**CSE3081 알고리즘 설계와 분석**

**Sorting 방법의 효율적인 구현**

교수 : 임인성 교수님

학번 / 이름 : 20201223 김지오

**1. 다음의 예와 같이 시간 측정에 사용한 CPU의 속도와 메인 메모리의 용량을 기술하라.**

OS : window 11 Home

CPU : Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz 1.80 GHz

RAM : 8.00 GB

Compiler : Visual Studio 2022 Release Mode

**2. 먼저 본인이 구현한 정렬 방법의 성능을 잘 비교할 수 있도록 input size의 범위를 신중히 선택하라. 특히 시간 측정이 가능한 범위에서 매우 큰 input size에 대해서도 실험할 것.**

모든 sort를 완성시킨 후 돌려본 결과

(1) 교과서적인 insertion sort 방법 Input size : 2^14 ~ 2^19

(2) 교과서적인 heap sort 방법 Input size : 2^14 ~ 2^21

(3) min heap 후 insertion sort 방법 Input size : 2^14 ~ 2^19

(4) 교과서적인 quick sort 방법 Input size : 2^14 ~ 2^21

(5) Intro sort 방법 : 2^14 ~ 2^21

(6) Merge sort로 진행하되, 길이가 짧을 시 Insertion Sort 방법 : 2^14 ~ 2^21

위와 같은 범위로 실험하였습니다.

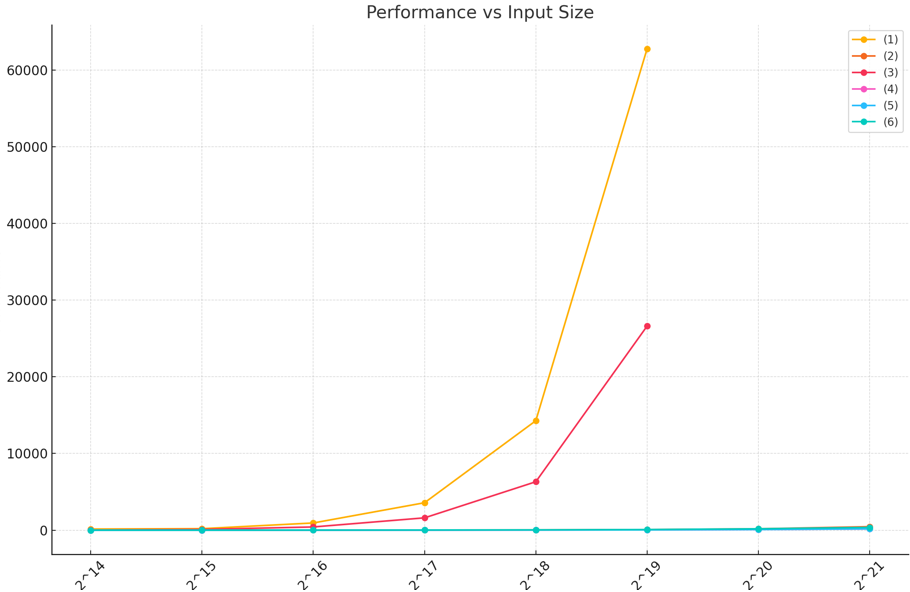
다음 항목의 결과 테이블을 깔끔하게 만들기 위해 (1)~(6) 방법으로 기입했습니다.

**3. 각 정렬 함수에 대한 시간 측정 결과를 적절한 형식의 테이블로 요약하라. 이때 각 방법의 시간 측정 결과 의 특성을 효과적으로 비교하기 위하여 그래프를 그린다던가 또는 수식으로 함수 관계를 보인다던가 하는 등의 자신만의 공학적인 방법을 사용하라.**

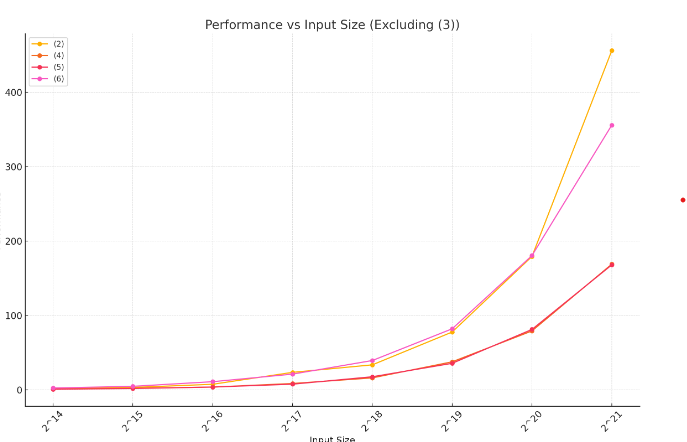
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | (1) | (2) | (3) | (4) | (5) | (6) |
| 2^14 | 147.74 | 2.18 | 23.39 | 0.89 | 0.81 | 2.39 |
| 2^15 | 198.11 | 3.16 | 102.37 | 2.06 | 1.75 | 4.68 |
| 2^16 | 940.13 | 7.43 | 419.86 | 3.63 | 3.72 | 10.89 |
| 2^17 | 3579.82 | 23.44 | 1609.42 | 8.47 | 7.55 | 21.05 |
| 2^18 | 14274.99 | 33.52 | 6314.87 | 16.01 | 15.84 | 39.34 |
| 2^19 | 62773.31 | 77.62 | 26612.55 | 37.61 | 35.58 | 81.97 |
| 2^20 |  | 179.19 |  | 79.10 | 79.21 | 180.40 |
| 2^21 |  | 455.99 |  | 168.9 | 167.9 | 355.74 |

(1) : insertion sort, (2) : heap sort, (3) : insertion sort after min heap, (4) quick sort, (5) intro sort, (6) : merge sort and insertion sort, Result value unit is ‘ms’

의 도표는 3번의 측정을 평균을 내서 만들어진 결과입니다.



insertion sort와 min heap 후 insertion sort는 다른 sort에 비해 너무 차이가 나서 (1),(3)을 제외하고 그래프를 또 만들었습니다.



**4. 과연 시간 측정 결과가 각 정렬 방법의 이론적인 시간 복잡도와 일치하는지 비교 분석하라.**

(1) Insertion sort의 평균적인 Time Complexity는 O(n^2)입니다. 따라서, input size가 2배가 되면 이론적으로 Time Complexity는 4배가 됩니다. Input Size가 2^14에서 2^15으로 변할 때는 차이가 많이는 없는 것으로 보아, 2^14일때보다 2^15일때가 조금 더 오름차순으로 정렬이 돼 있었다는 추측을 할 수 있습니다. 그 외에는 input size가 2배 커질 때마다 4.74배, 3.80배, 3.99배, 4.40배가 측정이 되었고, 평균적으로 3.7배정도 뛰는 것을 확인 할 수 있었습니다.

(2) heap sort의 평균적인 Time Complexity는 O(n\*log n)입니다. 따라서 input size가 2배가 되면 이론적으로 2배보다 살짝 더 커지는 것을 확인할 수 있습니다. input size가 2배가 됨에 따라 각각 1.45배, 2.35배, 3.15배, 1.43배, 2.32배, 2.31배, 2.54배로 평균 2.2배로 이론 값과 매우 비슷하게 나왔습니다.

(3) Insertion sort after min heap은 평균적인 Time Complexity는 알 수 없지만, min heap은 어느 정도 sort가 돼 있는 상태를 의미하며, 그 상태에서 Insertion sort를 하는 것은 Insertion sort보다는 적게 걸리는 것을 확인하면 됩니다. 결과도 또한 모든 경우에서 Insertion sort보다 적게 나오는 것을 확인하였습니다.

(4) 일반적인 Quick sort의 평균적인 Time Complexity는 O(n\*logn)이며, 최악의 경우 O(n^2)까지 나옵니다. 따라서 2배에서 4배 사이지만 2배에 가까울 것이라는 추측을 가지고 있었습니다. Input size가 2배가 됨에 따라 2.31배, 1.76배, 2.33배, 1.89배, 2.35배, 2.1배, 2.14배를 도출하였고, Time Complexity는 평균 2.13배가 나왔습니다. 이론과 비슷한 수치를 도출해 냈습니다.

(5) Intro sort의 평균적인 Time Complexity는 O(n\*logn)이며, 최악의 경우도 O(n\*log n)입니다. 이론적으로 Intro sort는 quick sort를 하되, 최악을 피하기 위하는 방법이기 때문에 quick sort보다 적게 나올 것이라 예측하였습니다. Intro Sort의 Time Complexity의 데이터 값을 확인해 보면 Quick Sort와 매우 비슷하면서 Time Complexity가 0.91배, 0.84배, 1.02배, 0.89배, 0.99배, 1.00배, 0.99배를 기록하며, 평균적으로 0.94배라는 유의미한 수치가 나왔습니다. Input Size에 따른 Time Complexity의 차이는 Quick Sort와 비슷한 수치를 기록하였습니다.

(6) merge sort를 하며 값이 작을 때는 Insertion Sort의 Time Complexity는 O(n\*log n)에서 조금 더 최적화된 상태라 생각했습니다. Input size가 2배가 됨에 따라 1.96배, 2.33배, 1.93배, 1.87배, 2.08배, 2.2배, 1.97배로 평균적으로 2.15배가 나와 이론과 매우 비슷하게 나왔습니다.

**5. 본 숙제에서 시도한 정렬 방법을 위한 최적화 기법들이 얼마나 효과가 있었는지 분석하라**

.(3)의 Min heap을 만든 뒤, Insertion sort는 Insertion sort보다 더 좋은 결과를 도출하였습니다. 다만, heap sort보다 Time Complexity가 크게 높았기 때문에 길이가 긴 array로 Insertion sort는 정렬이 잘 돼있다고 한들, 완전한 Sort가 돼 있지 않는 이상 비효율적인 방법일거란 결론을 도출하였습니다.

(5)의 Intro Sort는 Quick sort의 최악인 상황의 Time Complexity를 방지하기 때문에, 3번의 실행결과로는 유의미한지를 알 수 없으나, 결과값만 보면 약 0.94배라는 수치를 도출해냄으로써 Quick Sort보다 최적화 된 Sort라고 말할 수 있었습니다.

(6) Merge sort를 하며 값이 작을 때 Insertion Sort의 경우 Merge sort는 하나의 원소가 될때 까지 Divide하기 때문에 배열의 크기가 15이하면 Insertion sort를 하는 것은 최적화를 했다고 볼 수 있습니다.