전공: 컴퓨터공학 학년: 2 학년 학번: 20171653 이름: 신나현

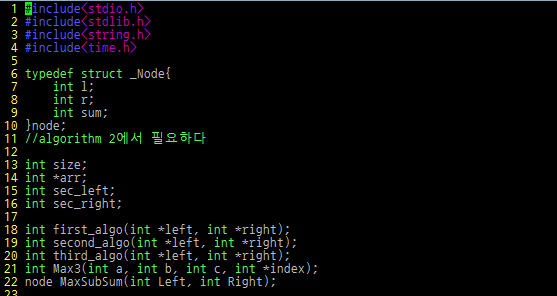
1. 과제 목적

The goal of this MP is to understand how different algorithms perform differently while producing the same result for the same problem.

1. 개발환경

Linux machine.

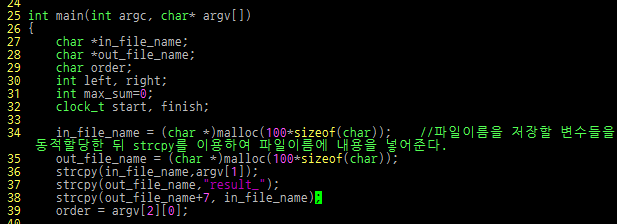
1. 제작



필요한 헤더파일을 include해주고, 사용할 struct를 typedef해준다.

사용할 전역변수들과 함수들을 선언해 준다.

<main함수>

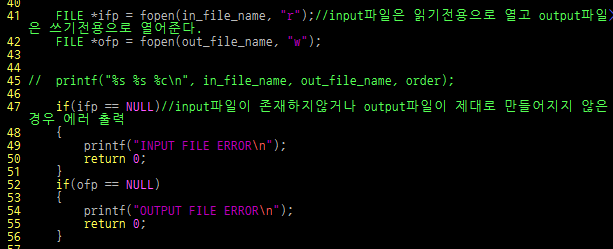


$ ./mss20171653 input00001.txt 2 으로 실행하면

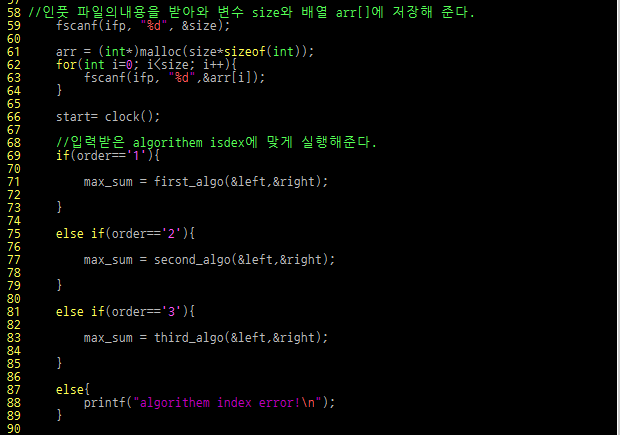
main(int argc, char\* argv[])를 이용하여 값을 입력 받게 되는데, 이 때 argv[1]에는 “input00001.txt”가 저장되고, argv[2][0]에는 2가 저장된다.

main함수에서 필요한 변수들을 선언해준다.

in\_file\_name과 out\_file\_name을 malloc을 이용해 동적할당 한 뒤, strcpy를 이용해 각각의 변수에 내용을 저장해 준다. order에는 argv[2][0]의 값을 넣어준다.



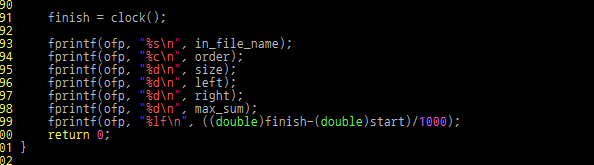
input파일은 읽기모드로, output파일은 쓰기모드로 fopen을 이용해 열어준다. 이 때 파일을 여는 과정에서 에러가 발생하는 경우, 에러를 출력해 준다.



fscanf로 파일 안의 내용을 받아와 size에 값을 저장하고, malloc을 이용해 arr을 size\*sizeof(int)의 크기로 동적할당 해준다.

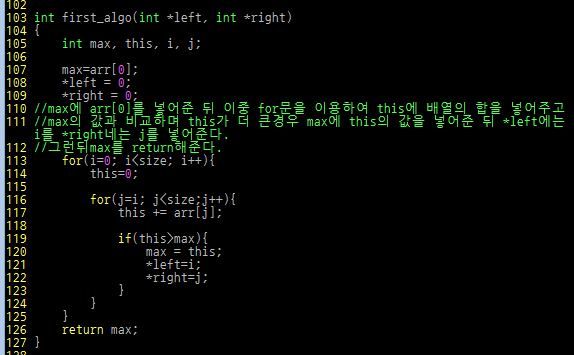
그런 뒤, for문을 이용해 arr에 파일의 내용을 저장해 준다.

시간복잡도를 측정하기 위해 start에 clock()을 이용해 시작시간을 넣어주고, order에 맞는 알고리즘을 실행한다.



그런 다음 finish에 clock()을 이용해 종료 시간을 넣어주고, 문제에서 요구한대로 output파일에 fprintf를 이용해 출력해 준다.

<int first\_algo(int \*left, int\* right) 함수>



필요한 변수들을 선언해 준 뒤 max에는 arr[0]의 값을 넣어준다.

\*left, \*right는 초기값을 0으로 설정해준다.

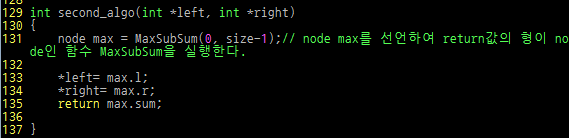
이중 for문을 이용하는데 이 때 첫 번째 for문안에서 this를 0으로 설정해 준다. 그 안의 두 번째 for문에서 this에 j=i부터 size보다 작을 동안 arr[j]의 값을 더해준다.

이 때, this가 max보다 커지는 경우, max에는 this값을 넣어주고, \*left에는 i를, \*right에는 j를 넣어준다.

for문이 끝나면 max값을 return해 준다.

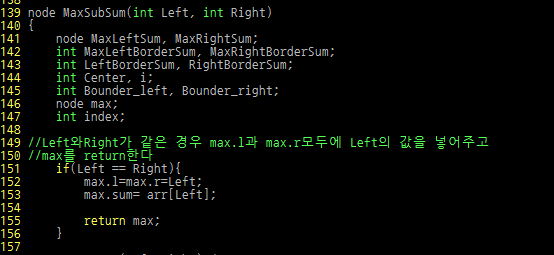
size보다 작을동안의 이중 for문을 반복하기 때문에 O(n^2)인 알고리즘이다.

<int second\_algo(int \*left, int \*right) 함수>



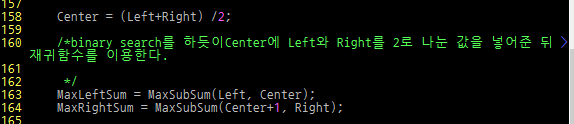
이 함수에서 MaxSubSum함수를 실행해준다, 그런뒤, \*lef와 \*right에 해당하는 값을 넣어주고max.sum을 return 해준다.

<node MaxSubSum(int Left, int Right) 함수>



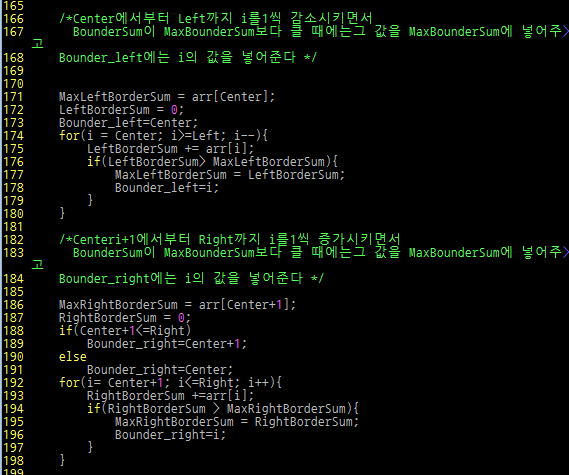
필요한 변수들을 선언해준다.

Left와 Right가 같은 경우, max.l=max.r=Left를 넣어주고, max.sum에는 arr[Left]의 값을 넣어준 뒤 max를 return해준다.



Center에는 Left와 Right를 더한 값을 2로 나눈 몫을 넣어준다.

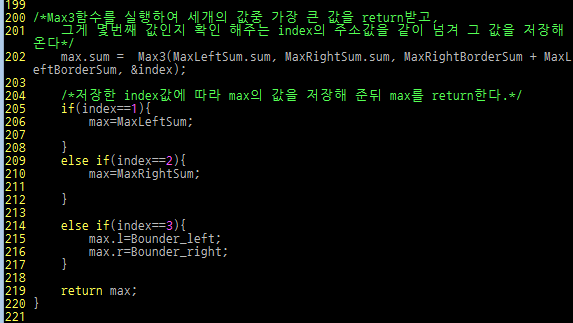
MaxLeftSum은 Left와 Center값을 이용해 재귀함수를 호출하여 값을 받아오고, MaxRightSum은 Center+1과 Right를 이용해 재귀함수를 호출하여 값을 받아온다.

.

MaxLeftBorderSum을 구할 때에는 중앙에서부터 왼쪽으로 가면서 중앙부터 어디까지 더한 값이 최대가 되는지 찾아 MaxLeftBounderSum에 넣어주고, Bounder\_left에 그 때의 i값을 넣어준다.

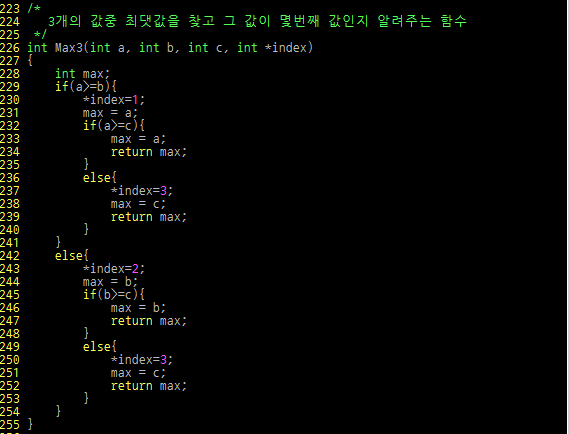
MaxRightBounderSum을 구할 때에는 반대로 중앙+1에서부터 오른쪽으로 가면서 어디까지 더한 값이 최대가 되는지 찾아 MaxRightBounderSum에 넣어주고, Bounder\_right에 그 때의 i값을 넣어준다.

이 함수에서는 재귀함수를 이용하여 Binary Search를 하는 방식으로 최댓값을 찾고 있으므로 시간복잡도는 O(nlogn)이다.



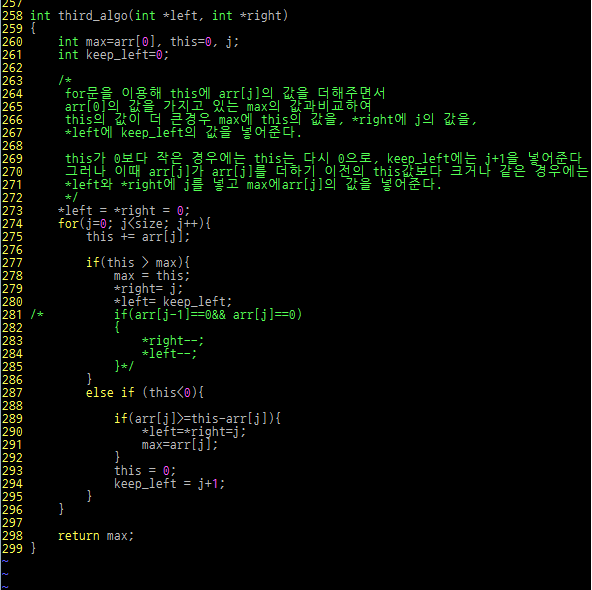
Max3함수를 실행하여 3개의 값들 중 가장 큰 값과, 그 값이 3개중 몇 번째 값인지를 받아와 그에 맞는 결과값을 node max에 저장해 준 뒤, max를 return해준다.

<int Max3(int a, int b, int c, int \*index) 함수>



가장 큰 값을 찾아 return해주고 그에 맞는 값을 \*index에 저장해준다.

<third\_algo(int \*left, int \*right) 함수>



함수 내에서 필요한 변수들을 선언해 준다.

max의 초기값은 arr[0] 으로, \*left와 \*right의 초기값은 0으로 설정해 준다.

for문을 j=0부터 j<size인 동안 돌려 this에 arr[j]값을 더해준다. 이 과정에서 this가 max보다 큰 경우 max에 this값을 넣어주고, \*left에는 keep\_left를, \*right에는 j값을 넣어준다.

그렇지 않을 때 this<0인 경우에 만약 arr[j]가 this – arr[j]보다 크거나 같을 때에는 \*left와 \*right에 j의 값을 넣어준다. 그 외에는 this=0, keep\_left는 j+1을 넣어준다.

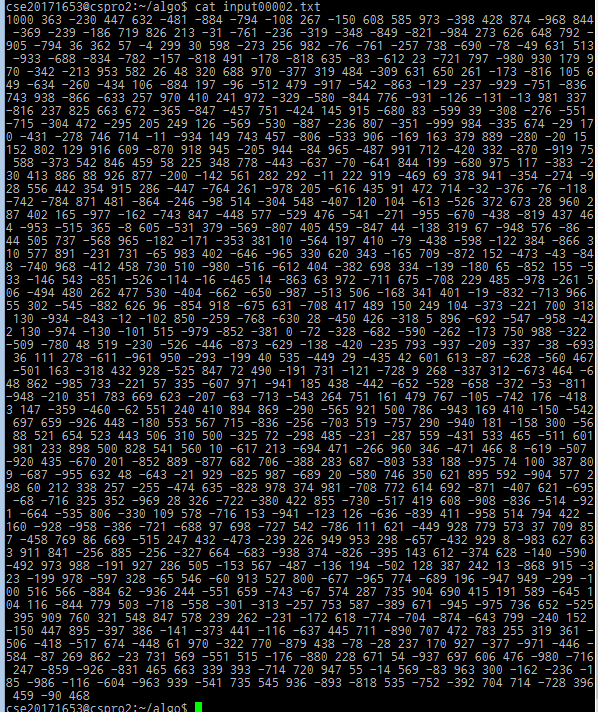
이 알고리즘에서는 size보다 작은 동안 하나의 for문을 돌기 때문에 시간복잡도는 O(n)이다.

1. 시험

Input00001.txt



Input00002.txt

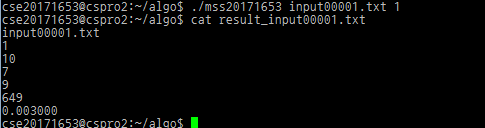


Input00003.txt

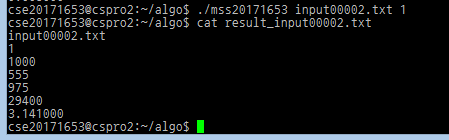


<Algorithm 1>

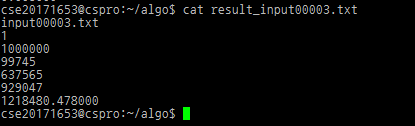
input00001.txt



input00002.txt

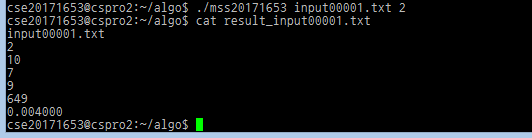


input00003.txt

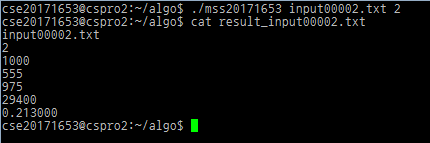


<Algorithm 2>

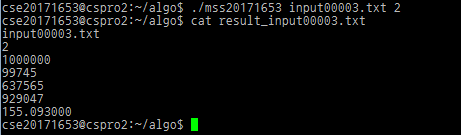
input00001.txt



input00002.txt

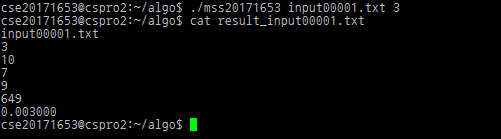


input00003.txt

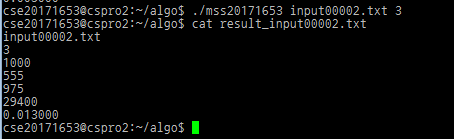


<Algorithm 3>

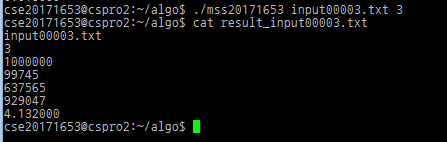
input00001.txt



input00002.txt



input00003.txt



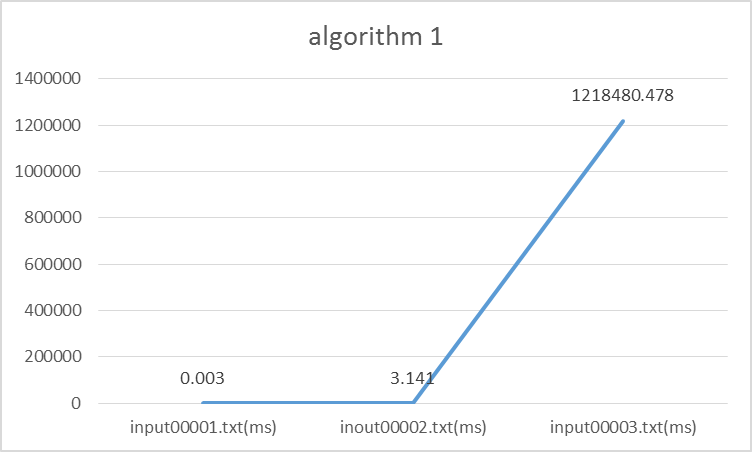
The hardware specification of the machine



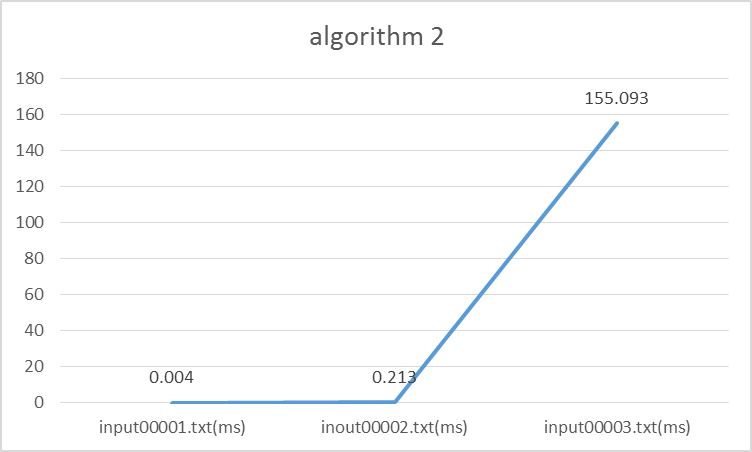
1. 시간 복잡도



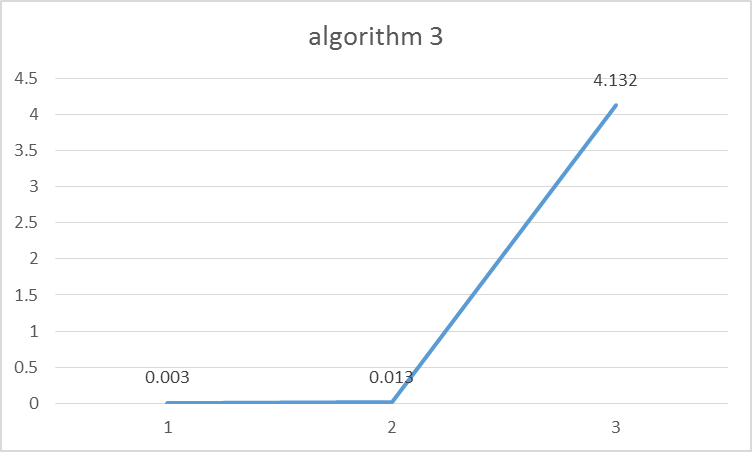
Algorithm 1



Algorithm2



Algorithm3



알고리즘 1에서는 input의 크기가 커질수록 걸리는 시간이 급격하게 증가하였다.

심지어 input00003.txt를 넣어 주었을 때에는 약 20분정도가 걸렸다.

위의 그래프를 통해 알고리즘1>알고리즘2>알고리즘3의 순서로 시간이 오래 걸린다는 것을 알 수 있다. 알고리즘 1의 시간복잡도는 O(n^2), 알고리즘2의 시간복잡도는 O(nlogn), 알고리즘3의 시간복잡도는 O(n)이다.

이번 과제를 통해서 같은 결과값을 찾는 프로그램을 짤 때에 어떤 알고리즘을 선택하느냐에 따라서, 매우 효율적으로 문제를 해결할 수 도 있고, 매우 비효율적으로 문제를 해결할 수 도 있다는 것을 깨달았다.

앞으로는 더 효율적인 알고리즘을 이용해 프로그래밍할 수 있도록 더 노력해야겠다.