알고리즘 과제 1

20161662 허남규.

목차

개요

이번 과제에서는 MSS 문제를 해결하기 위해 시간 복잡도가 다른 알고리즘 3개를 구현하여 입력 크기에 따른 수행 시간의 변화를 확인합니다.

수행 환경

- CPU: Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz
  - 코어 개수: 4
  - L2 캐시: 256 KB
  - L3 캐시: 6 MB
- RAM: 8 GB
- OS: macOS Sierra 10.13.6

상세

- 입력 크기: 2^1 - 2^{20} (2배씩 증가)
- 입력 순서: 임의
- 측정 항목: 수행 시간

구현

이번 과제는 모듈 단위로 구현했습니다. 독립적인 기능을 모듈로 나누고 각각의 모듈 인터페이스에 대한 다큐멘테이션을 철저히 함으로써 주석의 필요성을 최소화했습니다. 모듈별 다큐멘테이션은 '.h' 파일 내 함수 선언 부분에서 주석으로 추가했습니다. '.c' 파일 내 함수 정의 부분에서는 적절한 변수 이름을 사용하여 가독성을 높였습니다. '.c' 파일에서 주석은 가독성에 기여가 되는 상황에 한하여 제한적으로 추가했습니다.

spec.h 인터페이스

mss 문제 풀이에 대한 인터페이스를 선언합니다.

```
typedef struct _mss_result {
      int start;
      int end;
      int sum;
} mss_result;

typedef mss_result mss_function(int *data, int size);
```

위와 같이 mss 문제에 대한 정답을 담는 구조체, 그리고 mss 풀이 함수의 형식을 정의합니다.

mss.h 모듈

실제 mss 풀이 함수 3개가 구현된 모듈입니다. 3개의 함수 모두 spec 모듈에서 정의한 mss_function 형식을 따라갑니다.

input.h 모듈

mss 문제의 입력 데이터를 생성하고, 저장하고, 불러오는 기능을 제공합니다.

generate.c 모듈

전체 프로그램에는 포함되지 않는 모듈로, 임의의 입력 데이터를 생성하는 모듈입니다. Makefile을 이용하여 다음과 같이 컴파일하고 실행할 수 있습니다.

```
make generator
./generator
```

혹은 다음 Makefile명령어를 이용하여 analysis 폴더 안에 데이터를 바로 생성할 수 있습니다.

```
make generate
```

유닛 테스트

모듈별로 ifdef 매크로로 감싸진 유닛 테스트 코드가 포함되어 있습니다. 다음과 같이 gcc의 -D 옵션을 사용하여 모듈의 유닛 테스트를 컴파일하고 실행할 수 있습니다.

```
gcc -D TEST mss.c
./a.out
```

Makefile을 이용하여 다음과 같이 모듈별 유닛 테스트를 일괄적으로 실행할 수 있습니다.

```
make unittest
```

Python 기반 테스트 모듈

위의 유닛 테스트와 별개로, 수동으로 작성된 테스트케이스를 가지고 구현 프로그램 전체를 테스트하는 기능이 tests/test.py에 구현되어 있습니다. python3와 parse 라이브러리를 필요로 합니다.

Python 기반 분석 모듈

자동화된 알고리즘 수행 시간 분석을 위해 입력 데이터 생성, 알고리즘 수행, 결과 출력을 일괄적으로 처리하는 기능이 analysis/analyze.py에 구현되어 있습니다. 마찬가지로 python3와 parse 라이브러리를 필요로 하여, 결과는 results.csv로 저장됩니다.

결과 – 알고리즘별 수행 시간

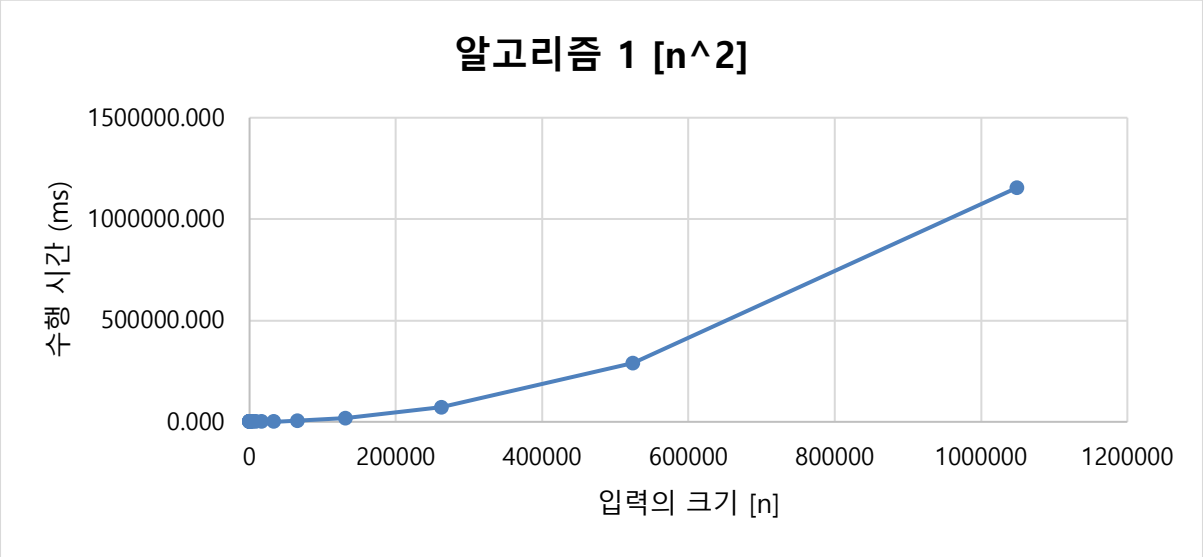
아래 표를 통해 알고리즘별로 수행 시간의 증가 추세를 확인할 수 있습니다.

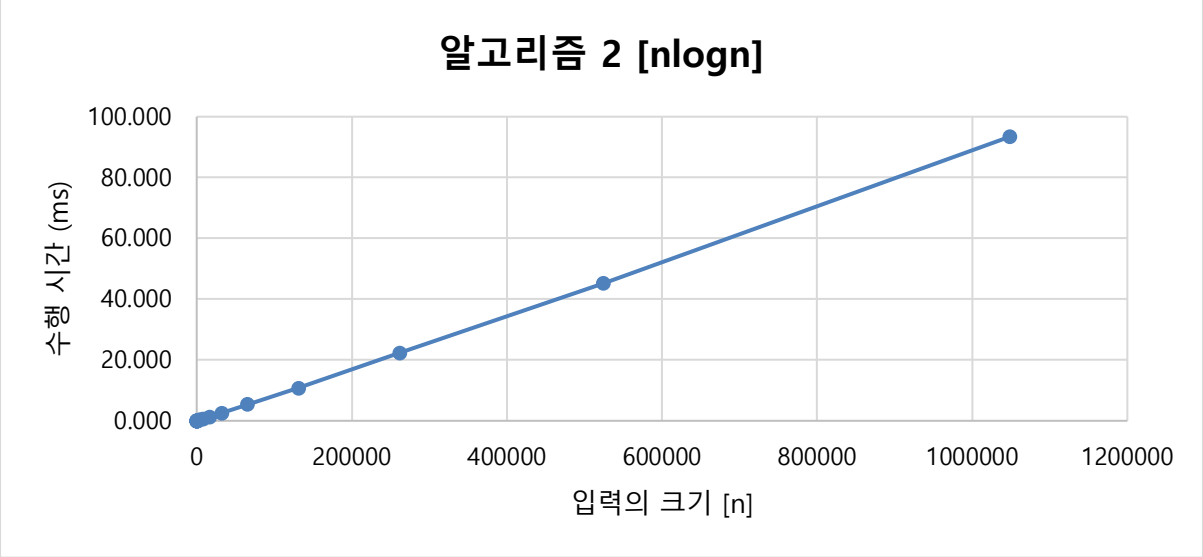
결과 표

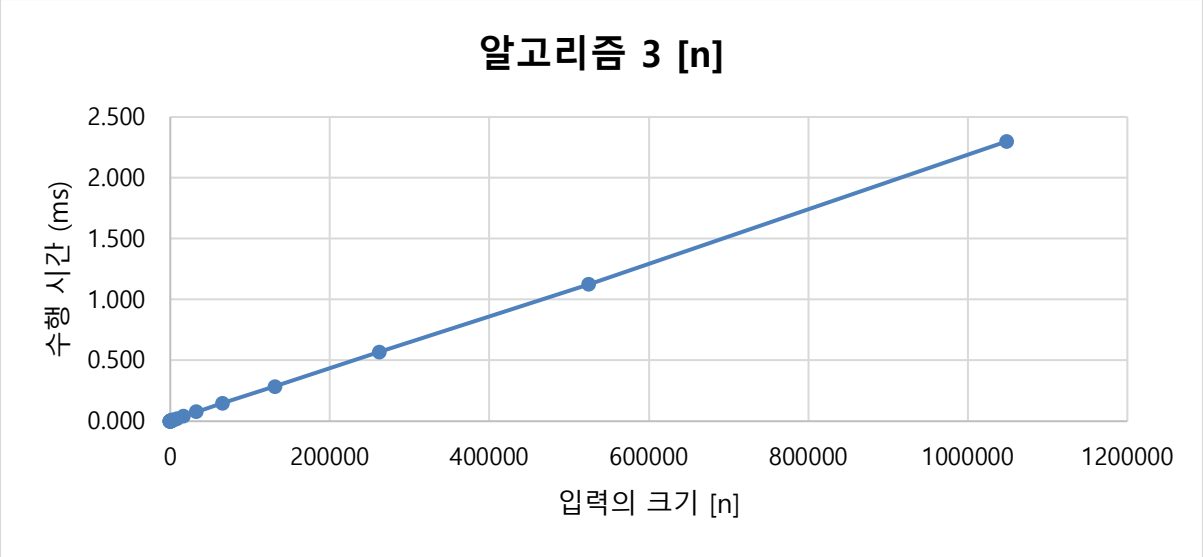
입력 크기	수행 시간		
	n2	nlogn	n
2	0.001	0.002	0.000
4	0.001	0.002	0.001
8	0.002	0.002	0.002
16	0.002	0.003	0.001
32	0.003	0.004	0.001
64	0.007	0.007	0.002
128	0.020	0.012	0.002
256	0.074	0.021	0.002
512	0.283	0.040	0.003
1024	1.085	0.078	0.005

2048	4.438	0.150	0.008
4096	17.743	0.304	0.012
8192	69.118	0.652	0.023
16384	280.231	1.259	0.041
32768	1121.899	2.577	0.077
65536	4482.835	5.364	0.147
131072	17980.084	10.812	0.284
262144	72064.547	22.322	0.568
524288	288732.250	45.167	1.125
1048576	1153830.000	93.428	2.299

결과 그래프







결과 분석

예상한 바와 같이 세 알고리즘 모두 시간 복잡도에 상응하는 추세로 증가하는 모습을 확인할 수 있습니다. 모양만 두고 봤을 때 nlogn과 n는 시각적으로 구별하기 힘들지만 절대적인 수행 시간은 n보다 높게 나타남을 확인할 수 있습니다. 이는 logn 항 뿐만 아니라 함수 호출, 잦은 cache miss 등 divide-and-conquer 방식의 알고리즘에서 발생하는 overhead로 인한 것으로 예상할 수 있습니다.

입력의 크기가 작은 경우에는 수행 시간이 단조롭게 증가하지 않는 점을 확인할 수 있는데, 이는 단순히 측정에서 발생한 오차로 볼 수 있습니다. 입력의 크기가 커짐에 따라 연산 반복 횟수가 증가하면서 오차가 상쇄되어 수행 시간의 변화가 안정화되는 모습을 확인할 수 있습니다.