1. 목표

여러가지 정렬 알고리즘을 구현하고 성능을 비교하면서 알고리즘의 중요성을 깨닫도록 합니다.

2. 문제

아래와 같은 여러 가지 정렬 알고리즘을 구현하고 비교해 봅니다.

- [B] Bubble Sort
- [I] Insertion Sort
- [H] Heap Sort
- [M] Merge Sort
- [Q] Quick Sort
- [R] Radix Sort

추가로 가장 빠른 정렬 방법을 찾아주는 알고리즘을 구현합니다.

- [S] Search

Search 를 구현할 때 모든 정렬 알고리즘을 일일이 수행 후 그중 가장 빠른 방법을 선택하게 되면 시간 복잡도는 O(n^2)로 비효율적입니다. 입력 data 만을 보고 가장 빠른 정렬 방법을 O(n)에 찾아주는 알고리즘을 구현합니다.

아이디어:

1. 입력의 최대 자릿수가 k 이하면 radix sort 사용.
2. Hash table 에 입력을 넣어서 충돌이 일어나는 빈도로 중복을 근사. 충돌 발생률이 일정 비율 k 이상이면 중복이 많다고 판단하여 OOO sort 사용.
3. 입력이 이미 정렬이 얼마나 되어 있는지 모든 i 와 i+1 쌍을 통해 근사. 정렬 비율이 k 이상이면 OOO sort 사용.

위 아이디어들은 필수로 구현해 보도록 합니다. 또한 데이터의 특성을 잘 고려해서 더 최적화할 수 있는 조건이 있으면 추가로 구현해 보도록 합니다. 본인의 정렬 알고리즘 구현에 따라서 각 경우에 대한 최적의 정렬 방법이 다를 수 있습니다. 또한 k 는 실험을 통해 설정해야 되는 하이퍼파라미터입니다.

Search 입력의 모든 숫자는 정수이며 int 의 범위 내에 속합니다. 숫자들의 총개수는 50000 개 이하입니다.

결국 모든 정렬 방법을 한 번씩 돌리는 것보다 Search 와 이로부터 찾은 정렬 알고리즘을 한 번만 돌리는 시간이 더 빠르게 구현하는 것이 주된 목표이고, 정렬들의 올바른 동작과 더불어 이를 채점하게 됩니다.

3. 뼈대 코드

이번 숙제는 뼈대 코드를 필히 사용해야 합니다. (같은 입출력 여건에서의 객관적인 비교를 위함)
뼈대 코드를 사용하지 않을 경우 입출력 문제나, 잘못된 시간체크 연산에 의해 불이익을 받을 수 있습니다.
또한 뼈대 코드의 내용을 이해해서 자신의 것으로 만들기 바랍니다.
뼈대 코드의 마지막 부분에 각각의 소팅에 대한 함수가 있습니다. 거기서부터 구현을 시작하면 됩니다.

4. 입력 형식숫자 입력 -> 소팅 방법

1. 입력은 숫자의 나열로 이루어지며, 첫 줄에 총 숫자의 갯수, 둘째줄부터 각각의 숫자가 입력됩니다.

```
9                (총 숫자의 갯수)
100
7
925
-234
10
-9999
12738
239
-2391            (여기까지 총 9 개)
```

2. 숫자의 나열 대신 난수를 입력할 수도 있습니다. 이 때는 첫 줄에 r (갯수) (최소값) (최대값) 이 오고 끝납니다.

   r 1000 -30000 30000 (-30000 에서 30000 까지(-30000 과 30000 도 포함) 1000 개의 숫자를 랜덤하게 생성하고 그것을 입력으로 대신한다)

3. 숫자의 나열이 정의 되었으면 다음 줄에서 소팅 방법이 입력됩니다. 방법은 B, I, H, M, Q, R (2 번 항목의 소팅 종류 참조) 또는 S (최적 알고리즘 탐색) 중의 한 글자로 입력됩니다.
4. 소팅 방법이 입력되면 소팅을 수행하고 수행결과를 출력합니다. 출력한 다음 다시 소팅 방법을 입력받습니다. (3 번으로 돌아감) 만약 소팅 방법에 X 가 입력되었다면 종료합니다.
   소팅 입력의 모든 숫자는 정수이며 int 의 범위 내에 속합니다. 숫자들의 총 갯수는 1000000 개 이하입니다

5. 출력 형식 2 가지 입력 방법(숫자의 나열 or 난수)에 따라 다르게 출력

1. 숫자의 나열이 입력되었다면 정렬된 숫자들을 오름차순(작은값에서 큰값으로) 으로 한 값당 한줄에 출력합니다.(정렬 결과 출력)
   (작성한 코드가 정렬을 잘 수행하는지를 확인하고자 하는 목적)

   -9999
   -2391
   -234
   7
   10
   100
   239
   925
   12738

2. 난수가 입력되었다면 정렬을 수행하되 결과는 출력할 필요 없고 대신 수행 시간(ms)를 출력합니다.
   (작성한 코드가 각각의 정렬 방법에 따라 어느 정도의 시간이 걸리는지를 알아보고자 하는 목적)
   뼈대 코드에 구현이 되어 있습니다.

6. 입출력 예제

```
$ java SortingTest                                          ← 프로그램 실행
5                                                ← 이렇게 입력(숫자가 총 5 개라는 뜻)
3                                                    ← 이렇게 입력(첫번째 숫자)
1                                                    ← 이렇게 입력(두번째 숫자)
9                                                    ← 이렇게 입력(세번째 숫자)
3                                                    ← 이렇게 입력(네번째 숫자)
4                                                  ← 이렇게 입력(다섯번째 숫자)
Q                                          ← 이렇게 입력(Quick Sort 를 수행하라)
1                        ← 이렇게 출력(QuickSort 로 오름차순 정렬한 첫번째 숫자)
3                        ← 이렇게 출력(QuickSort 로 오름차순 정렬한 두번째 숫자)
3                        ← 이렇게 출력(QuickSort 로 오름차순 정렬한 세번째 숫자)
4                        ← 이렇게 출력(QuickSort 로 오름차순 정렬한 네번째 숫자)
9                      ← 이렇게 출력(QuickSort 로 오름차순 정렬한 다섯번째 숫자)
B                                          ← 이렇게 입력(Bubble Sort 를 수행하라)
1                       ← 이렇게 출력(BubbleSort 로 오름차순 정렬한 첫번째 숫자)
```

```
3                    ← 이렇게 출력(BubbleSort 로 오름차순 정렬한 두번째 숫자)
3                    ← 이렇게 출력(BubbleSort 로 오름차순 정렬한 세번째 숫자)
4                    ← 이렇게 출력(BubbleSort 로 오름차순 정렬한 네번째 숫자)
9                  ← 이렇게 출력(BubbleSort 로 오름차순 정렬한 다섯번째 숫자)
S                                  ← 이렇게 입력(Search 를 수행하라)
Q                   ← 이렇게 출력(가장 빠른 정렬 방법)
X                                          ← 이렇게 입력(이제 종료해라)
$                                                ← 프로그램 종료
```

```
$ java SortingTest                                          ← 프로그램 실행
r 100 -200 200    ← 이렇게 입력(숫자가 총 100 개이며 범위는 -200 에서 200 사이에서 랜덤하게 생성)
Q                                          ← 이렇게 입력(Quick Sort 를 수행하라)
100 ms                    ← 이렇게 출력(QuickSort 를 수행하는데 걸린 시간.)
B                                          ← 이렇게 입력(Bubble Sort 를 수행하라)
10000 ms                ← 이렇게 출력(BubbleSort 를 수행하는데 걸린 시간.)
X                                          ← 이렇게 입력(이제 종료해라)
$                                                ← 프로그램 종료
```

7. 보고서

이번 과제는 여러 정렬 알고리즘을 비교하는 것이 목적이므로 보고서를 받습니다. 보고서는 다음의 내용을 포함해야 합니다.
과제 제출시 보고서 파일명은 Report.pdf 로 제출하세요 (4 장 이내).

1. 정렬 알고리즘의 동작 방식. 정렬의 경우 실행 결과만으로 알고리즘에 맞게 구현했는지 알 수 없으므로, 이에 대한 **설명**을 적습니다.
2. 동작 시간 분석. **정렬에 걸린 시간을 측정하고 이를 분석합니다.** 특히, **data 의 개수에 따른 분석은** 반드시 포함하도록 합니다.
3. Search 동작 방식. 어떤 아이디어들을 어떻게 구현했는지, 하이퍼파라미터는 어떻게 설정했는지 설명하도록 합니다.
4. 동작 시간 분석. 모든 정렬을 일일이 수행하는 것과 Search 를 동작시키고 출력되는 정렬을 한번만 실행시킨 것의 시간을 비교해봅니다.
5. **4 장을 다 채우지 않아도 감점은 없습니다. 표지는 사용하지 않고 상단에 이름과 학번만 적습니다.**
6. **'맑은 고딕' 글자크기는 10pt 으로 보고서를 작성**합니다.

단, 다음의 경우 감점이 될 수 있습니다.

1. 4 장을 초과하는 경우. 전달하고자 하는 내용을 간결하게 요약하는 것이 중요합니다.

2. 알고리즘 설명 시 다른 곳의 설명을 단순히 복사한 경우. 자신이 알고리즘의 원리를 이해했다는 것을 보여야 합니다.
3. 구현 내용을 일일이 설명하려 하는 경우. 핵심 내용에 대한 설명만 적도록 합니다.
4. 실험 결과를 단순히 나열한 경우. 실험 결과 역시 요약해서 적습니다. 가령 같은 조건에서 여러 번 실험한 경우, 실험횟수/최대값/최소값/평균/표준편차를 적는 정도로 충분합니다.

8. 유의사항

1. 효율적인 알고리즘 사용 여부가 점수에 영향을 줄 수 있습니다.
2. 수행 시간은 입출력 부분은 빼고 실제 소팅 시간만 재도록 합니다.
3. **제출 전, 반드시 컴파일 가능 유무와 프로그램의 수행시간을 확인**합니다. 컴파일시 옵티마이제이션 옵션은 쓰지 않습니다.

⑩ 아래와 같은 명령어를 입력하면 컴파일이 이루어져야 하며, SortingTest 라는 이름의 클래스가 생성되어야 채점이 이루어집니다.

```
$ javac SortingTest.java
```

⑩ 컴파일 후, 일정 시간 초과시 프로그램을 강제 종료시키기 위해 다음과 같이 timeout 명령어를 반드시 사용합니다.

```
$ timeout [수행시간(초)] java SortingTest
        timeout 0.5 java SortingTest // 0.5초 수행
        timeout 1 java SortingTest // 1초 수행
```

⑩ 수행시간 측정을 위해 다음과 같이 time 명령어를 사용할 수 있습니다.

```
$ time java SortingTest
```

6. 자주 묻는 질문(필독!!!)

1)채점

과제4는 프로그램채점 : 보고서(코드 구현 포함) = 5 : 5입니다.

코드에서 객체지향에 관한 것은 채점하지 않습니다. 뼈대 코드를 어기지 않았는지, 정렬 알고리즘이 올바르게(효율적으로) 구현되었는지 확인합니다.
채점에 사용되는 테스트셋은 재귀로 인한 Stack overflow가 일어나지 않게끔 적당히 조절되어 있습니다.

2) 실행시간 측정

sorting 시간을 측정할 경우 세번째 실행(run 이후 같은 정렬 알고리즘 3번 이상 호출)부터 수행시간이 급격하게 감소하는 경우가 있는데, 이것은 컴파일러의 특징(JIT >컴파일)일 것 같습니다. 시간 측정은 매번 run에 한번만 측정하거나 3번째 이후부터 측정하는 등 같은 조건에서 측정하시면 됩니다. 혹시 실험하시다가 이런 경우가 발생하면 참고하시면 됩니다.