

2018년 2학기 알고리즘 설계와 분석
CSE3081-01반 MP1 과제 보고서

학번: 20171665

이름: 이 선 호

2018. 10. 09

목 차

I | 실험 계획

1. 실험 환경	3
2. 실험 전제	5

II | 실험 결과

1. 개요	6
2. $O(n^2)$ 알고리즘	7
3. $O(n \log n)$ 알고리즘	8
4. $O(n)$ 알고리즘	9

III | 결론

1. 결론 및 논의	10
2. 추가 의견	11

I 실험 계획

1. 실험 환경

Sequence에서 가장 합이 큰 subsequence를 구하기 위한 실험을 진행할 PC의 시스템과 하드웨어 환경을 조사했다.

The screenshot shows the Windows System Information window. The left pane lists categories like Hardware Resources, System Summary, and Software Environment. The right pane displays a table of system details.

항목	값
OS 이름	Microsoft Windows 10 Home
버전	10.0.17134 빌드 17134
기타 OS 설명	사용할 수 없음
OS 제조업체	Microsoft Corporation
시스템 이름	DESKTOP-SK3OAM1
시스템 제조업체	ECS
시스템 모델	H110M4-C3D/C3V
시스템 종류	x64 기반 PC
시스템 SKU	Default string
프로세서	Intel(R) Core(TM) i5-6400 CPU @ 2.70GHz, 2701Mhz, 4 코어, 4 논리 프로세서
BIOS 버전/날짜	American Megatrends Inc. 5.11, 2015-11-20
SMBIOS 버전	3.0
포함된 컨트롤러 버전	255.255
BIOS 모드	레거시
BaseBoard 제조업체	ECS
BaseBoard 모델	사용할 수 없음
BaseBoard 이름	기판
플랫폼 역할	데스크톱
보안 부팅 상태	지원 안 됨
PCR7 구성	바인딩 불가능
Windows 디렉터리	C:\WINDOWS
시스템 디렉터리	C:\WINDOWS\system32
부팅 장치	\Device\HarddiskVolume2
지역	대한민국
하드웨어 추상화 계층	버전 = "10.0.17134.285"
사용자 이름	DESKTOP-SK3OAM1\griff
표준 시간대	대한민국 표준시
설치된 실제 메모리(RAM)	8.00GB
총 실제 메모리	7.95GB
사용 가능한 실제 메모리	3.99GB
총 가상 메모리	11.7GB
사용 가능한 가상 메모리	6.16GB
페이지 파일 공간	3.75GB
페이지 파일	C:\pagefile.sys
커널 DMA 보호	해제
가상화 기반 보안	사용 안 함
장치 암호화 지원	자동 장치 암호화에 실패한 이유: PCR7 바인딩이 지원되지 않음, 하드웨어 보안 테스...
Hyper-V - VM 모니터 모드 확장	예
Hyper-V - 두 번째 수준 주소 변환	예
Hyper-V - 편웨어에 가상화 사용	예
Hyper-V - Data Execution Protect	예

At the bottom, there are search fields: '찾을 내용(W):', '찾기(D)', and '찾기 닫기(C)'. There are also checkboxes for '선택한 범주만 검색(S)' and '범주 이름만 검색(R)'.

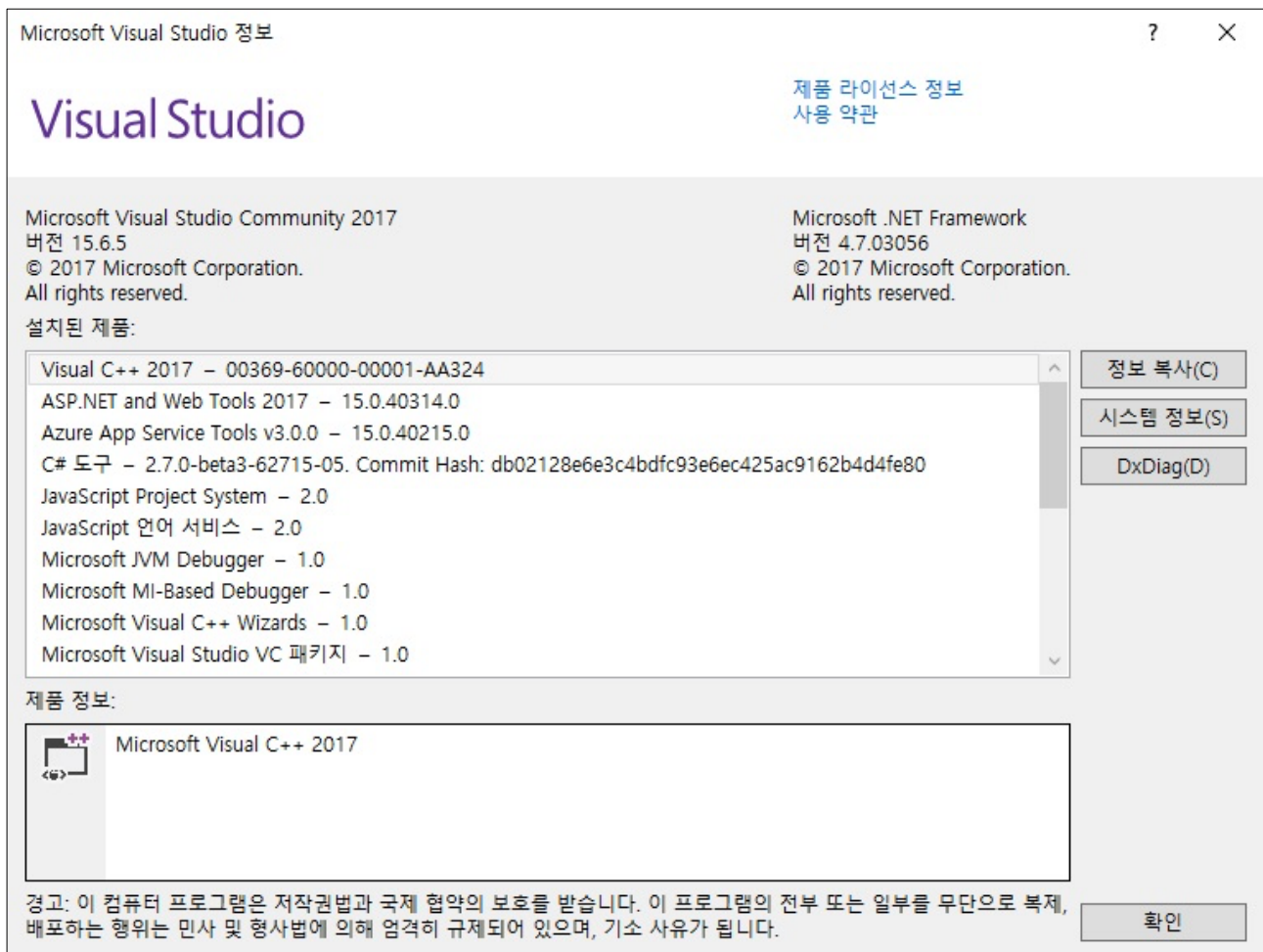
[그림 I -1] PC 시스템 환경과 하드웨어 정보

위의 정보를 간략히 요약하자면 [표 I -1]의 내용과 같다.

항목	값
OS 이름	Microsoft Windows 10 Home
버전	10.0.17134 빌드 17134
시스템 종류	64bit 운영체제, x64 기반 PC
프로세서	Intel(R) Core(TM) i5-6400 CPU @ 2.70GHz, 2701Mhz, 4코어, 4논리 프로세서
설치 메모리	8.00GB

[표 I-1] 시스템과 하드웨어 환경 사양 요약

소스 코드를 컴파일(Compile)하여 실행할 프로그램은 Microsoft에서 제작한 Visual Studio 2017이며, 이와 관련한 자세한 정보는 [그림 I-2]의 내용과 같다.



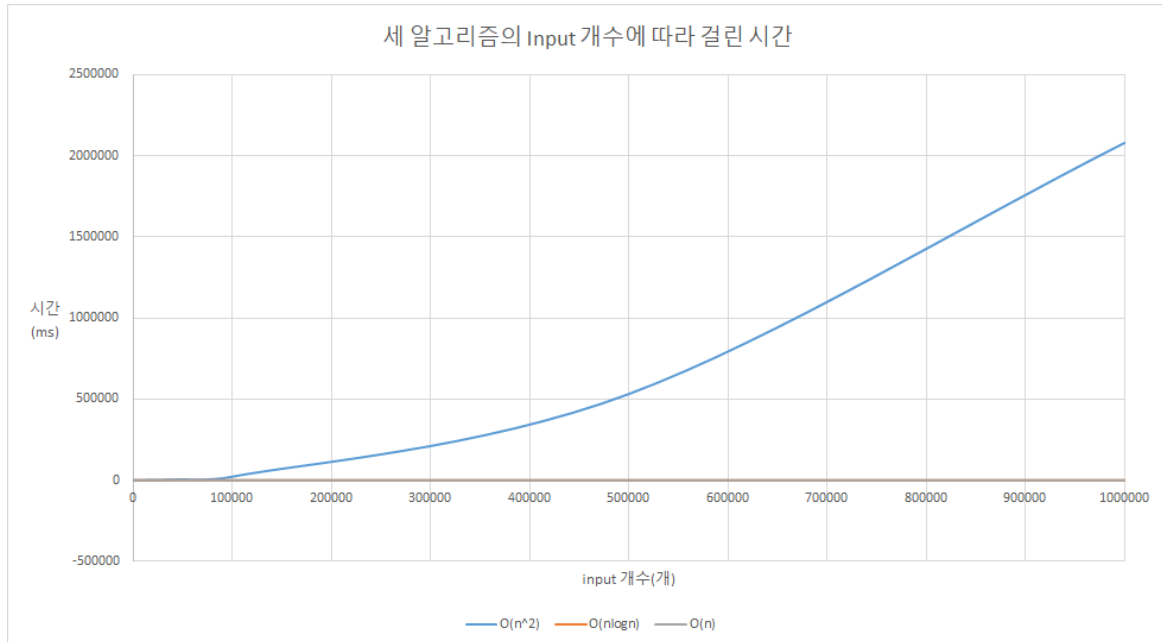
[그림 I-2] Visual Studio 환경 정보

2. 실험 전제

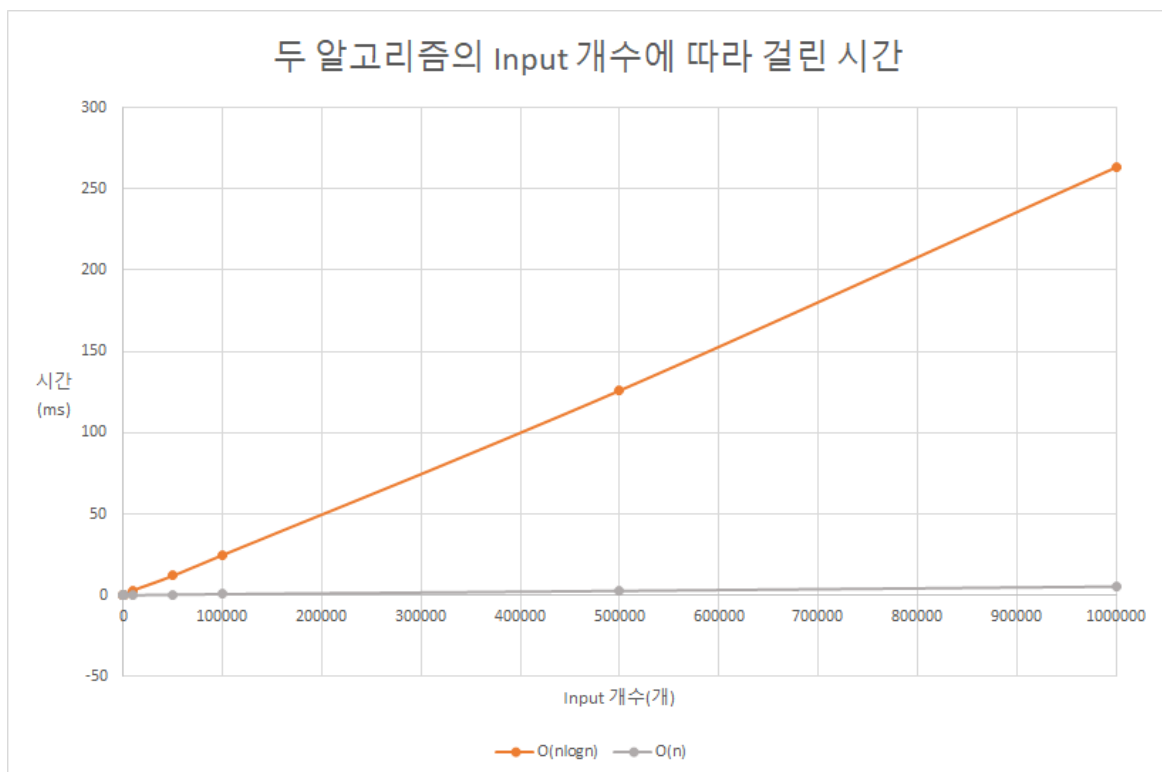
- ◆ 소스 코드는 C언어로 작성했으며, 하나의 파일로만 만들었다. Visual Studio 또는 Linux(Unix)에서 기본적으로 제공하는 라이브러리인 `stdio.h`, `stdlib.h`, `string.h`, 그리고 `time.h`를 이용했다.
- ◆ 메모리 낭비를 없애기 위해 소스 코드에서 input으로 받는 정보는 `malloc` 함수를 사용하여 동적으로 메모리를 할당했다.
- ◆ Linux 환경에서 파일 명령어와 함께 필요한 두 인자를 사용자가 입력해 `main` 함수에서 넘겨받을 수 있도록 변수 `argc`와 `argv`를 사용했다. Windows에서는 `command`로 직접 인수를 입력하는 것이 불가능하므로 디버깅 속성 설정에서 전달하는 인자를 입력하여 적용했다.
- ◆ 최대 입력받을 수 있는 sequence의 길이는 실험에서 안내한 2^{20} 을 초과하지 않기 위해 동적 할당으로 최대 2^{31} 까지 가능하다. sequence의 각 element는 integer 자료형으로 입력받으며, element의 value 범위는 $-2^{31} \sim 2^{31}$ 까지 가능하다.
- ◆ 3개의 알고리즘은 각각의 함수를 별도로 만들어서 실행하며 함수마다 최대 합(MaxSum), 부분열(subsequence)이 시작하는 인덱스(StartIndex), 끝나는 인덱스(EndIndex) 등 3개 이상의 값을 반환하기 때문에 return으로 반환되는 자료형은 void로 설정하였다.
- ◆ 알고리즘 실행 시간을 측정하기 위해 `time.h` 라이브러리에 있는 `clock` 함수를 사용했다. 시간 측정 단위는 millisecond로 설정하기 위해 알고리즘이 실행되기 시작하는 시간과 끝나는 시간의 interval을 구하여 `CLOCKS_PER_SEC`로 나눈 후 1000을 곱하였다.
- ◆ 각 알고리즘의 input의 개수에 따른 시간 복잡도(Time Complexity)를 조사하기 위해 순차적으로 1, 10, 10^2 , 10^3 , 10^4 , $10^4 \times 5$, 10^5 , $10^5 \times 5$, 10^6 개의 input에 따른 각각의 케이스를 실행했다. 실험 결과의 정확도를 높이기 위해 각각의 케이스마다 3번씩 시행하여 평균값을 구한 뒤 실험 결과에 반영했다.
- ◆ 실험에서 사용할 테스트 케이스는 numbergenerator.org의 Random Numbers Generator(<http://numbergenerator.org/>)를 이용했다. 그리고 실제로 프로그램이 잘 작동했는지에 관한 검증은 <https://ostermiller.org/calc/sum.html> 페이지를 이용했다.

II 실험 결과

1. 개요



[그림 II-1] 세 알고리즘의 그래프



[그림 II-2] $O(n \log n)$ 과 $O(n)$ 알고리즘 그래프

$O(n^2)$ 알고리즘이 $O(n \log n)$ 과 $O(n)$ 알고리즘보다 input 개수가 늘어날수록 시간 복잡도가 기하급수적으로 늘어난다는 사실을 알 수 있다.

[그림 II-1]의 그래프에서는 $O(n \log n)$ 과 $O(n)$ 알고리즘의 걸린 시간이 $O(n^2)$ 에 비해 현저히 적어서 한눈에 쉽게 파악하기 어렵다. 그래서 두 알고리즘의 비교 그래프를 [그림 II-2]에 작성했다.

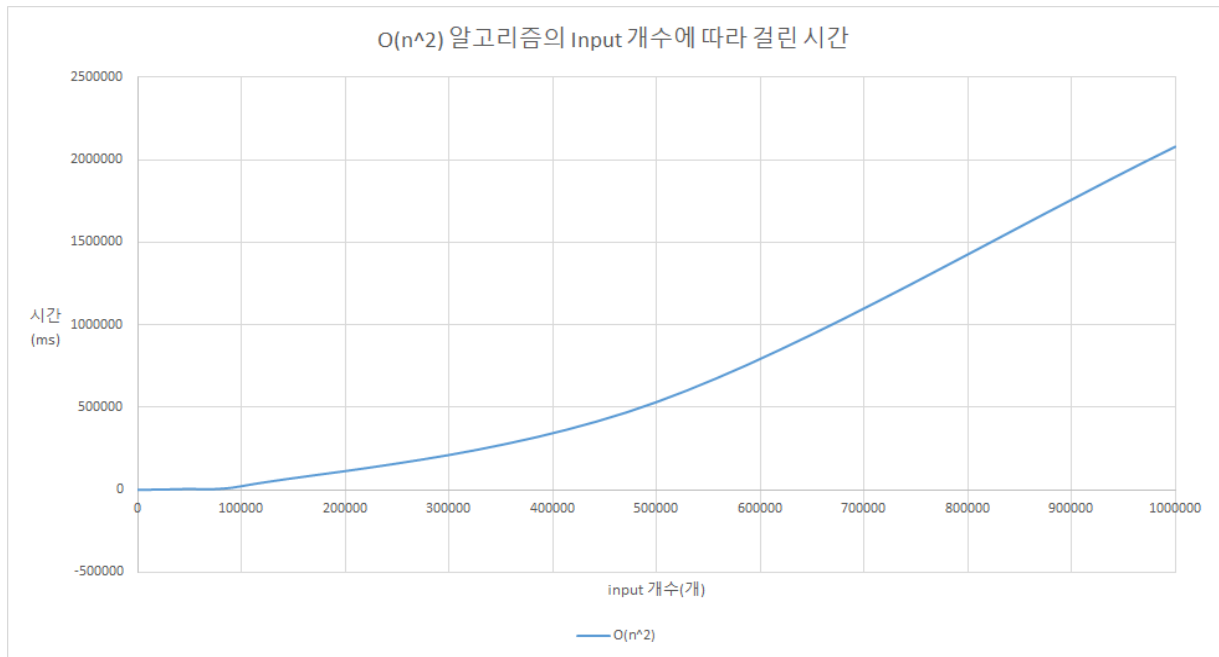
input 개수(개)	$O(n^2)$	$O(n \log n)$	$O(n)$
1	0	0	0
10	0	0	0
100	0	0	0
1000	3.333333	0.333333	0
10000	268.6667	3	0
50000	5450	12	0.333333
100000	21779	24.66667	0.666667
500000	531534.7	126	2.666667
1000000	2079822	263.3333	5.333333

[표 II-1] 세 알고리즘의 input 개수에 따라 걸린 시간 (시간 단위: milliseconds)

2. $O(n^2)$ 알고리즘

$O(n^2)$	첫 번째 실행	두 번째 실행	세 번째 실행	평균
1	0	0	0	0
10	0	0	0	0
100	0	0	0	0
1000	3	4	3	3.333333
10000	244	311	251	268.6667
50000	5464	5457	5429	5450
100000	21520	21900	21917	21779
500000	531347	531301	531956	531534.7
1000000	2079135	2079382	2080948	2079822

[표 II-2] $O(n^2)$ 알고리즘의 input 개수에 따라 걸린 시간 (시간 단위: milliseconds)



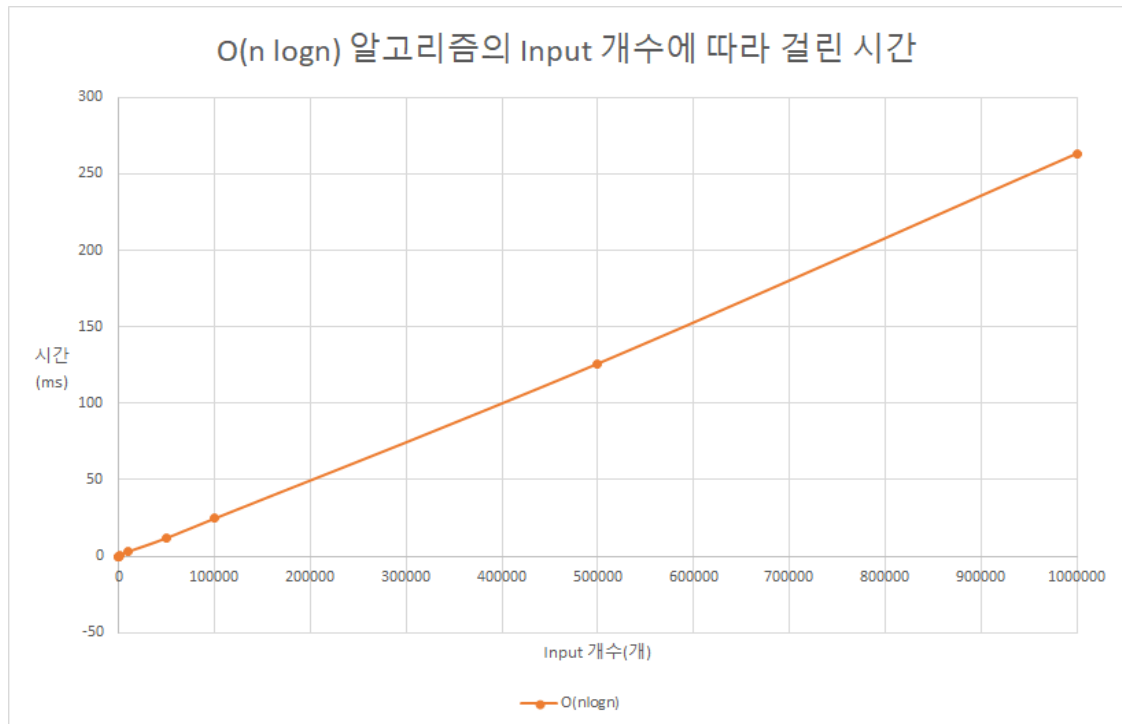
[그림 II-3] $O(n^2)$ 알고리즘 그래프

[그림 II-3]을 통해 $O(n^2)$ 알고리즘 그래프 개형이 이차곡선 포물선 형태를 그리는 것을 볼 수 있다. 그리고 input 개수에 따라 걸린 시간이 기하급수적으로 늘어났다.

3. $O(n \log n)$ 알고리즘

$O(n \log n)$	첫 번째 실행	두 번째 실행	세 번째 실행	평균
1	0	0	0	0
10	0	0	0	0
100	0	0	0	0
1000	1	0	0	0.333333
10000	3	3	3	3
50000	13	12	11	12
100000	23	26	25	24.66667
500000	121	128	129	126
1000000	262	264	264	263.3333

[표 II-3] $O(n \log n)$ 알고리즘의 input 개수에 따라 걸린 시간 (시간 단위: milliseconds)



[그림 II-4] $O(n \log n)$ 알고리즘 그래프

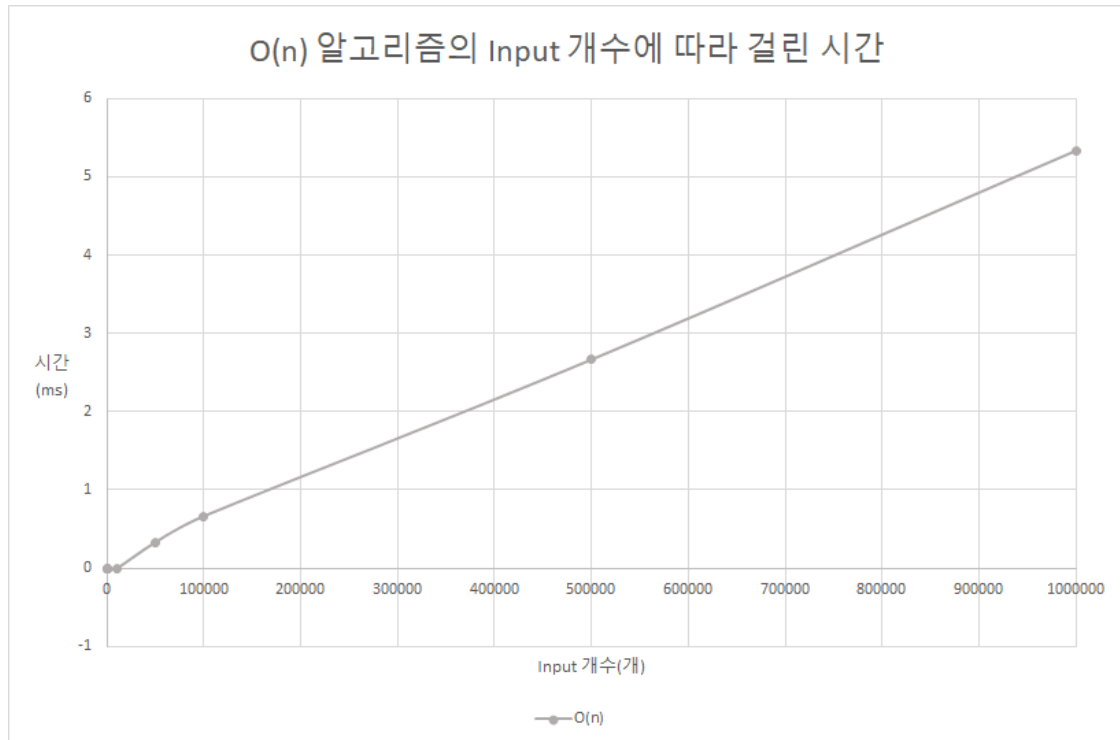
[그림 II-4]을 통해 $O(n \log n)$ 알고리즘 그래프 개형이 input 개수가 늘어남에 따라 45° 보다 점차 늘어난다는 점을 알 수 있다.

4. $O(n)$ 알고리즘

$O(n)$	첫 번째 실행	두 번째 실행	세 번째 실행	평균
1	0	0	0	0
10	0	0	0	0
100	0	0	0	0
1000	0	0	0	0
10000	0	0	0	0
50000	1	0	0	0.333333
100000	1	1	0	0.666667
500000	3	2	3	2.666667
1000000	6	5	5	5.333333

[표 II-4] $O(n)$ 알고리즘의 input 개수에 따라 걸린 시간 (시간 단위: milliseconds)

[그림 II-5]을 통해 $O(n)$ 알고리즘 그래프 개형이 input 개수가 늘어남에 따라 큰 기울기 변화 없이 일정하게 걸린 시간이 linear하게 증가한다는 점을 확인할 수 있다.



[그림 II-5] $O(n)$ 알고리즘 그래프

III 결론

1. 결론 및 논의

- ♦ $O(n^2)$ 알고리즘의 걸리는 시간이 다른 두 알고리즘보다 input의 element 수가 늘어날수록 급격히 증가한다는 사실을 알 수 있다.
- ♦ $O(n^2)$ 알고리즘처럼 1을 초과하는 지수를 시간 복잡도 식의 항으로 갖는 알고리즘이 input의 수가 늘어날수록 걸리는 시간이 기하급수적으로 커진다는 점을 예측할 수 있다. 반대로 시간 복잡도 식의 항에서 1보다 작거나 같은 지수를 갖고나 \log

항을 갖는 식을 갖는 알고리즘은 input의 수가 늘어난다고 하더라도 걸린 시간이 크게 증가하지 않는 점을 알 수 있다.

- ♦ input에 개수가 커질수록 세 알고리즘의 걸리는 시간을 서로 비교하면 $O(n^2)$ 이 가장 크고, 그 다음에 $O(n\log n)$, 마지막으로 $O(n)$ 임을 알 수 있다.
- ♦ input 개수가 매우 작으면 세 알고리즘의 걸리는 시간은 상대적으로 큰 차이가 없음을 유추할 수 있다.

2. 추가 의견

- ♦ 알고리즘이 걸린 시간의 측정 단위를 milliseconds로 해서 input 개수가 매우 작을 때에는 걸린 시간이 0으로 나와서 세 알고리즘의 걸린 시간을 자세히 비교가 힘들었다는 점이 아쉬웠다. 특히 $O(n)$ 알고리즘 그래프에서 input 개수가 작을 때 비교가 어렵고 시간 결과가 0.3333 또는 0.6666 등 한정된 값으로만 나오게끔 되어서 정확도가 낮아진다. 이를 위해서는 milliseconds까지만 지원하는 clock 함수를 사용하는 것보다 nanoseconds까지 지원 가능한 gettimeofday나 clock_gettime 함수를 사용하는 것이 더 나을 것으로 예상된다.