arrays

Unit 02 | Data Structure & Array

- C언어를 기준으로 배열에 대해 알아보자 2
  (C는 자료구조의 가장 기본적인 형태를 소개하기 좋다!)

- 배열 선언
  - Int의 크기는 4바이트이다
  - 메모리에 대한 접근은 바이트 단위로 한다
  - 메모리를 가리키는 주소는 1바이트마다 1씩 증가

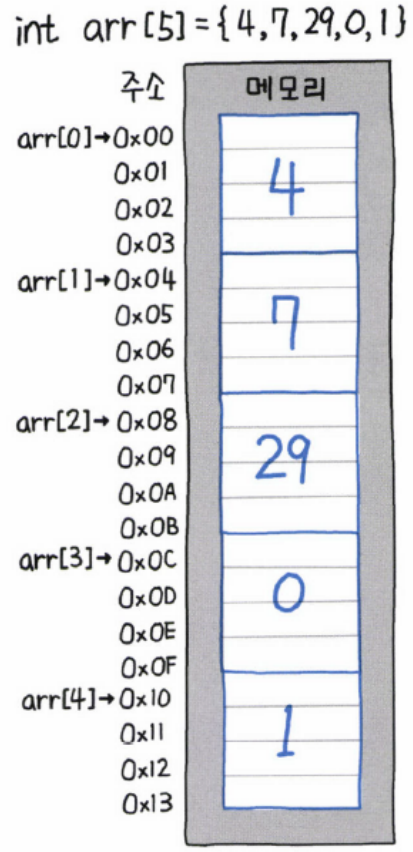


그림 7-1 물리 메모리에 순서대로 배치된 배열

Unit 02 | Data Structure & Array

e.g *그림7-1의 배열에서 4번째 값에 접근하려면 어떻게 해야할까?*

int 배열이므로 각각의 엘레멘트는 4바이트가 된다.
즉 (4-1)x4=12가 되고, 0x00에서 12만큼 증가한 16진수 0x0C가 된다

- 배열의 특징
  - 어느 위치에나 O(1)조회가 가능한 장점이 있다
  - 배열이 1억개가 되더라도 즉시 주소 계산 가능, 언제든 해당 메모리 주소에 있는 값을 O(1)에 조회 가능

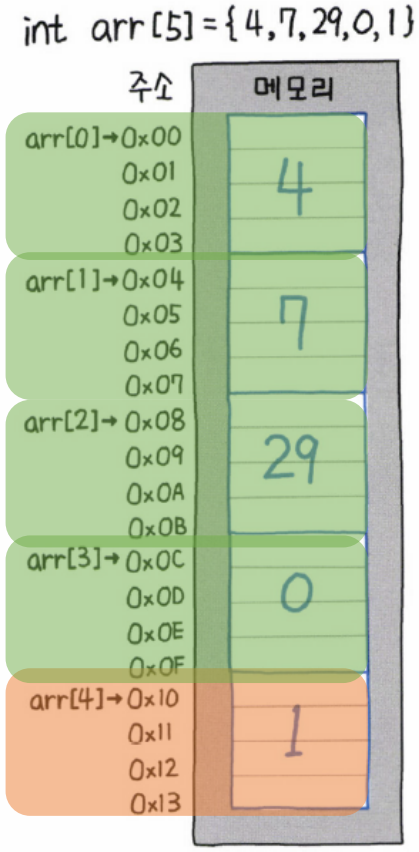


그림 7-1 물리 메모리에 순서대로 배치된 배열

Unit 03 | Dynamic Array

- 지금까지 배운 배열은…
  - 앞서 배열이란 고정된 크기만큼의 연속된 메모리 할당(Contiguous)이라고 설명했다
  - 크기를 지정하지 않고 자동으로 리사이징(resizing)하는 배열인 동적 배열이 등장했다

  - 대부분의 프로그래밍 언어에서는 동적배열을 지원하며 파이썬에서는 list가 대표적인 동적 배열 자료형이다

실제 데이터는 크기를 가늠하기 힘들지 않은가?
때로는 너무 작은 영역을 할당해서 모자라기도 하고,
때로는 너무 많이 할당하여 메모리 낭비가 일어나는데…

크기를 지정하지 않고
자동으로 조정할 수 있다면 좋지 않을까?

Unit 03 | Dynamic Array

- 동적 배열의 원리
  - 동적 배열의 원리는 간단하다
  - 미리 초깃값을 작게 잡아 배열을 생성한다
  - 데이터가 추가되어 배열이 꽉 채워지면, 늘려주고 복사한다
  - 이러한 과정을 더블링(Doubling)이라 하며 2배씩 늘려준다
    - 파이썬의 경우 CPython 내부 동적 배열 리스트에서는 0, 4, 8, 16, 25, 35, 46…순으로 배열을 재할당하도록 정의한다
    - 이러한 재할당비율을 '그로스 펙터(Growth Factor)'라 한다
  - 동적 배열은 분할 상환 분석(Amortized Analysis)에 따른 시간 복잡도를 설명하는 대표적인 자료형이기도 하다

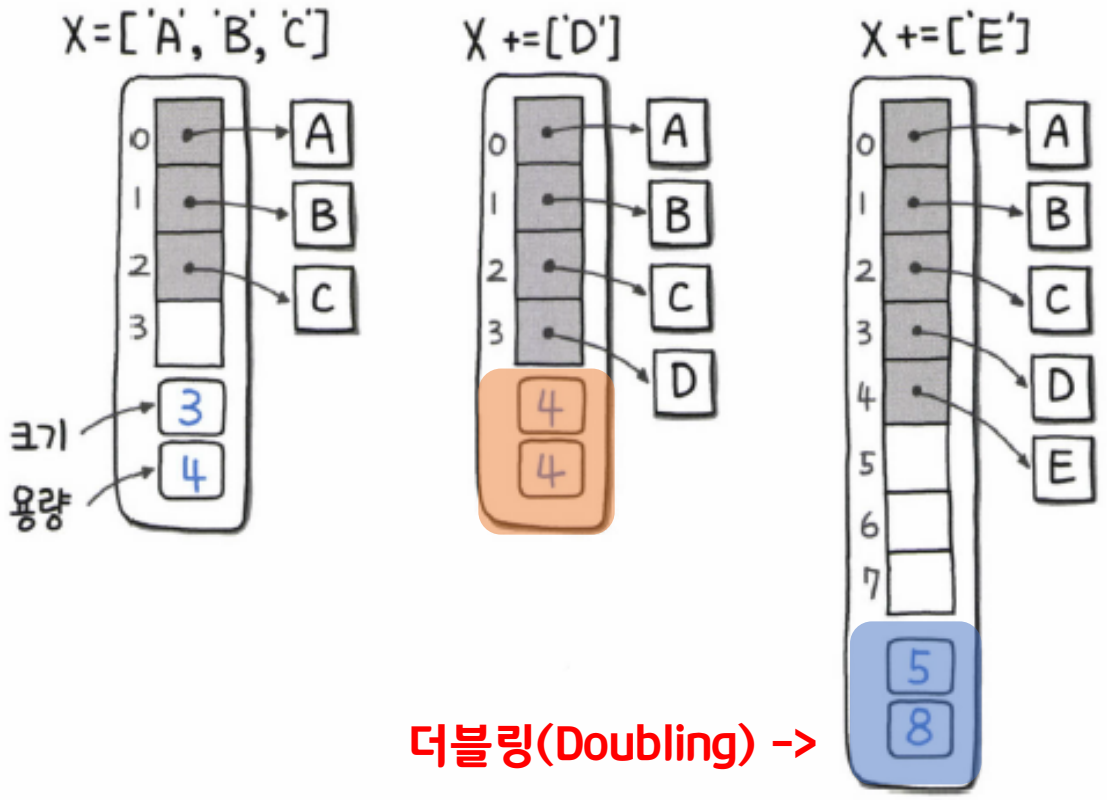
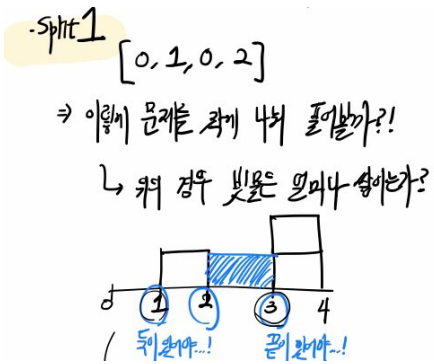


그림 7-2 동적 배열에 엘리먼트를 추가하는 구조[2]

Unit 04 | 08. 빗물 트래핑

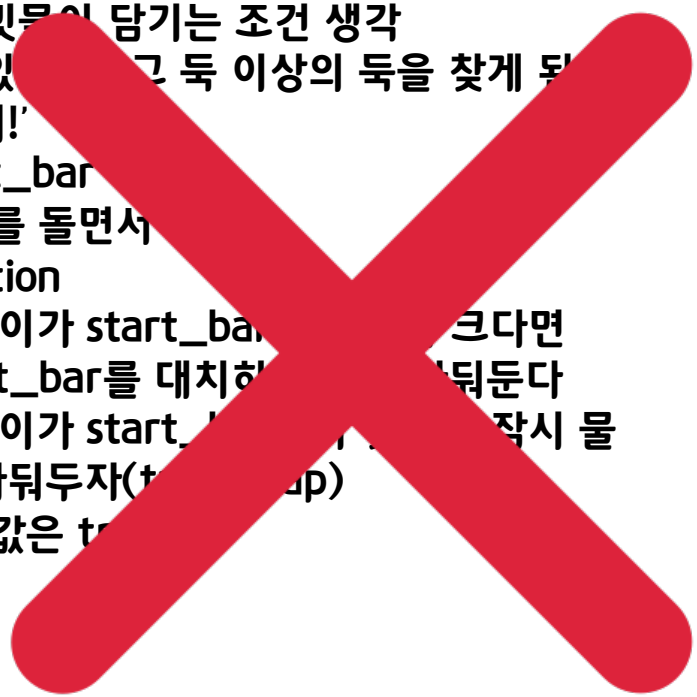
- **빗물 트래핑 – 1st Trial**
  - Test case에서 빗물이 담기는 조건 생각
  - '0보다 큰 둑이 있을 때, 그 둑 이상의 둑을 찾게 된다면 물이 쌓이겠지!'
    1) Find start_bar
    2) 전체 높이를 돌면서
       If condition
       1) If 높이가 start_bar와 같거나 크다면 start_bar를 대치하고 물을 가둬둔다
       2) If 높이가 start_bar보다 낮다면 잠시 물을 가둬두자(tmp_trap)
    3) 최종 리턴값은 trap변수




```python
class Solution(object):
    def trap(self, height):
        """
        :type height: List[int]
        :rtype: int
        """

        start_bar = 0
        tmp_trap = []
        trap = []

        for h in height:
            if h >= start_bar:
                start_bar = h
                trap.append(sum(tmp_trap))

            elif h < start_bar:
                tmp_trap.append(start_bar - h)

        return sum(trap)
```

Unit 04 | 08. 빗물 트래핑

- 빗물 트래핑 – 1st Trial
  - Test case에서 빗물이 담기는 조건 생각
  - '0보다 큰 둑이 있[illegible] 둑 이상의 둑을 찾게 되[illegible] 면 물이 쌓이겠지!'
    1) Find start_bar
    2) 전체 높이를 돌면서
       If condition
       1) If 높이가 start_ba[illegible] 크다면 start_bar를 대치하[illegible]워둔다
       2) If 높이가 start_[illegible] 잠시 물을 가둬두자(t[illegible]up)
    3) 최종 리턴값은 t[illegible]



문제점 발생! => Test Case에서는 잘 동작합니다만…

다음과 같이 start_bar의 높이보다 낮은 경우에도 물을 가둬둘 수 있다…!

우어어어…

Input:	[4,2,3]
Output:	0
Expected:	1

Unit 04 | 08. 빗물 트래핑

- 빗물 트래핑 – 1st Trial
  - Test case에서 빗물이 담기는 조건 생각
  - '0보다 큰 둑이 [illegible] 때, 그 둑 이상의 둑을 찾[illegible] 된다면 물이 쌓[illegible]
    1) Find start_b[illegible]
    2) 전체 높이를 돌[illegible]
       If condition:
       1) If 높이가 start_[illegible] 크[illegible] 면 start_bar를 [illegible] 물을 가둬 둔다
       2) If 높이가 st[illegible]_bar보다 [illegible] 작[illegible] 시 물을 [illegible]사(tmp_trap)
    3) 최종 리턴[illegible]ap변수

문제점 발생!

다음과 같이 start_bar의 높이보다 낮은 경우에도 물을 가둬둘 수 있다…!

우어어어…

```
Input:      [4,2,3]

Output:     0

Expected:   1
```

O(n**2)이 아닌
O(n)으로 풀 수 있는 두 가지 방법에 대해 알아보자!

Unit 04 | 08. 빗물 트래핑

- Solution 01 – 투 포인터(two pointer)를 최대로 이동
  - 투 포인터를 활용해 O(n)풀이를 해본다
  - 변수
    - volume: 물 높이
    - left, right: 좌.우 포인터 (투 포인터)
    - left_max, right_max: 좌.우 기둥 최대 높이
  - 풀이
    - 그림 7-5에서 가장 높은 막대의 최대 높이는 3 (높이가 더 높아도 상관 없다)

      Volume += left_max – height[left]
      Volume += right_max – height[right]

    - 이와 같이 최대 높이의 막대까지 좌우 기둥 최대 높이와 현재 높이의 차이만큼 물 높이를 더해간다

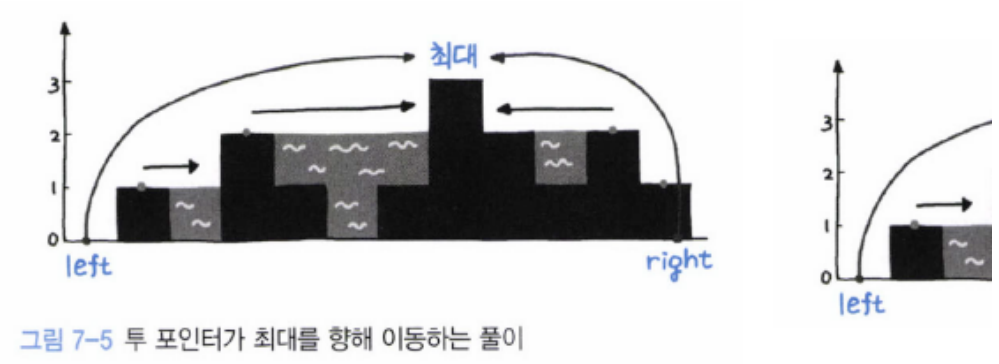


그림 7-5 투 포인터가 최대를 향해 이동하는 풀이

```python
class Solution(object):
    def trap(self, height):
        """
        :type height: List[int]
        :rtype: int
        """

        if not height:
            return 0

        volume = 0  # 부피의 초기값을 0으로 초기화한다
        left, right = 0, len(height) - 1
        left_max, right_max = height[left], height[right]

        while left < right:
            # left_max, right_max = max(height[left], left_max), max(heigth[right], right_max)
            left_max, right_max = max(height[left], left_max), max(height[right], right_max)
            # 더 높은 쪽을 향해 투 포인터 이동
            if left_max <= right_max:
                volume += left_max - height[left]
                left += 1
            else:
                volume += right_max - height[right]
                right -= 1
        return volume
```

Unit 04 | 08. 빗물 트래핑

- Solution 02 – 스택(Stack) 쌓기
  - 이번에는 스택을 활용해 O(n)풀이를 해본다
  - 변수
    - volume: 물 높이
    - stack: 여러 height의 인덱스가 들어있다
  - 풀이
    - 그림 7-6과 같이 스택에 쌓아 나가면서 현재 높이가 이전 높이보다 높을 때, 즉 꺾이는 변곡점(Inflection Point)을 기준으로 격차만큼 물 높이 volume을 채운다
    - 이 때 이전 높이가 들쑥날쑥하기 때문에 계속 스택으로 채워 나가다가 변곡점을 만날때마다 스택에서 하나씩 꺼내면서 이전과의 차이만큼 물 높이를 채워나간다

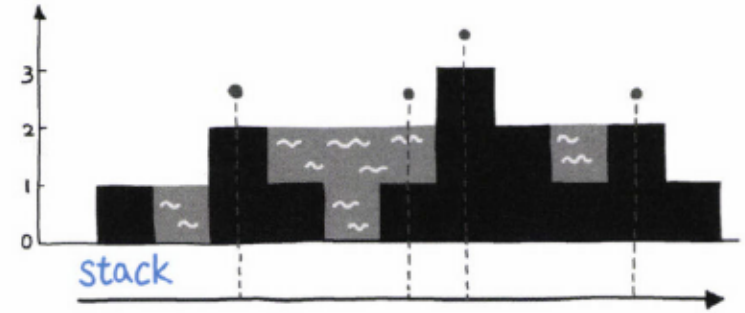


그림 7-6 스택에 쌓아 나가면서 풀이

```python
class Solution(object):
    def trap(self, height):
        """
        :type height: List[int]
        :rtype: int
        """

        stack = []  # 여러가지 height의 인덱스 저장
        volume = 0

        for i in range(len(height)):
            # 변곡점을 만나는 경우
            while stack and height[i] > height[stack[-1]]:  # 현재의 높이가 이전 높이보다 높으면
                # 스택에서 꺼낸다
                top = stack.pop()

                if not len(stack):  # stack이 비어있다면 break를 걸어준다
                    break

                # 이전과의 차이만큼 물 높이 처리
                distance = i - stack[-1] -1  # 물 부피의 길이: 현재 위치 - 마지막 인덱스(stack[-1]) -1
                waters = min(height[i], height[stack[-1]]) - height[top] # 물의 높이:

                volume += distance * waters

            stack.append(i)

        return volume
```

Unit 05 | 12. 주식을 사고팔기 가장 좋은 시점

- Solution 01. Brute-Froce O(n**2)
  - 가장 먼저 생각해볼 수 있는 방법은 역시 Brute-Force….
  - 하지만 시간 복잡도가 O(n**2)이라서 Runtime Error를 맞이하게 될 것이다…
  - 어떻게 하면 시간 복잡도를 줄일 수 있을까?

01. Solution1. Brute_Force

```
[13]  1 from typing import List
      2
      3 prices = [7, 1, 5, 3, 6, 4]
      4
      5 class Solution:
      6     def maxProfit(self, prices: List[int]) -> int:
      7         profit = [0]
      8         # max_profit = 0
      9         for i in range(len(prices)):
     10             for j in range(i+1, len(prices)):
     11                 if prices[i] > prices[j]:
     12                     pass
     13                 elif prices[i] < prices[j]:
     14                     profit.append(prices[j]-prices[i])
     15         return max(profit)
```

```
 1 a = Solution()
 2 a.maxProfit(prices)
```

```
5
```

Unit 05 | 12. 주식을 사고팔기 가장 좋은 시점

- Solution 02 – 최대, 최소 개념을 활용한 풀이
  - Python 내장함수 sys.maxsize활용
  - 빗물 트래핑에 비하면… Eeeeasy합니다!ㅎㅎ

02. Solution2.

```python
import sys

class Solution(object):
    def maxProfit(self, prices):
        """
        :type prices: List[int]
        :rtype: int
        """

        # 01. 최솟값, 최댓값을 활용해본다
        max_price = -sys.maxsize  # -infinite...
        min_price = sys.maxsize
        max_profit = 0

        for i in range(0, len(prices)):
            if prices[i] < min_price:  # min값 같은 경우에는 크게 문제가 되지 않는다.
                                       # max_price, max_profit을 어떻게 구할지 생각해보자.
                min_price = prices[i]

            elif prices[i] - min_price > max_profit:
                max_price = prices[i]
                max_profit = max_price - min_price

        return max_profit
```

```
[23]  1 a = Solution()
      2 a.maxProfit(prices)
```

```
5
```

들어주셔서 감사합니다.