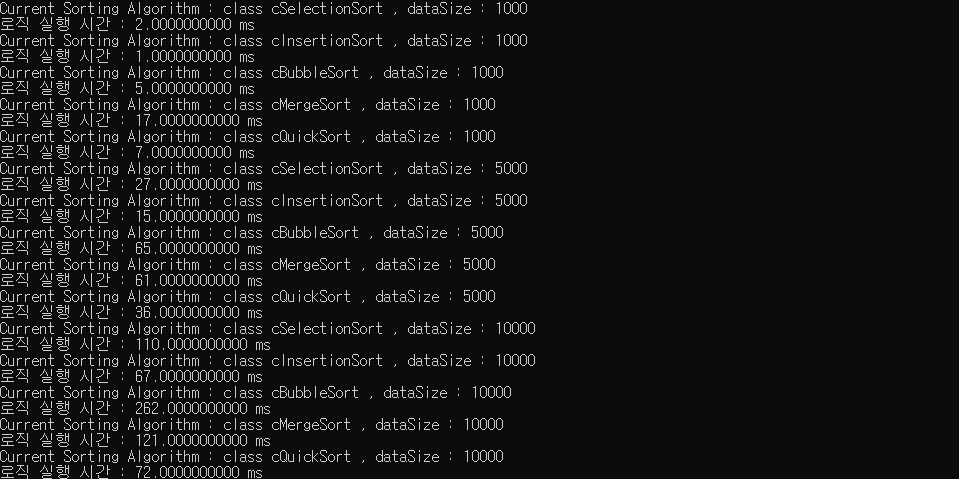
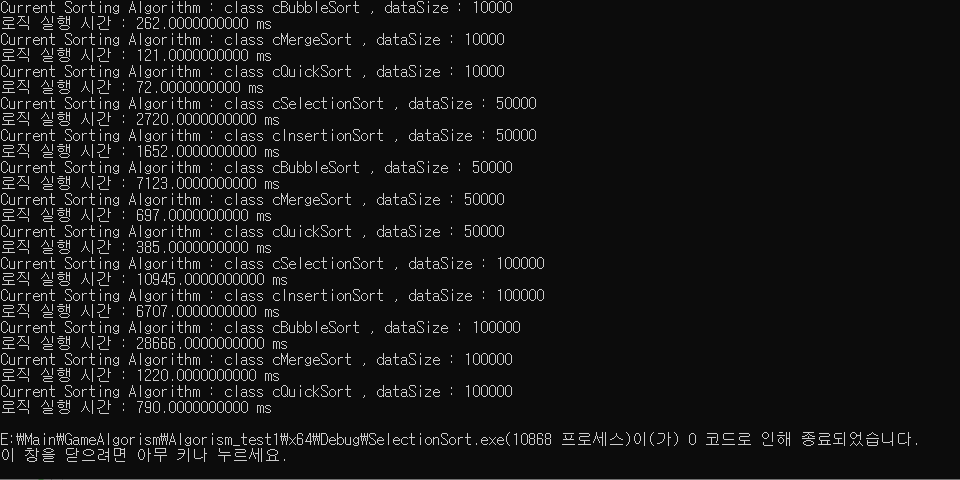
알고리즘 효율성 분석

201613105 박건욱

1. 실험환경
   1. 컴퓨터 사양 : AMD Ryzen 7 2700x Eight-Core Processor 3.70 GHz, 16.0GB DDR4 Ram
   2. 컴파일러 버전 : Visual Studio 2017
2. 테스트에 적용된 데이터 개수
   1. 1000
   2. 5000
   3. 10000
   4. 50000
   5. 100000
3. 실험방법
   1. Sorting 직전에 clock 함수로 현재시간을 구한다.
   2. Sorting 함수를 호출시킨다.
   3. Sorting이 끝난 직후 clock 함수로 현재시간을 불러온 후 A 과정에서 구한 시간을 빼준다
   4. C과정에서 도출된 연산 실행 시간을 출력해준다.
   5. Sorting 된 데이터가 유효한지 데이터 검사 함수를 실행해준다.
   6. Sorting 이 되지않은 데이터를 확인 시 에러 메세지를 출럭해준다.
4. 실험결과





데이터가 적을땐 상대적으로 insertionSort, selectionSort 가 실행속도가 빠르지만 데이터 수가 많아질수록 Big O 식에 의해 mergeSort, quickSort의 실행속도가 빠르게 측정된다.

1. 결론 및 느낀점

실제의 작업환경에서 sort를 사용할 땐 데이터의 개수를 파악하고 데이터의 정렬성 도

같이 예측하고 상황에 맞는 sorting 방법을 채택해야 할 것 같다.

대용량의 데이터 처리로 인해 재귀함수에서 나올 수 있는 callstack overflow 현상을 예방하기 위해 재귀함수를 사용하지 않고 stack 자료구조를 이용한 재귀적 루프문으로 mergeSort, quickSort를 설계하였다. 또한 초기 자료형으로 최대 3억개까지 데이터를 절렬시켜보고자 시도했지만 기하급수적으로 올라가는 실행속도 때문에 결국 case 5 10000개로 결정했다.

느낀점으로는 알고리즘을 작성할 때 로직 완성 후 로직의 정확성만 테스트 하고 필요한 경우에는 프로그램 프로파일링을 통한 cpu 부하정도만 간단히 체크하였는데, 향상된 알고리즘 실력을 가지기 위해서는 귀찮더라도 시간을 좀더 투자하여 극한의 상황에서의 정확성과 알고리즘 실행속도를 측정해야 한다는 사실을 다시 생각해보는 계기가 되었다.