정렬 과제 – 제출방식

- **4명까지 한 팀을 이루어 과제 수행**
  - 분반간 팀 구성 가능, 재수강생은 재수강생끼리 팀구성, 1~3인팀 허용
    - 예외적인 경우 허용할 수 있으니 교수에게 문의할 것
    - 강의지원시스템에 “팀원구하기 게시판 " 만들어 두었으니 팀원 구하기에 활용
      - 글머리로 구해요/완료 선택 가능함. 분반간 게시판 공유됨.
  - pdf 파일 1개와 폴더 2개 압축하여 제출
    - 문서 파일에는 문제풀이와 (정렬 프로그램) 결과 분석을 포함
      - 첫 페이지(표지)에 과제에 성실히 참가한 팀원 이름만 올릴 것
      - 적절한 그래프와 표를 이용하여, 정렬 알고리즘을 비교할 수 있게 보고서를 작성할 것
    - 폴더에는 정렬 프로그램 code를 포함
      - source code만 포함할 것(.java. 필요하다면 README.txt)
        - 정렬 대상을 파일에서 읽는 방식으로 구현하지 말 것
      - (가능한 주석 없이도) 각 클래스가 어느 정렬에 해당하는지 알아보기 쉽게 클래스 이름과 메소드 이름을 구성할 것
      - 각 정렬을 각기 main에서 구동하면 안 됨. 하나의 main에서 각 입력 유형과 크기에 따라 각 정렬 방식을 구동하고, 각 수행시간을 알아보기 쉽게 출력하도록 구성할 것

- **제출 방법**
  - 과제 결과물은 6/5(토) 23:00까지 팀대표자 1인이 upload

정렬 과제 – stable sort 분석

- **stable sort란 동일한 값이 입력 배열에 들어가 있을 때, 정렬 결과에서 동일한 값이 원 배열의 순서대로 저장되는 정렬 방식**
  - insertion, merge, radix sort는 stable함.
  - selection, shell, quick, heap sort는 non-stable함.
- **stable한 sort는 어떻게 stable한 결과를 보일 수 있는지 알고리즘을 기반으로 대략적으로 설명하시오.**
- **non-stable한 sort는 stable하지 않은 예제를 보여 stable 하지 않음을 보이시오.**

정렬 과제 – 비교 정렬 알고리즘 비교

- **아래 각 정렬 방식을**
  - 배열을 이용한 일반 insertion sort, binarySearch를 적용한 insertionSort
  - shellSort(shell size는 각 팀에서 결정)
  - 처음 값이 pivot인 recursiveQuickSort, median of three가 pivot인 recursive quick sort
  - recursiveMergeSort, iterativeMergeSort, naturalMergeSort
  - heapSort, bubbleSort, selectionSort
  - java.util.Arrays의 Arrays.sort와 java.util.Collections의 Collections.sort
- **int, double, string, class Student 의 자료형을 가지는**
  - Student는 100점 만점 기준의 5개 교과목의 성적을 포함하는 클래스
    - Comparable으로 구현하며 교과목 평균이 정렬 key
    - 평균 저장하지 말고, compare()나 compareTo()가 getAvg()를 통해 매번 평균값을 구하게 할 것
- **random values / increase order values / decrease order values 각 배열에 대해서**
- **충분히 큰 n의 5개 이상의 값에 대해서**
- **적용하여 정렬을 수행한 경우 수행 시간의 성능을 비교하시오.**
  - 각 알고리즘에서 각 자료형과 random/increasing/decreasing, 각 입력의 크기 n에 대해서 비교
  - 각기 다른 알고리즘 간에 대해서도 비교
  - 속도 측정의 기준(m-sec, μ-sec, clock rate)은 각 팀에서 알아서 결정
    - 데이터 생성 시간은 빼고 측정. 정렬 결과 맞는지 확인하고, 출력 제외하고 시간 측정 할 것.

정렬 과제 – 비교 v.s 비비교 정렬 비교

- **comparison based v.s. non-comparison based**

  & arithmetic operation v.s. logical operation

  - 0~2^{16}-1 범위의 충분히 큰 random unsigned int 배열을 생성하고
    - 다양한 n과 다양한 자료형, 정렬형태에 대해서는 시행하지 않아도 됨

  - 아래 각 방식에 의한 정렬의 속도를 비교할 것
    - 1) modular 연산(%0x10) 으로 16진수 LSB 값을 얻어 정렬하는 radix sort
    - 2) masking(&)과 shift 연산(>>)연산으로 16진수 LSB값을 얻어 정렬하는 radix sort
    - 3) modular 연산(%10) 으로 10진수 LSB 값을 얻어 정렬하는 radix sort
    - 4) 팀에서 만든 sort 중 random int에 대해서 제일 좋은 효율을 낸 것

    - 단, radix sort에서는 count sort 로 각 radix내에서의 정렬을 구현할 것