External documents

1. Main
   1. large size의 배열을 정의하고 난수생성 객체를 생성하여 랜덤한 값으로 배열을 채운다. 그리고 각 배열들의 객체를 생성하고 객체 내에 sort를 접근하여 하나씩 정렬한다.
   2. Small size의 배열을 정의하고 sorted 되었을 때와 랜덤할 때를 구분하여 보여준다.

나머지는 위와 같고, class에 대한 자세한 설명은 class 별로 하겠다.

1. Compare 클래스

비교하는 값을 담을 수 있는 class 이다. Compare()는 호출당하면 count+1 시킨다.

Showcount()는 count 값을 리턴한다.

1. Insertion\_sort 클래스

성능 측정을 위해서 System.nanotime을 사용하였고 이것은 함수 코드의 시간을 측정할 때 사용되는 함수이다. 값을 나노세컨드로 반환한다.

Clone()으로 함수를 복사하여 사용해서 main에서 배열은 값이 변함이 없도록 하고 복사

받은 배열로 정렬을 진행한다. Compare 객체를 생성해서, 정렬이 일어나는 동안 비교횟

수를 카운트한다. 카운트 횟수와 소요시간을 출력하며 클래스가 종료된다.

1. Merge\_sort 클래스

Divide and conquer 알고리즘이다. 재귀로 merge\_sort를 호출한다. merge에서 비교와 합병이 이루어진다. Temp[]배열에 복사하여 merge 후 arr에 넣는 방식으로 구현했다. 나머지는 기본적인 merge\_sort 구현과 동일하다.

1. Print 클래스

배열을 clone해서 사용하지 않을 때 정렬되었는 지 확인하기 위해 만든 클래스이다.

1. Quick sort first element

첫번째 원소를 pivot으로 하는 quick sort이다. Divide and conquer 알고리즘이다. Quicksort를 재귀해서 호출하고, pivot을 구하고 비교하는 함수는 partition으로 정의했다.

1. Quick sort median 클래스

Pivot을 median of three 에 의해서 구하는 sort이다. 나머지는 위와 같다.

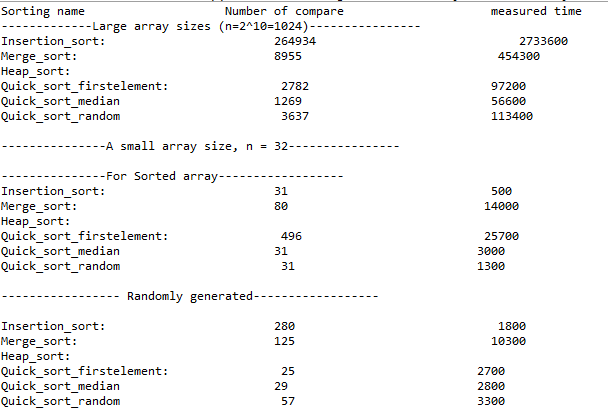
1. Quick sort random 클래스

Pivot을 random하게 구하는 sort이다 . 나머지는 위와 같다.

1. Random number 클래스

선형합동법에 의거해서 random 함수를 만들었다. System.currentTimeMillis()로 현재 시스템 시간을 가져온다. 고정된 값으로 나눠서 랜덤한 값을 얻는다.

1. 결과



1. 배운점

Large size 에서는 insertion\_sort가 가장 비교 횟수가 가장 많고, 시간도 가장 오래걸린다.

Quick sort가 대체적으로 가장 비교횟수가 적고, 시간도 가장 적게 걸린다.

작은 사이즈의 인풋에 대해서 첫번째 이미 정렬되었을 경우에는 insertion\_sort와 Quick sort가 가장 비교횟수가 적고, 시간은 insertion\_sort가 가장 적게걸린다.

두번째 정렬되지 않은 배열에 대해서는 quick sort가 가장 적은 비교횟수와 대체로 적은 시간이 걸린다.

따라서 작은 input에 대해서는 insertion sort와 quick sort 가 좋다고 볼 수 있다.

큰 input에 대해서는 quick sort가 가장 좋다고 볼 수 있다. 따라서 quick sort와 insertion sort를 인풋 크기와 정렬되었는 지 여부에 따라 결정하여 사용하는 것이 효율적이라고 생각한다. Heap sort는 구현을 못했습니다.