CSE3081-01 알고리즘 설계와 분석

[숙제 1]

Detection of Brightest Area in 2D Image

20231604 컴퓨터공학과 정유연

2024-09-30

<목차>

1. 컴퓨터 실험환경

2. 알고리즘

3. 메인 함수, 변수, function, 출력

4. 결과값, 출력 파일 확인

5. 실행 결과 및 수행 시간

1. 컴퓨터 실험환경

- OS: Windows 11 Home
- CPU: 13th Gen Intel(R) Core(TM) i5-1340P 1.90 GHz
- RAM: 16.00GB
- Compiler: Visual Studio 22 Release Mode/x64 Platform

제출한 프로젝트 파일 이름: HW1

작성한 소스파일: HW1₩HW1₩FileName.c

HW1₩HW1₩HW1.vcxproj 눌러 실행

```
FILE* open_file(const char* filename, const char* mode) {
    char FileName[100] = "Data\\";
    strcat(FileName, filename);

    FILE* stream = fopen(FileName, mode);
    if (stream == NULL) {
        perror("FILE OPEN ERROR");
```

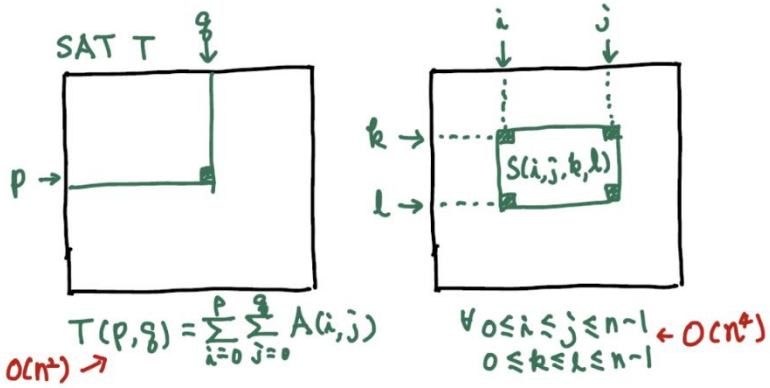
위 open_file로 파일을 open함. Data₩디렉토리에 데이터를 넣어야 함.

2. 알고리즘

Algorithm 3	int algo3(int **A)	Summed Area-Table 기법 적용 방법	O(n^4)

2차원 배열 T에 배열 A의 부분 누적합을 저장한다. A[0][0]부터 A[j]A[i]까지 j X i 크기의 직사각형의 부분합이 저장된다. Table construction을 하는 데에 2중 루프를 이용하므로 O(n^2)시간이 소요된다.

이후 4중 루프를 통해 배열의 시작 열 i, 끝 열 j, 시작 행 k, 끝 행 l을 설정한다. O(n^4)시간 소요된다.



summed area-table기법을 이용해 왼쪽 위 인덱스 A[k][i]부터 오른쪽 아래 인덱스 A[l][j]까지 부분합을 s에 저장하여 주어진 maxSum과 대소 비교를 한다.

구하는 부분합 s는 위쪽 그림과 같다.

부분합 s= T[i][j] - T[k – 1][j] - T[i][i – 1] + T[k – 1][i - 1]

아래와 같이 4중 for문으로 인덱스를 설정한다.

```
for (int i = 0; i < wid; i++) {
    for (int j = i; j < wid; j++) {
        for (int k = 0; k < hei; k++) {
            for (int l = k; l < hei; l++) {
                int s = T[l][j];
                if (k) s -= T[k - 1][j];
                if (i) s -= T[l][i - 1];
                if (k > 0 && i > 0)s += T[k - 1][i - 1];
                if (s > maxSum) {
```

Algorithm 4	int algo4(int** A)	1D 문제를 풀기 위하여 Divide-and-conquer기법을 적용한 방법	O(n^3 log n)

Divide-and-conquer기법을 적용한 max1D함수를 이용해 각 열의 최대합을 구한다. max1D함수는 하나의 1차원 배열을 크기가 0또는 1인 여러 개의 배열로 분할한 후 병합하며 최대합을 구하는 분할정복 알고리즘이다. 이진트리 형태로 배열을 계속 반으로 나누기 때문에 재귀 호출의 깊이는 O(log n)이다. 합병하는 과정에서 O(n)시간이 걸리므로 O(n log n)을 갖는다.

이제 2중 루프를 이용해 left부터 right까지의 열 중에서 최대합을 찾는다. 가장 안에 0부터 ROW크기만큼의 반복문을 이용하여 temp배열에 left부터 right까지 열의 행 값을 모두 더한다. max1D함수를 불러와서 temp배열에서 최대합을 찾고, 주어진 maxSum과 비교하며 최대값을 찾는다.

시작점 left는 0부터 m-1까지, 끝점 right는 left에서 m-1까지 이동하므로 n(n+1)/2 ~= O(n^2)를 갖는다. 가장 안쪽 반복문에서는 max1D함수를 호출하기에 시간복잡도는 O(n^3 log n)이다.

Algorithm 5	int algo5(int **A)	1D 문제를 풀기 위하여 Dynamic programming 기법을 적용한 방법 (Kadane's Algorithm)	O(n^3)

Dynamic programming기법을 적용한 kadanes알고리즘을 이용해 각 열의 최대합을 구한다. kadane알고리즘은 이전 행까지의 최대합을 저장하고, 이 값에 현재 행의 원소값을 더하여 비교함으로써 최대합을 구하는 알고리즘이다. 배열의 모든 원소를 한번씩 방문하므로 O(n)을 갖는다.

이제 2중 루프를 이용해 left부터 right까지의 열 중에서 최대합을 찾는다. 이때 가장 안에 0부터 ROW크기만큼의 반복문을 이용해 temp배열에 left부터 right까지 열의 행 값을 모두 더한다. kadanes함수를 불러와서 temp배열에서 최대합을 찾고, 주어진 maxSum과 비교하며 최대값을 찾는다.

시작점 left는 0부터 m-1까지, 끝점 right는 left에서 m-1까지 이동하므로 n(n+1)/2 ~= O(n^2)를 갖는다. 가장 안쪽 반복문에서는 kadanes함수를 호출하기에 시간복잡도는 O(n^3)이다.

3. 메인 함수, 변수, function, 출력

메인함수 void main(void)에서 입력 파일들

FILE* fp	입력 파일, HW1_config.txt를 연다.

int n	테스트 케이스 개수
int num	사용할 알고리즘 번호
char pgm_input[100]	입력 영상 파일 이름
FILE* fpgm	입력 영상 파일
char avg_input[100]	입력 영상의 평균값 파일 이름
FILE* favg	입력 영상의 평균값 파일
char bri_input[100]	그에 대한 결과 값 출력 파일 이름
FILE* fbri	그에 대한 결과 값 출력 파일

변수

int** twoD	fpgm파일에서 읽어들일 2D배열
int avg_val	favg에서 받은 평균값. twoD배열의 모든 값에 이 숫자를 뺀다.
int wid	입력배열의 열의 개수
int hei	입력배열의 행의 개수
int res	maximum sum subrectangle의 크기
int index_k	maximum sum subrectangle의 위-왼쪽 모서리 행
int index_i	maximum sum subrectangle의 위-왼쪽 모서리 열
int index_l	maximum sum subrectangle의 아래-오른쪽 모서리 행
int index_j	maximum sum subrectangle의 아래-오른쪽 모서리 열

함수(function)

int algo3(int **A)	Summed Area-Table 기법 적용 방법
int max1D(const int A[], int left, int right, int *startIdx, int *endIdx	하나의 1차원 배열을 크기가 0또는 1인 여러 개의 배열로 분할한 후 병합하며 최대합을 구함
int algo4(int** A)	Divide-and-conquer기법을 적용한 max1D함수를 이용해 2D 배열의 부분 최대값을 찾음
int kadanes(int A[], int n, int* startIdx, int* endIdx	Dynamic programming을 이용해 부분배열의 최대합을 구함

int algo5(int **A)	kadanes함수를 이용해 2D 배열의 부분 최대 값을 찾음. Kadane's algorithm을 이용했다.
FILE* open_file(const char* filename, const char* mode)	파일을 fopen한다.

출력

```
CHECK_TIME_START(_start, _freq);
int res = 0;
if (num == 3) {
    res = algo3(twoD);
}
else if (num == 4) {
    res = algo4(twoD);
}
else if (num == 5) {
    res = algo5(twoD);
}
else exit(0);
CHECK_TIME_END(_start, _end, _freq, _compute_time);

printf("%d %d %d %d %dWn", index_k, index_i, index_l, index_j, res);
fprintf(fbri, "%d %d %d %d %d", index_k, index_i, index_l, index_j, res);
printf("run time = %.3fms Wn", _compute_time);
```

#include "measure_cpu_time.h"를 통해 알고리즘 실행 전과 알고리즘 실행 후 시간을 측정해 출력했다

index_k, index_i, index_l, index_j, res를 순서대로 moon_n_brightest_num.txt에 fprint했다.

4. 실행 결과

- n=64,128,256일 때 알고리즘별 실행결과

```
Open cartoon_64.pgm by Algorithm 3
18 26 63 36 20772
run time = 17.151ms
Open cartoon_64.pgm by Algorithm 4
18 26 63 36 20772
run time = 10.637ms
Open cartoon_64.pgm by Algorithm 5
18 26 62 36 20450
run time = 0.961ms
Open cartoon_128.pgm by Algorithm 3
35 52 127 73 77231
run time = 251.688ms
Open cartoon_128.pgm by Algorithm 4
35 52 127 73 77231
run time = 77.110ms
Open cartoon_128.pgm by Algorithm 5
35 52 126 73 76494
run time = 6.204ms
Open cartoon_256.pgm by Algorithm 3
90 104 255 145 298722
run time = 4471.901ms
Open cartoon_256.pgm by Algorithm 4
90 104 255 145 298722
run time = 778.001ms
Open cartoon_256.pgm by Algorithm 5
90 104 254 145 297064
run time = 39.890ms

C:\Users\wjddb\Desktop\Algo1\x64\Release\Algo1.exe
```

```
Open cartoon_64.pgm by Algorithm 3
18 26 63 36 20772
run time = 21.604ms
Open cartoon_64.pgm by Algorithm 4
18 26 63 36 20772
run time = 14.006ms
Open cartoon_64.pgm by Algorithm 5
18 26 62 36 20450
run time = 1.311ms
Open cartoon_128.pgm by Algorithm 3
35 52 127 73 77231
run time = 291.850ms
Open cartoon_128.pgm by Algorithm 4
35 52 127 73 77231
run time = 127.471ms
Open cartoon_128.pgm by Algorithm 5
35 52 126 73 76494
run time = 12.040ms
Open cartoon_256.pgm by Algorithm 3
90 104 255 145 298722
run time = 5278.343ms
Open cartoon_256.pgm by Algorithm 4
90 104 255 145 298722
run time = 859.784ms
Open cartoon_256.pgm by Algorithm 5
90 104 254 145 297064
run time = 44.556ms

C:\Users\wjddb\Desktop\Algo1\x64\Release\Algo1.exe
```

```
Open cartoon_64.pgm by Algorithm 3
18 26 63 36 20772
run time = 24.379ms
Open cartoon_64.pgm by Algorithm 4
18 26 63 36 20772
run time = 13.381ms
Open cartoon_64.pgm by Algorithm 5
18 26 62 36 20450
run time = 1.144ms
Open cartoon_128.pgm by Algorithm 3
35 52 127 73 77231
run time = 299.450ms
Open cartoon_128.pgm by Algorithm 4
35 52 127 73 77231
run time = 89.977ms
Open cartoon_128.pgm by Algorithm 5
35 52 126 73 76494
run time = 9.404ms
Open cartoon_256.pgm by Algorithm 3
90 104 255 145 298722
run time = 5026.804ms
Open cartoon_256.pgm by Algorithm 4
90 104 255 145 298722
run time = 737.985ms
Open cartoon_256.pgm by Algorithm 5
90 104 254 145 297064
run time = 46.219ms

C:\Users\wjddb\Desktop\Algo1\x64\Release\Algo1.exe
```

- n=512,1024일 때 알고리즘별 실행결과

```
Open moon_512.pgm by Algorithm 3
163 159 299 356 3100306
run time = 88698.383ms
Open moon_512.pgm by Algorithm 4
163 159 299 356 3100306
run time = 6597.426ms
Open moon_512.pgm by Algorithm 5
163 159 299 356 3100306
run time = 294.663ms
Open moon_1024.pgm by Algorithm 3
326 318 596 714 12275784
run time = 1845646.125ms
Open moon_1024.pgm by Algorithm 4
326 318 596 714 12275784
run time = 55395.230ms
Open moon_1024.pgm by Algorithm 5
326 318 596 714 12275784
run time = 3373.677ms

C:\Users\wjddb\Desktop\Algo1\x64\Release\Algo1.exe
```

```
Open moon_512.pgm by Algorithm 3
163 159 299 356 3100306
run time = 96157.703ms
Open moon_512.pgm by Algorithm 4
163 159 299 356 3100306
run time = 7125.565ms
Open moon_512.pgm by Algorithm 5
163 159 299 356 3100306
run time = 328.612ms
Open moon_1024.pgm by Algorithm 3
326 318 596 714 12275784
run time = 4650857.500ms
Open moon_1024.pgm by Algorithm 4
326 318 596 714 12275784
run time = 57686.316ms
Open moon_1024.pgm by Algorithm 5
326 318 596 714 12275784
run time = 3123.455ms

C:\Users\wjddb\Desktop\Algo1\x64\Release\Algo1.exe
```

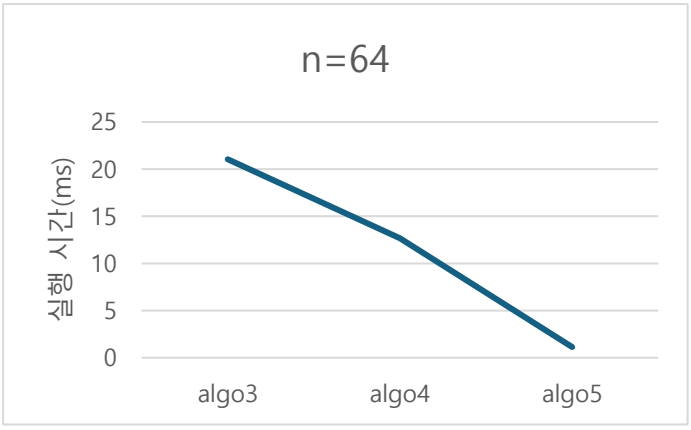
```
Open moon_512.pgm by Algorithm 3
163 159 299 356 3100306
run time = 171660.328ms
Open moon_512.pgm by Algorithm 4
163 159 299 356 3100306
run time = 11485.146ms
Open moon_512.pgm by Algorithm 5
163 159 299 356 3100306
run time = 586.424ms
Open moon_1024.pgm by Algorithm 3
326 318 596 714 12275784
run time = 2991933.500ms
Open moon_1024.pgm by Algorithm 4
326 318 596 714 12275784
run time = 97170.117ms
Open moon_1024.pgm by Algorithm 5
326 318 596 714 12275784
run time = 5633.922ms

C:\Users\wjddb\Desktop\Algo1\x64\Release\Algo1.exe
```

알고리즘이 달라져도 같은 파일에 대해서는 결과가 모두 같음을 확인했다.

n=2048일때는 개인 컴퓨터 속도 문제로 인해 12시간이 지나도 실행결과가 나오지 않아 생략했다.

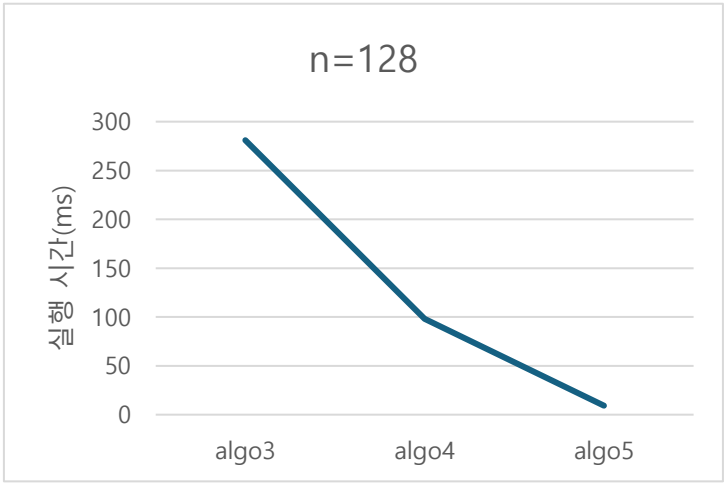
1. 해상도 n=64

(ms)	algo3	algo4	algo5
first	17.151	10.637	0.961
second	21.604	14.006	1.311
third	24.379	13.381	1.144
avg	21.04467	12.67467	1.138667



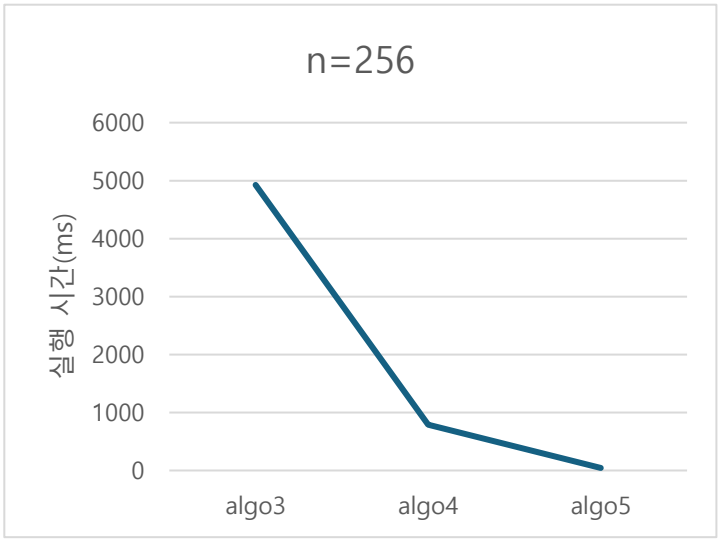
2. 해상도 n=128

(ms)	algo3	algo4	algo5
first	251.668	77.11	6.204
second	291.85	127.471	12.04
third	299.45	89.977	9.404
avg	280.9893	98.186	9.216



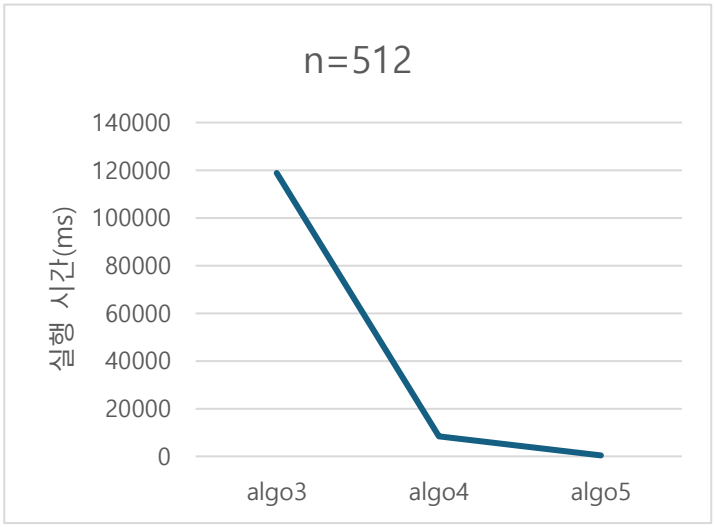
3. 해상도 n=256

(ms)	algo3	algo4	algo5
first	4471.901	778.001	39.89
second	5278.343	859.784	44.556
third	5026.804	737.985	46.219
avg	4925.683	791.9233	43.555



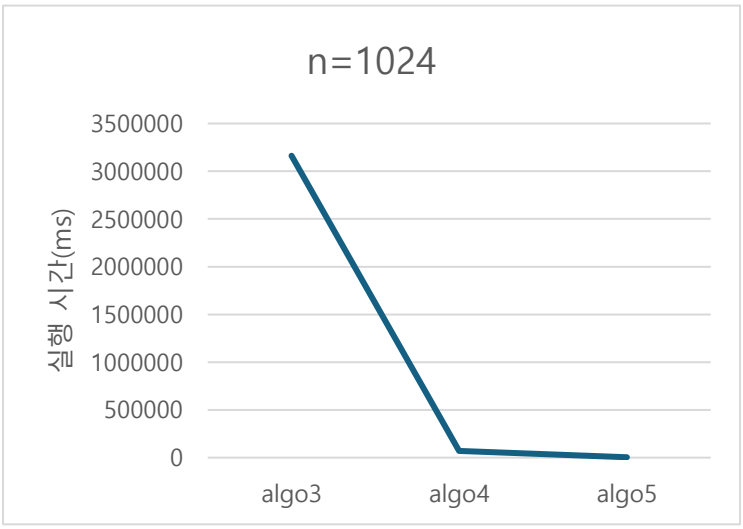
4. 해상도 n=512

(ms)	algo3	algo4	algo5
first	88698.38	6597.426	294.663
second	96157.7	7125.565	328.612
third	171660.3	11485.15	586.424
avg	118838.8	8402.712	403.233



5. 해상도 n=1024

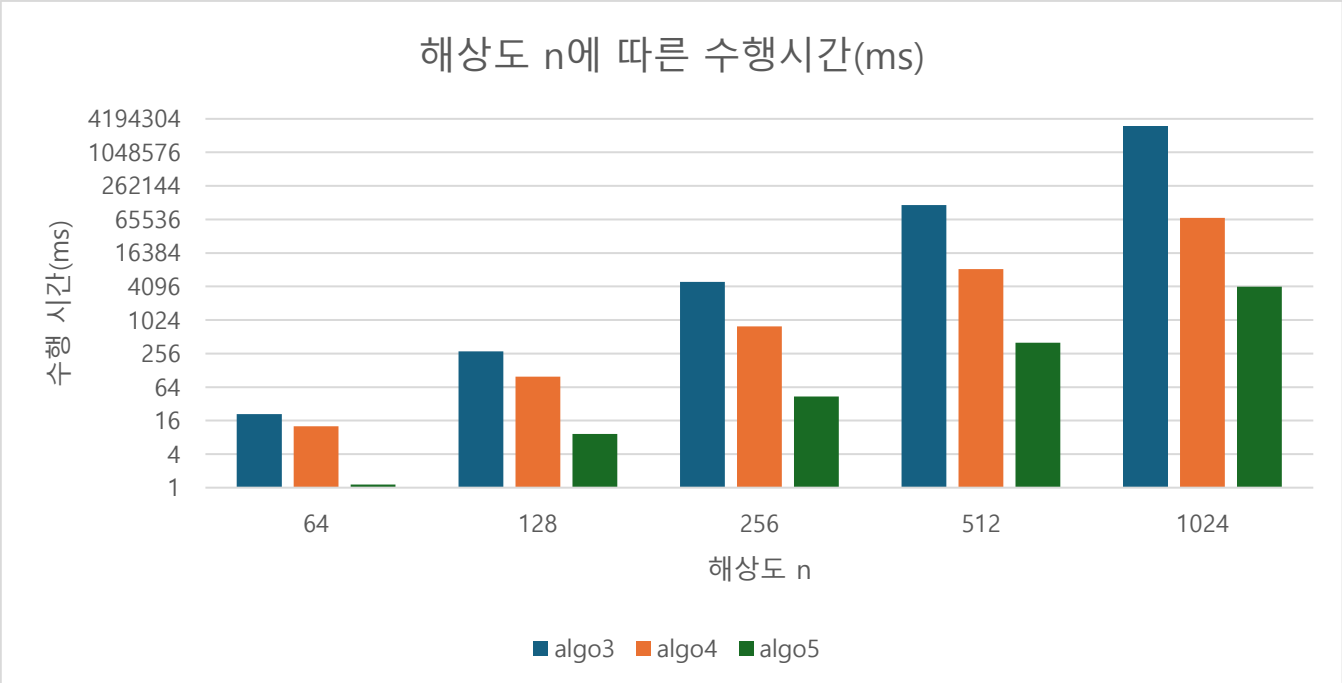
(ms)	algo3	algo4	algo5
first	1845646	55395.23	3373.677
second	4650858	57686.32	3123.455
third	2991934	97170.12	5633.922
avg	3162812	70083.89	4043.685



n이 2배 증가할 때 실행시간의 증가(몇 배)

	n=128/n=64	n=256/n=128	n=512/n=256	n=1024/n=512	평균 증가(배)
algo3	13.35204	17.52979	24.12636	26.61431	20.40562
algo4	7.746634	8.065542	10.61051	8.340627	8.690829
algo5	8.093677	4.72602	9.258019	10.02816	8.026469

n이 2배 증가할 때 수행시간이 algorithm3은 20.4배, algorithm4는 8.7배, algorithm5는 8배 커지는 것을 통해 이론적인 시간복잡도 O(n^4), O(n^3 log n), O(n^3)과 비례하는 것을 확인했다.

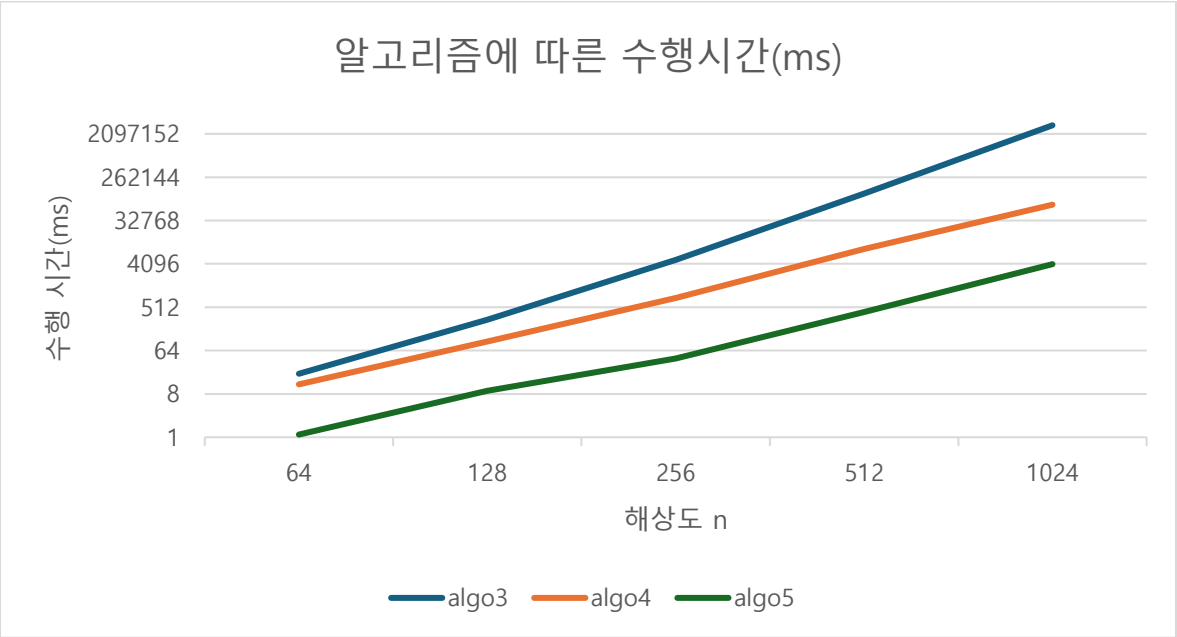


수행시간을 로그스케일로 지정했을 때 알고리즘 3은 n이 2배 증가할 때 실행시간이 약 4배 증가함을 보인다.

알고리즘 4는 n이 2배 증가할 때 실행시간이 3배보다 크게 증가함을 보인다.

알고리즘 5는 n이 2배 증가할 때 실행시간이 약 3배 증가함을 보인다.

전체적으로 n이 작을 때보다 n이 클 때에 알고리즘 간 실행 시간 차이가 현격해짐을 알 수 있다.



전체적으로 y축 실행시간이 로그 스케일일 때 세 그래프의 기울기가 선형적으로 일치한다. 이때 algorithm4는 n이 커질수록 조금 더 완만해지는 모습을 보인다. 여기에서 시간복잡도에 log가 포함됐을 때 차이를 알 수 있다. n이 작을 때는 비교적 o(n^4)인 algorithm3과 유사하다.

algorithm3은 n이 커질수록 유독 수행시간이 급격하게 증가하고, algorithm5는 둘과 비교했을 때 전반적으로 수행시간이 빠르다.

이번 비교를 통해 실제 수행시간과 시간복잡도가 비례함을 알 수 있었다.