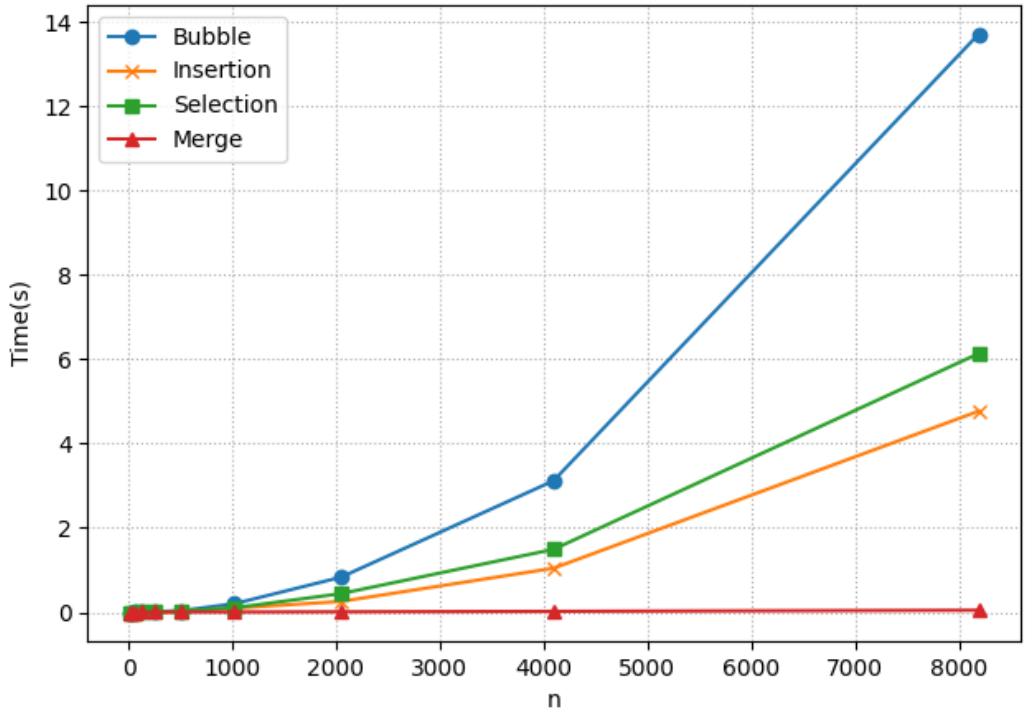
3주차 과제 / 3-1: 정렬들의 성능비교



1. 머지가 제일 빠르다. 나눠서 비교하고 합치는 거라 비교 연산의 횟수가 적어서 리스트 크기가 커져도 괜찮나보다.

2. 인서션과 셀렉션의 차이는 어디서 오는지 궁금해서 열심히 티스토리를 뒤져봤다. 그러나 둘 다 O(n^2)으로 비슷하다. 리스트 크기가 커지면 인서션이 더 괜찮다. 라고 간단하게 나와 있었다. 이게 궁금한게 아닌데….

-> 다행히도 Quora에서 똑같은 질문을 한 사람이 있었다.

The primary distinction between insertion sort and selection sort is that insertion sort sorts by exchanging one element at a time with the partially sorted array, whereas selection sort sorts by selecting the smallest element from the remaining elements and exchanging it with the element in the correct location.

셀렉션 소트: 제일 작은 수를 찾기 위해 남은 애들을 다 뒤져봐야한다.

인서션 소트: 어느정도 정렬된 배열에서 한 번에 하나씩만 교환

개인적인 추론으로는 선택 정렬에서 전체 배열을 다 탐색해야 한다는 데서 차이가 벌어지는 것 같다. 만약에 1 2 3 8 7 9 10 11 … 8379274 이렇게 있을 때, 7이 앞에서 발견되었는데도 끝까지 다 돌아봐야 하므로(중간에 6.1 이런거 있을수도 있으니까) 비효율적인듯?

반대로 삽입정렬 같은 경우에는 한 루프에서 한 요소에 대한 자리를 찾아주기만 하면 된다. 셀렉션과 다르게 이너루프에서 또 한 번 끝까지 루프를 돌리지는 않는다는 것이다.

```
def selection(arr):
    for i in range(len(arr) - 1):
        tmp = arr[i]
        idx = i
        for j in range(i + 1, len(arr)):
            if arr[j] < tmp:
                tmp = arr[j]
                idx = j
        arr[i], arr[idx] = arr[idx], arr[i]
    # print(arr)

def insertion(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and arr[j] > key:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
```

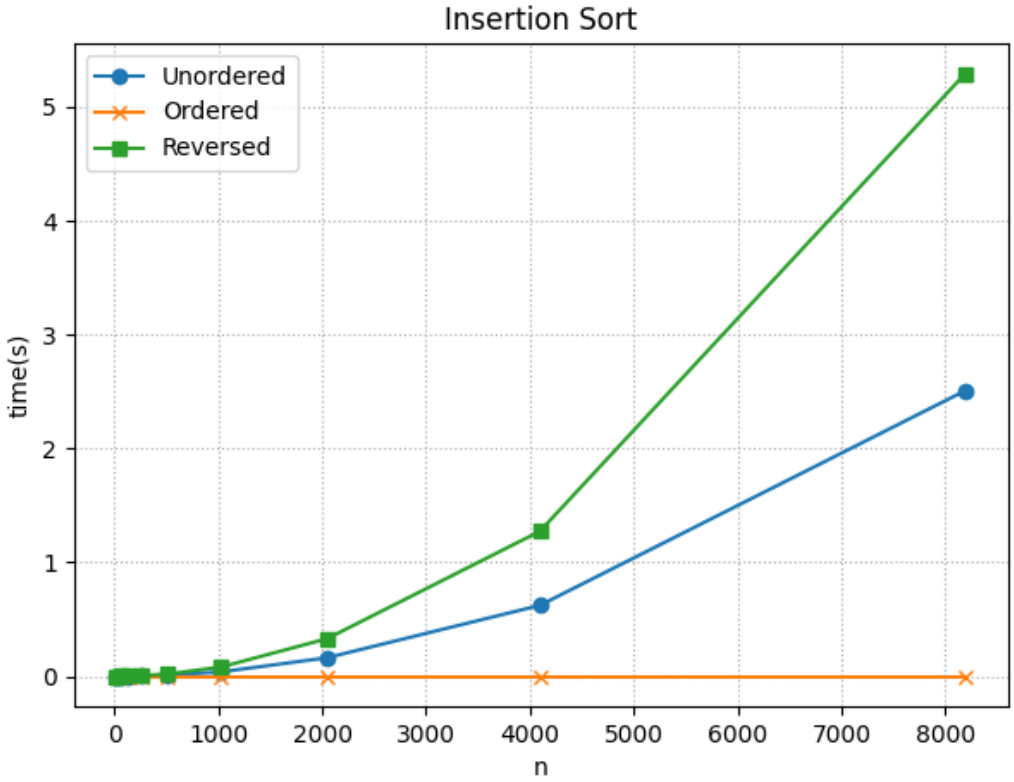
그니까 형광펜 친 코드의 차이 같다. 사실 잘 모르겠다.

3. 버블은 성능이 안좋다. 왜 이렇게 구린걸까 궁금해서 알아봤다. 버블 정렬은 루프를 한 번 돌 동안에 모든 인접 요소를 비교하고 교환한다. 그걸 루프가 끝날 때까지 한다. 그니까 연산을 개 많이 할 수밖에 없다.

버블 vs 삽입: 삽입 정렬은 어느 정도 정렬되어 있는 경우에 거~의 연산을 안 한다. 최선의 경우 O(n)으로 꽤나 괜찮다.(과제를 3-2 먼저 해서 안다.) 반면 버블 정렬 같은 경우 이미 정렬된 배열에서도 비교를 열심히 한다.

버블 vs 선택: 선택 정렬은 배열 끝까지 비교연산을 하지만, 그렇다고 교환을 하면서 비교하지는 않는다. 그저 미니멈 값을 찾고 싶어 하는 것뿐…. 반면 버블 정렬은 교환도 한다. 불필요한 연산 횟수가 많아지는 것이다.

3주차 과제 / 3-2: 삽입정렬 성능비교



삽입 정렬의 시간 복잡도

최선: O(n) / 최악: O(n^2)

1. Reversed: 최악의 경우에 해당한다.(고 티스토리 글에서 봤다.) O(n^2)의 시간복잡도를 가져서 가장 성능이 별로다.

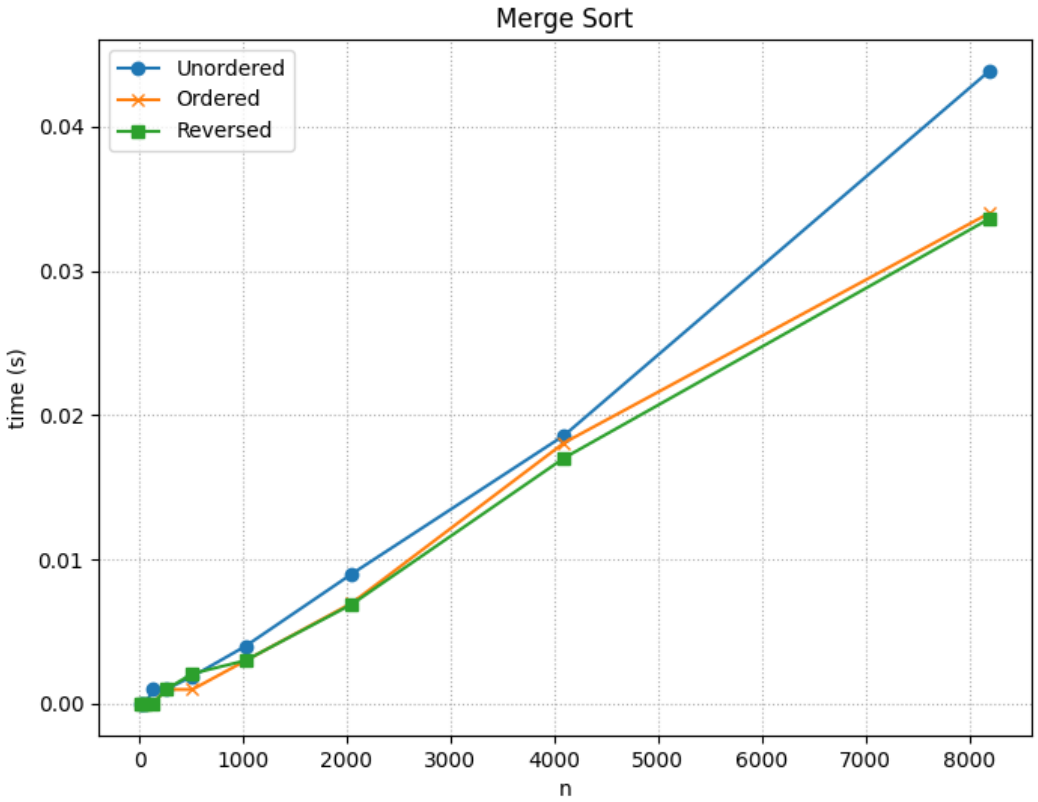
2. Unordered: 무작위로 발생시킨 리스트를 정렬할 경우에 평균의 시간 복잡도를 가진다고... 티스토리 글에서 봤다. 그래서 중간에 끼어들어가 있다.

3. Ordered: 티스토리 글에 의하면 최선의 경우에 해당한다. 그래서 O(n)의 시간 복잡도를 가진다. 그도 그럴게 정렬할게 없으니까 이너루프에서 할게 없다. 아우터루프만 열심히 돌려서 제일 좋다~

=> 전체적으로 봤을 때, n이 작으면, 그러니까 정렬할게 많지 않으면 셋 다 비슷하다. 당연한 소리겠지만 연산량이 많아지면서 차이가 잘 보이게 된다.

=> 티스토리 글에... 따르면 O(n^2)의 시간복잡도를 가지는 애들은 정렬할게 많지 않은 상황에서 쓰는게 제일 효율적이라고 한다~!

3주차 과제 / 3-3: 병합정렬 성능비교



음…. 하긴 했는데 사실 왜 저렇게 나오는지 모르겠습니다…. 아무리 뭘 만져봐도 올려주신 그림과는 다르게 나와요…. ㅜㅜㅜㅜㅜㅜ

내가 이해한 병합 정렬은 어떤 리스트가 들어오든 O(nlogn)의 성능을 가진다는 사실이다. 정렬이 되어 있든, 거꾸로 정렬이 되어 있든, 랜덤이든 간에, 병합 정렬로 들어가는 순간 하나 남을 때까지 쪼개진다. 그 이후 세개의 while문에 들어가서 리스트가 소진될 때까지 정렬을 한다. 셋 다 똑같이 나와야 하는거 아닌가? 라는 생각이 들어서 이것저것 알아봤는데, 다 머지 소트는 데이터와는 무관하다는 대답밖에 없었다. 흠….

모르겠습니다 교수님…. 왜 저런걸까요…. 죄송합니다….