**알고리즘**



**Programming\_Assignment #1**

|  |  |
| --- | --- |
| **이름** | **변정흠** |
| **학과** | **융합전자공학과** |
| **학번** | **201810879** |
| **과목** | **알고리즘** |
| **담당교수** | **김윤호 교수님** |
| **제출일자** | **2022. 11. 04.** |

실험 조건 2가지

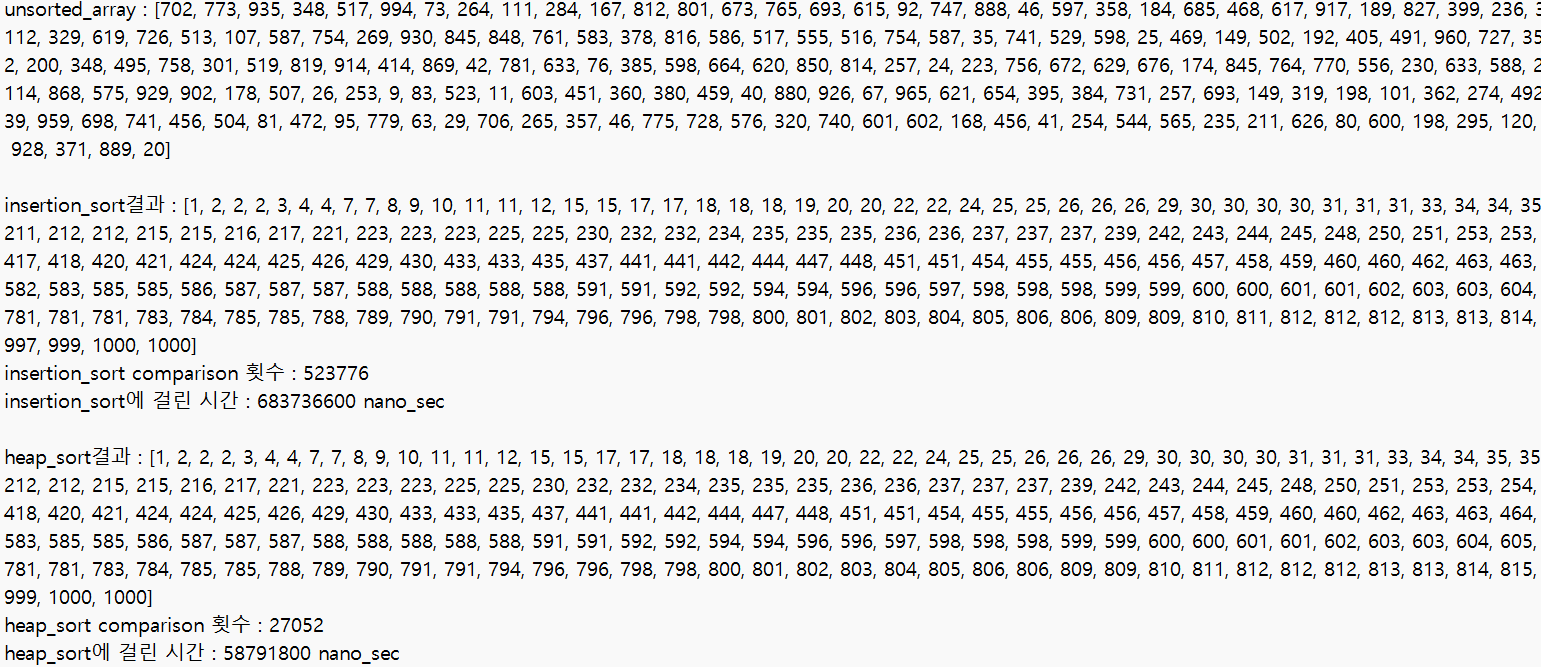
1. Random array의 size가 32일 때.
   1. sorted array
   2. Randomly Generated
2. Random array의 size가 1024일 때.
3. **텍스트이(가) 표시된 사진

   자동 생성된 설명insertion\_sort, Heap\_sort, Merge\_sort, Quick\_sort\_ver1, Quick\_sort\_ver2, Quick\_sort\_ver3, Quick\_sort\_ver4 (size: 32, input : sorted)**
4. **insertion\_sort, Heap\_sort, Merge\_sort, Quick\_sort\_ver1, Quick\_sort\_ver2, Quick\_sort\_ver3, Quick\_sort\_ver4 (size: 32, input : Unsorted)**

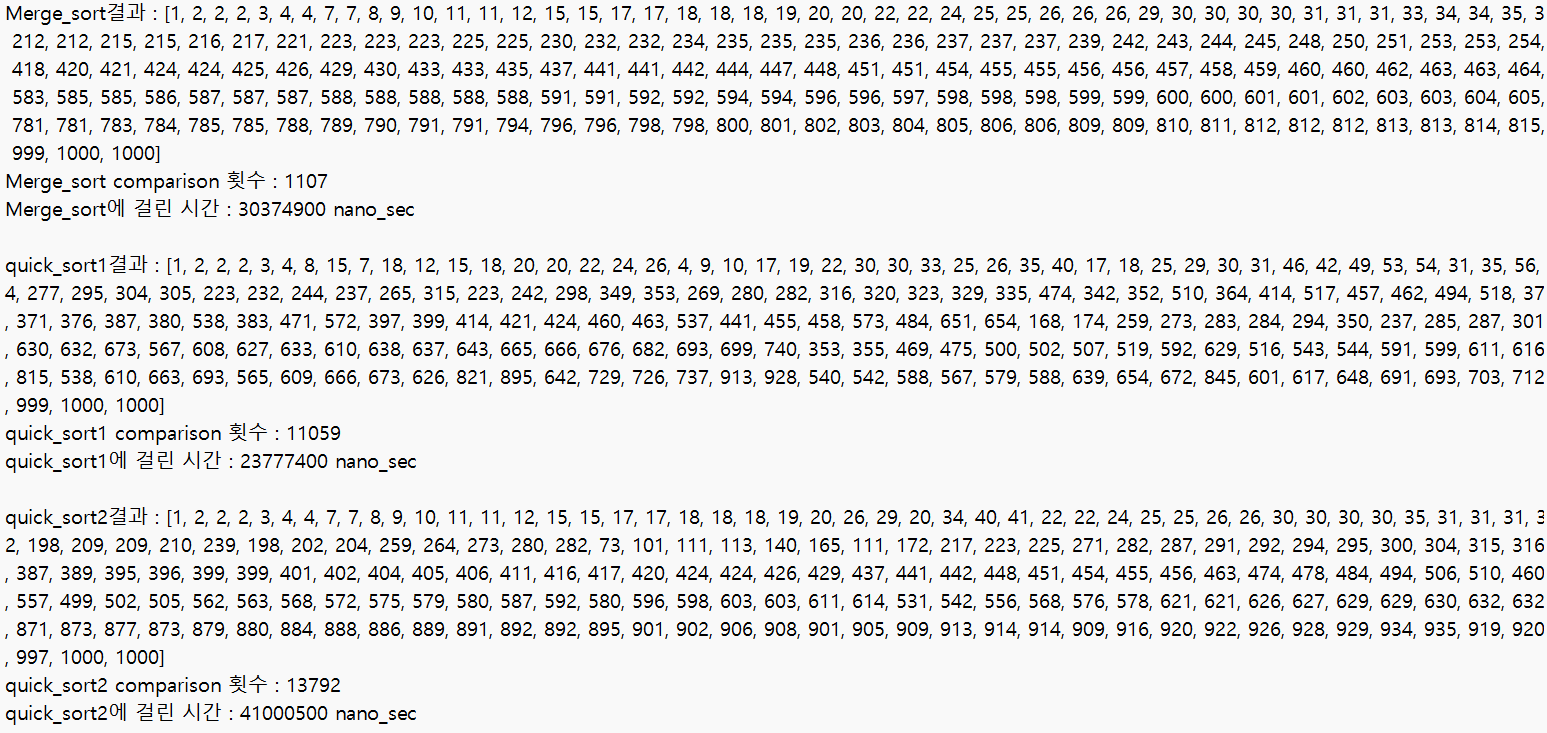
**텍스트이(가) 표시된 사진

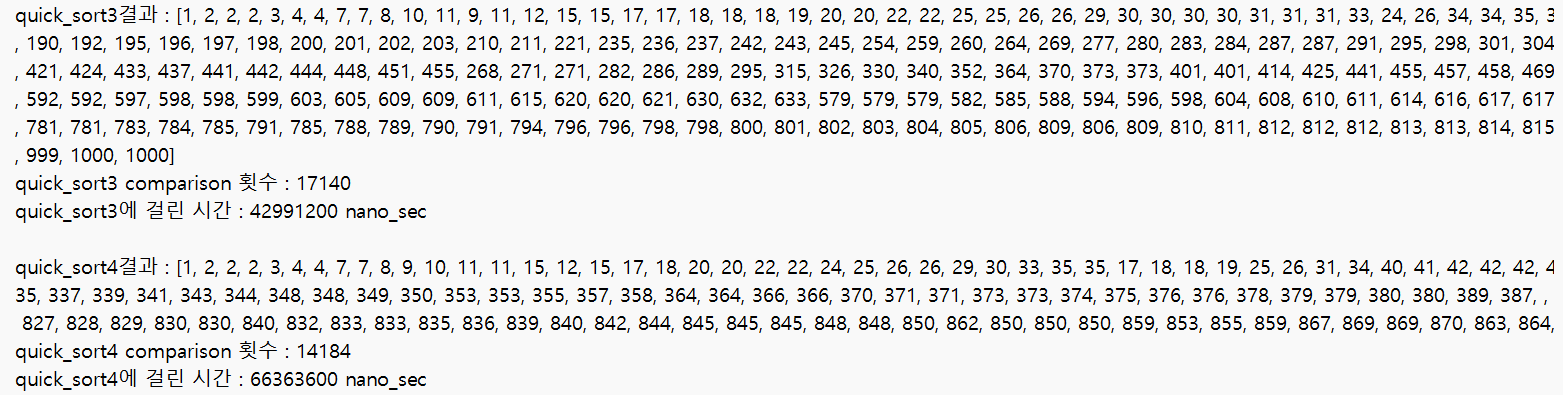
자동 생성된 설명**

1. **insertion\_sort, Heap\_sort, Merge\_sort, Quick\_sort\_ver1, Quick\_sort\_ver2, Quick\_sort\_ver3, Quick\_sort\_ver4 (size: 1024, input : Unsorted)**

****

**Daawddda**

****

****

|  |  |  |
| --- | --- | --- |
| **Sorted Input에서 알고리즘별 효율** | **Unsorted\_Input에서 알고리즘별 효율** | **Unsorted\_Input(n=1024)에서 알고리즘별 효율** |
| **1. insertion\_sort (비교 횟수: 496)**  **2. Heap\_sort(비교 횟수: 395)**  **3. Merge\_sort(비교 횟수: 16)**  **4. Quick\_sort\_ver1(비교 횟수: 554)**  **5. Quick\_sort\_ver2(비교 횟수: 293)**  **6. Quick\_sort\_ver3(비교 횟수: 270)**  **7. Quick\_sort\_ver4(비교 횟수: 234)** | **1. insertion\_sort (비교 횟수: 496)**  **2. Heap\_sort(비교 횟수: 372)**  **3. Merge\_sort(비교 횟수: 18)**  **4. Quick\_sort\_ver1(비교 횟수: 185)**  **5. Quick\_sort\_ver2(비교 횟수: 227)**  **6. Quick\_sort\_ver3(비교 횟수: 241)**  **7. Quick\_sort\_ver4(비교 횟수: 214)** | **1. insertion\_sort (비교 횟수: 52.4k)**  **2. Heap\_sort(비교 횟수: 27.1k)**  **3. Merge\_sort(비교 횟수: 1.1k)**  **4. Quick\_sort\_ver1(비교 횟수: 11.1k)**  **5. Quick\_sort\_ver2(비교 횟수: 13.8k)**  **6. Quick\_sort\_ver3(비교 횟수: 17.1k)**  **7. Quick\_sort\_ver4(비교 횟수: 14.2k)** |
| **sorted\_Input에서 걸린시간** | **Unsorted\_Input에서 걸린시간** | **Unsorted\_Input(n=1024)에서 걸린시간** |
| **1. insertion\_sort (걸린시간: 555microsec)**  **2. Heap\_sort(걸린시간 : 509microsec)**  **3. Merge\_sort(걸린시간: 533microsec)**  **4. Quick\_sort\_ver1(걸린시간: 1417microsec)**  **5. Quick\_sort\_ver2(걸린시간: 760microsec)**  **6. Quick\_sort\_ver3(걸린시간: 571microsec)**  **7. Quick\_sort\_ver4(걸린시간: 1062microsec)** | **1. insertion\_sort (걸린시간: 720ms)**  **2. Heap\_sort(걸린시간 : 756ms)**  **3. Merge\_sort(걸린시간: 733ms)**  **4.Quick\_sort\_ver1(걸린시간: 513microsec)**  **5. Quick\_sort\_ver2(걸린시간: 578microsec)**  **6. Quick\_sort\_ver3(걸린시간: 641microsec)**  **7. Quick\_sort\_ver4(걸린시간: 960microsec)** | **1. insertion\_sort (걸린시간: 683.7ms)**  **2. Heap\_sort(걸린시간 : 58.8ms)**  **3. Merge\_sort(걸린시간: 30.4ms)**  **4. Quick\_sort\_ver1(걸린시간: 23.8ms)**  **5. Quick\_sort\_ver2(걸린시간: 41.0ms)**  **6. Quick\_sort\_ver3(걸린시간: 43.0ms)**  **7. Quick\_sort\_ver4(걸린시간: 66.4ms)** |

**결론**

N=32에서 알고리즘 효율 : Quick\_sort\_ver3 > Quick\_sort\_ver2 > Quick\_sort\_ver1> insertion\_sort > Heap\_sort > Quick\_sort\_ver4

N = 1024에서 알고리즘 효율 : Quick\_sort\_ver1 >. Merge\_sort > Quick\_sort\_ver2 > Quick\_sort\_ver3 > Heap\_sort > Quick\_sort\_ver4 > insertion\_sort

**Lesson learned from programming Assignment.**

이번 실습을 통해 Sudo코드를 실제 코드로 구현하는 방법에 대해 깊게 고찰해 볼 수 있었으며 Input Size값에 따라 알고리즘의 효율이 많이 다른 것을 체감해 볼 수 있었습니다. 또 option으로 본인이 pivot값을 찾는 알고리즘을 만들어 보라고 하셨는데 만들면 만들수록 효율이 떨어져 알고리즘을 만드는 것이 얼마나 어려운 일인지를 알 수 있었습니다. 모르는 자료는 찾아보며 공부했는데 틀린부분이 많을 것 같아 추후 더 공부해볼 생각입니다.