**CSE3081-02 알고리즘 설계와 분석**

**[숙제 1] 보고서**

**20161663 허재성**

알고리즘 설명

이번 숙제에서는 크기가 N\* N인 정수형 2차원 배열 Square이 있을 때, 해당 2차원 배열에서 합이 가장 큰 부분직사각형인 Maximum Sum Subrectangle을 찾기 위한 3가지 알고리즘(Algorithm 3, 4, 5)을 구현했다.

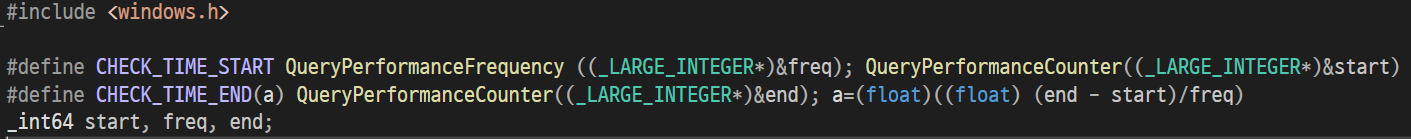
Algorithm 3은 크기가 N\* N인 2차원 배열 Square의 모든 Subrectangle의 부분합을 서로 비교하여 가장 합이 큰 Subrectangle을 찾는다. 각 Subrectange의 부분합을 O(1) 시간에 구하기 위해 Summed Area Table을 시간 복잡도 O(N2) 시간에 걸쳐서 만들고 이용한다. Square의 가능한 Subrectangle의 개수는 Big-O가 O(N4)이고 각 Subrectangle과 이전의 최대 부분합과 비교하는 비교 연산은 O(1) 시간이 걸리므로 비교하는데 걸리는 총 시간은 O(N4)이다. 따라서 Maximum Sum Subrectangle을 찾기 위한 총 시간의 시간복잡도는 O(N2 + N4) = O(N4)이다.

Algorithm 4와 Algorithm 5는 1차원 정수형 배열에서 Maximum Sum Subsequence을 찾는 알고리즘을 활용하여 Subrectangle을 찾는다. Algorithm 4와 Algorithm 5는 공통적으로 Square에서 행 길이(세로 길이)가 N인 모든 부분 직사각형을 고려하여 직사각형의 각 행(0~N-1)의 합을 길이가 N인 1차원 배열 temp의 인덱스(0~N-1)에 대응시켜 저장한다. temp를 완성 후, temp에서 Maximum Sum Subsequence를 찾아내어 Square에서 Maximum Sum Subrectangle을 찾는다. Square에서 행 길이가 N인 부분 직사각형의 개수는 O(N2)이다. 각

Algorithm 4에서는 Maximum Sum Subsequence를 찾는데 Divide-And-Conquer 방법을 사용하여 O(NlogN) 시간에 Maximum Sum Subsequence를 찾아낸다. 행 길이가 N인 부분 직사각형 O(N2)개에 대하여 각각 O(NlogN) 시간이 걸리므로 Algorithm 4에서 Maximum Sum Subrectangle을 찾는 시간복잡도는 O(N3logN)이다.

Algorithm 5에서는 Maximum Sum Subsequence를 찾는데 Dynamic Programming 방법을 사용하여 O(N) 시간에 Maximum Sum Subsequence를 찾아낸다. 행 길이가 N인 부분 직사각형 O(N2)개에 대하여 각각 O(N) 시간이 걸리므로 Algorithm 4에서 Maximum Sum Subrectangle을 찾는 시간복잡도는 O(N3)이다.

실험 방법

시간 측정의 경우 정밀한 시간 측정을 위하여 소스 파일에 아래와 같이<window.h> 헤더를 추가하고 매크로를 추가하였다.

사용하려는 알고리즘의 시작 이전에 CHECK\_TIME\_START를 호출하여 시간 측정을 시작하고, 알고리즘을 마치면 CHECK\_TIME\_END를 호출하여 알고리즘의 수행 시간 측정을 마친다. 위와 같은 방법으로 <time.h> 헤더를 추가하여 시간을 측정하는 것보다 더 정밀한 시간을 측정할 수 있다.

알고리즘의 시간 측정 결과를 ms(millisecond) 단위로 출력하여 알고리즘 사이의 시간 비교 시 비교하기 쉽게 하였다. 시간이 작게 나와서 지나치게 작은 소수를 관찰해야 할 경우, ms 단위를 사용하는 것이 낫다.

Algorithm 3, Algorithm 4, Algorithm 5의 수행 시간을 측정하기 위해서 입력 데이터로 다양한 N\*N 크기의 2차원 배열을 만들어 사용하였다. 2차원 배열의 원소의 음수 발생 확률이 낮을 경우, N이 커질수록 결과로 찾아낸 Subrectangle이 커져서 Subrectangle이 전체 Square와 비슷하거나 같은 결과가 많이 나오므로 다양한 Subrectangle의 결과를 확인하기 위하여 배열에 음수가 충분히 있을 수 있게 음수 발생 확률을 높게 하였다.

N의 크기를 결정할 때에는 기본적으로 100으로 시작해서 2배씩 증가시켜 N=100, 200, 400, 800, 1600, 3200, 6400일 경우를 관찰한다. 이론적으로 시간복잡도만을 고려했을 때, 수행 시간은 N이 2배로 증가하면 Algorithm 3의 경우 24인 16배, Algorithm 4의 경우 (23log2N)/(logN)(로그의 밑은 2로, 어떤 것이든 시간복잡도에 관계 없다.) = 8log2N/logN = 8(1 + log2/logN)배, Algorithm 5의 경우 23배인 8배로 증가한다. 실제 수행 시간의 비교를 더 명확히 하기 위해 N의 크기가 2배 관계인 경우를 위주로 관찰한다.

각 크기에 대하여 한 가지 데이터로 5번 실험을 하여 시간을 각각 측정 후 평균을 내어 신빙성이 높은 시간을 채택한다. 또한 한 가지 데이터가 아니라 5가지 데이터로 각각 시간 측정 후 평균을 내어 특정 크기 N에 대하여 믿을 만한 시간 측정 결과를 찾아낸다.

실험한 컴퓨터의 환경

OS : Windows 10

CPU : Intel® Core(TM) i3-5005U CPU @ 2.00GHz

RAM : 8.00GB

Compiler : Visual Studio 19 Release Mode/x64 Platform

실험 결과

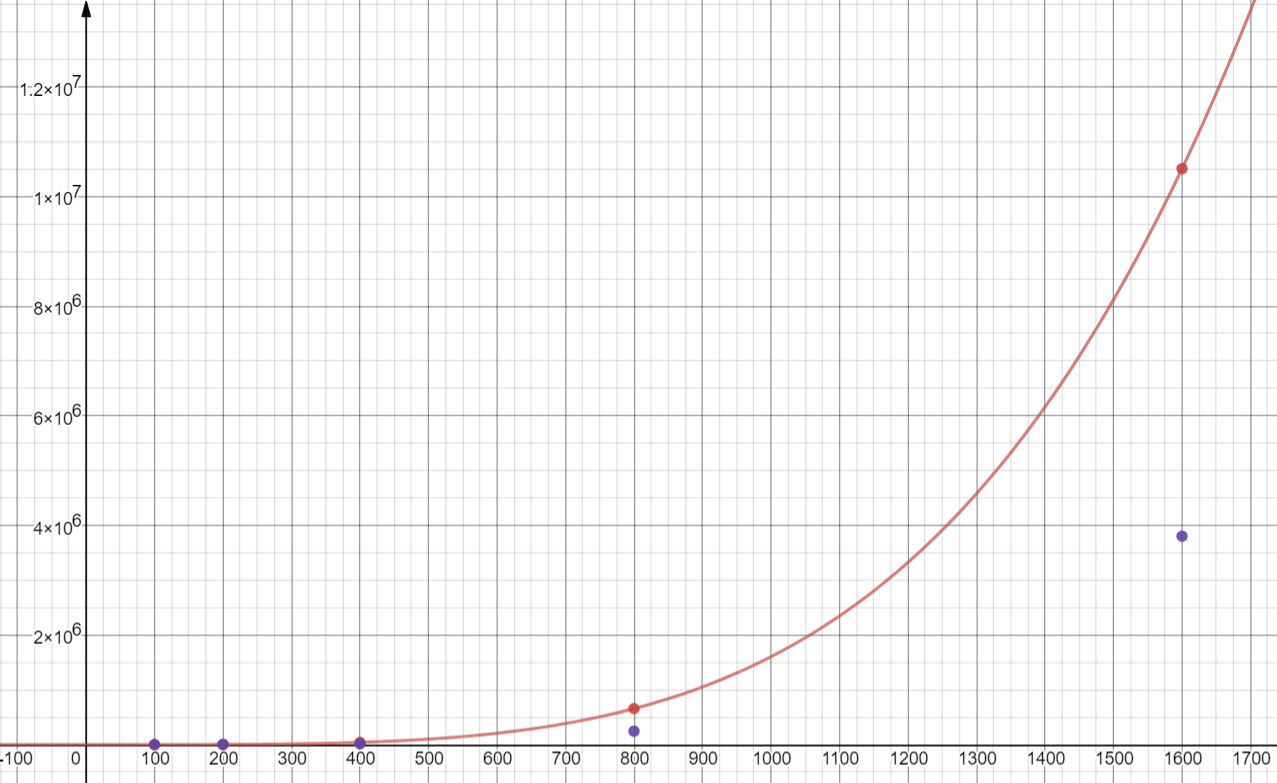
각 알고리즘에서 N이 2배로 증가할 경우 수행 시간의 증가를 비교한 표이다. 알고리즘 별로 같은 크기의 데이터에 대한 수행 시간을 좀 더 일반적으로 알아보기 위해 같은 크기의 다른 데이터 5개의 수행 시간을 구한 후 평균을 내었다.

Algorithm 3의 경우, 시간 측정의 한계 상 N=100부터 2배씩 증가시켜 1600일 때 까지 측정하였다.

Algorithm 3 ( O(N4) Algorithm )

|  |  |  |
| --- | --- | --- |
| N | 5개 데이터 평균 시간(ms) | 이전 시간 대비 시간 증가율(소수 첫째자리까지) |
| 100 | 160.32516 |  |
| 200 | 1,407.832 | 8.8 |
| 400 | 14,729.08 | 10.5 |
| 800 | 243,991.4 | 16.6 |
| 1600 | 3,802,442 | 15.6 |

Algorithm 3의 경우, 시간복잡도가 O(N4)이므로 시간 복잡도만 고려했을 때 N이 2배 증가하면 수행 시간이 (2N)4/N4 = 16배 증가할 것으로 예상할 수 있다.



위의 그래프에서 빨간색 실선으로 그려진 그래프는 시간 복잡도를 고려하여 그린 이론적인 증가율인 T=CN4이고(C는 상수) 그 위의 빨간 점은 N=100, 200, 400, 800, 1600일 때의 이론적인 시간을 나타낸다. 실선 그래프 아래의 보라색 점은 실제 N=100, 200, 400, 800, 1600일 때의 실제 시간을 나타낸다. 실제로는 시간 복잡도처럼 정확히 CN4의 비율로 증가하지는 않는 것을 확인할 수 있다.

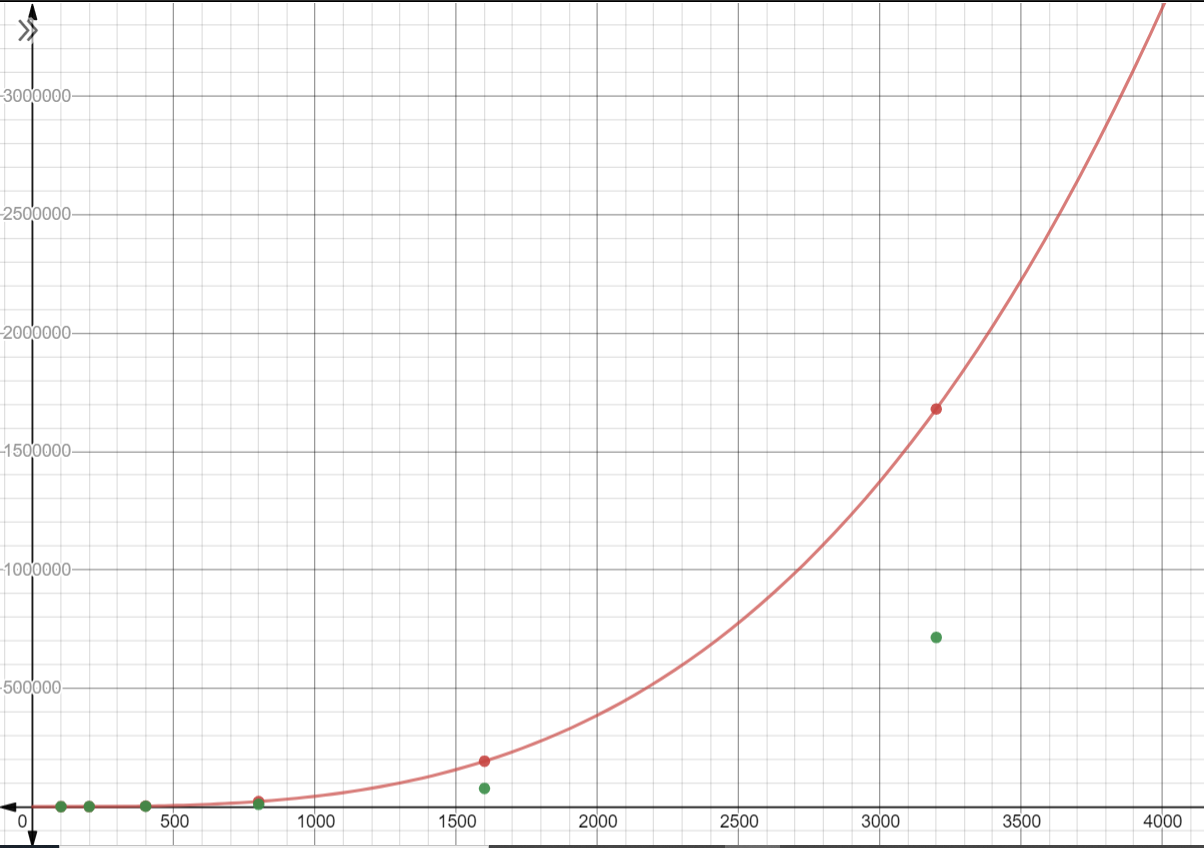
실제 알고리즘의 수행 횟수가 정확히 CN4이 아니라 차수가 N3 이하의 항과 상수항도 있기 때문에 정확히 CN4의 그래프와 일치하지는 않는다. 실험 데이터가 더 충분하여 더 큰 N으로 실험해보면 결국 CN4의 그래프와 유사해짐을 예측할 수 있다.

Algorithm 4의 경우 N=100부터 2배씩 증가시켜 3200일 때까지 측정하였다.

Algorithm 4 ( O(N3logN) Algorithm )

|  |  |  |
| --- | --- | --- |
| N | 5개 데이터 평균 시간(ms) | 이전 시간 대비 시간 증가율(소수 첫째자리까지) |
| 100 | 29.2462 |  |
| 200 | 232.7198 | 8.0 |
| 400 | 1,552.642 | 6.7 |
| 800 | 9,570.796 | 6.2 |
| 1600 | 76,822.82 | 8.0 |
| 3200 | 714,777.2 | 9.3 |

Algorithm 4의 경우, 시간복잡도가 O(N3logN)이므로 시간 복잡도만 고려했을 때 N이 2배 증가하면 수행 시간이(2N)3log(2N)/N3logN = 8(1+log2/logN)배 증가할 것으로 예상할 수 있다. Algorithm 3과 달리 N이 2배로 증가할 때, N의 값에 관계없이 이론적인 증가비율이 정해진 것이 아니라 N에 따라 증가비율이 달라진다. N이 커질수록 log2/logN이 0에 수렴하므로 증가비율이 8에 수렴할 것으로 예상할 수 있다.



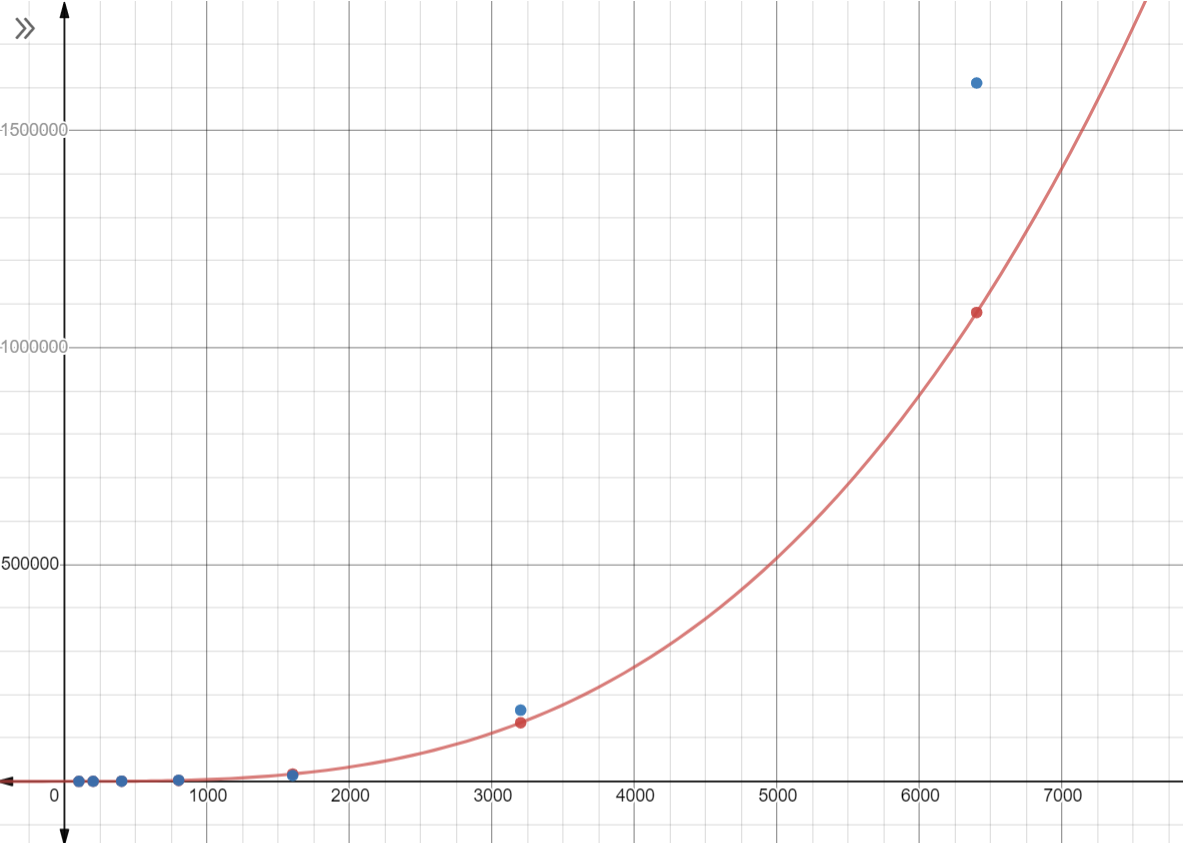
위의 그래프에서 빨간색 실선으로 그려진 그래프는 시간 복잡도를 고려하여 그린 이론적인 증가율인 T=CN3log2N이고(C는 상수) 그 위의 빨간 점은 N=100, 200, 400, 800, 1600, 3200일 때의 이론적인 시간을 나타낸다. 실선 그래프 아래의 초록색 점은 실제 N=100, 200, 400, 800, 1600, 3200일 때의 실제 시간을 나타낸다. 실제로는 시간 복잡도처럼 정확히 CN3log­2N의 비율로 증가하지는 않는 것을 확인할 수 있다.

실제 알고리즘의 수행 횟수가 정확히 CN3log2N이 아니라 다른 항도 있기 때문에 정확히 CN3log2N의 그래프와 일치하지는 않는다. N이 커질수록 N3log2N가 나머지 항을 무시할 수 있을 정도로 N3log2N의 형태를 따를 것을 예상할 수는 있다.

Algorithm 5의 경우 N=100부터 2배씩 증가시켜 6400일 때까지 측정하였다

Algorithm 5 ( O(N3) Algorithm )

|  |  |  |
| --- | --- | --- |
| N | 5개 데이터 평균 시간(ms) | 이전 시간 대비 시간 증가율(소수 첫째자리까지) |
| 100 | 4.12366 |  |
| 200 | 43.45316 | 10.5 |
| 400 | 318.7044 | 7.3 |
| 800 | 2,507.184 | 7.9 |
| 1600 | 13,387.82 | 5.3 |
| 3200 | 164,049.2 | 12.3 |
| 6400 | 1,610,036 | 9.8 |

Algorithm 5의 경우, 시간복잡도가 O(N3)이므로 시간 복잡도만 고려했을 때 N이 2배 증가하면 수행 시간이(2N)3/N3 = 8배 증가할 것으로 예상할 수 있다. 

위의 그래프에서 빨간색 실선으로 그려진 그래프는 시간 복잡도를 고려하여 그린 이론적인 증가율인 T=CN3이고(C는 상수) 그 위의 빨간 점은 N=100, 200, 400, 800, 1600, 3200, 6400일 때의 이론적인 시간을 나타낸다. 실선 그래프 아래의 파란색 점은 실제 N=100, 200, 400, 800, 1600, 3200, 6400일 때의 실제 시간을 나타낸다. 실제로는 시간 복잡도처럼 정확히 CN3의 비율로 증가하지는 않고 N이 클 때 오히려 이론적인 시간보다 더 많은 시간이 걸림을 알 수 있다. 이론과 달리 N3 이외의 항을 완전히 무시할 수 없기 때문이다. 하지만 N이 충분히 크다면 CN3에 수렴하는 것을 예상할 수 있다.

Algorithm 3, 4, 5 사이의 시간을 비교해보기 위해 N=800일 때, 같은 데이터에 대해서 각 알고리즘 별로 5번 시간을 측정하여 평균을 낸 후, 시간을 비교해보겠다.

|  |  |  |
| --- | --- | --- |
| N=800 | 5번 수행 평균 시간(ms) | 알고리즘 사이의 시간 비 |
| Algorithm 5 (O(N3)) | 2,350.824 |  |
| Algorithm 4 (O(N3log2N)) | 9,756.638 | 4.2 |
| Algorithm 3 (O(N4)) | 223,384.8 | 22.9 |

시간복잡도를 고려했을 때 이론적으로 임의의 N에 대하여 Algorithm 4는 Algorithm 5보다 log­2N배 더 많은 시간이 소요된다. log2800이 약 9.6이므로 이론상으로는 Algorithm 4는 Algorithm 5보다 9.6배 느려야 하지만 실제로는 4.2배 느리다. 각 알고리즘 연산 횟수의 최고차항(N3vs N3log2N) 외에도 다른 항들이 실제로 영향을 미치기 때문이다.

마찬가지로 임의의 N에 대해서 Algorithm 3은 Algorithm 4보다 이론적으로 N/log2N 배 느리다. N이 800일 경우 이론상으로 약 83배 느리다. 하지만 실제로는 23배 정도 느리다. 실제로 두 알고리즘의 최고차항 이외의 항도 고려해야 하기 때문에 이론과 차이는 있다. 하지만 N이 커질수록 N/log2N도 증가하므로(N/log2N의 도함수가 양수이다.) N이 커질수록 Algorithm 3은 Algorithm 4보다 수행이 크게 증가하는 것을 알 수 있다. Algorithm 3에서 Algorithm 4로 개선하는 것이 시간 복잡도를 크게 향상시키는 것을 알 수 있다.