**HW6: Sorting** 202211390 최준원

|  |
| --- |
| 1/ array의 크기가 증가함에 따라서, 평균의 경우(random data) 각 sorting algorithm의 time complexity를 측정해 보자.  - n=10,000인 array에 정수 data를 random하게 대입한다.  - 각 sorting algorithm이 같은 array를 sort해야 하기 때문에, 위의 array를 6개의 set로 만든다. - 각 sorting algorithm이 sorting을 수행한 시간을 순차적으로 측정한다.  - n=20,000, 30,000, ..., n=100,000일 때, 같은 실험을 반복 실행하여, 각 알고리즘의 time complexity를 비교하여 결론을 제시하고, 알고리즘들의 수행 속도 순위에 대한 결론도 제시하시오.  <스크린샷>  텍스트, 스크린샷, 메뉴, 폰트이(가) 표시된 사진  자동 생성된 설명텍스트, 스크린샷, 메뉴, 디자인이(가) 표시된 사진  자동 생성된 설명  위의 결과값을 그래프로 정리해보면 아래와 같다.  위 그래프의 값들은 배열의 크기에 따른 6개의 Sorting Algorithm의 정렬되지 않은 행렬을 정렬하는 수행시간(ms)을 측정한 값이다. 수행시간의 차이는 존재하나, Bubble Sort, Selection Sort, 그리고 Insertion Sort는 행렬의 크기가 증가함에 따라 수행시간의 변화량도 증가하는 양상을 보인다. 그에 반해 Merge Sort, Quick Sort, 그리고 Heap Sort는 그 외의 Sort들과는 다르게 수행시간의 변화량이 크게 증가하지 않았다. 즉 Bubble Sort, Selection Sort, Insertion Sort는 각 요소가 Random한 크기인 배열에 대해 O(n^2)의 Time Complexity를 가지며, Merge Sort, Quick Sort, Heap Sort는 O(nlogn)의 Time Complexity를 가진다.  알고리즘의 수행 속도는 Quick Sort가 제일 빨랐고, Merge Sort, Heap Sort, Insertion Sort, Selection Sort, Bubble Sort 순으로 빨랐다. Bubble Sort는 다른 Sorting Algorithm에 비해 특히 더 많은 시간이 소요됐으며, Bubble Sort 다음으로 느린 Selection Sort보다 수행 시간이 2배 더 많이 소요되었다. |
| 0/ 1번 실험을 진행하기 전에, 각자 구현한 sorting algorithm들이 정확히 동작하는지를 보이시오. (n=30 정도)  <스크린샷>  텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진  자동 생성된 설명  Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quick Sort, Heap Sort에 대해 정렬되지 않은 같은 행렬을 넣었을 때 동일한 정렬된 행렬이 나오는지를 크기 30의 배열로 확인해본 것이며, 2번 문제에서 활용되는 Random Quick Sort는 정렬된 행렬을 정렬하는 경우를 다루므로 결과가 정렬된 행렬과 상동임을 확인해본 것이다.  2/ 1번 실험은 데이터들이 random하게 섞여 있는 경우를 가정하였다. 그러나, 몇몇 알고리즘들은 데이터의 상태에 따라 알고리즘의 수행속도가 달라질 수 있다. 이러한 점을 실험하여 제시하시오.  - Bubble sort (adaptability를 위한 수정 버전)의 경우, sorting이 된 데이터를 입력함으로써, O(n)의 time complexity를 보이시오 (n=10,000 ~ n=100,000). 실험 결과는 머신의 성능에 따라서, 실행시간이 0 milisec으로 나와서 O(n)인지 정확히 확인이 되지 않을 수도 있는데, 이 경우는 그대로 제출하시오. (대신, (1)번 실험에서 같은 크기의 random data보다는 훨씬 빠른 시간에 sorting이 끝나는 것을 확인하시오.)  - Insertion sort도 bubble sort와 동일.  - Quick sort의 경우는 이미 sorting된 데이터를 입력받으면, 성능이 안 좋아져서 O(n2)의 성능을 보임을 제시하시오 (n=10,000 ~ n=100,000).  - Randomized quick sort를 구현하여, sorting된 데이터를 입력받아도 O(n\*logn)의 성능을 보임을 확인하시오 (n=10,000 ~ n=100,000). 이 경우, pivot을 맨 앞의 데이터로 사용하는 것이 아니라, random index를 발생시켜서, 그 index의 데이터와 맨 앞의 데이터를 swap 한 후, 일반 quick sort를 진행하는 방법으로 구현한다.  <스크린샷>  텍스트, 메뉴, 스크린샷, 폰트이(가) 표시된 사진  자동 생성된 설명텍스트, 메뉴, 스크린샷, 폰트이(가) 표시된 사진  자동 생성된 설명  위의 결과값을 그래프로 정리해보면 아래와 같다.  위 그래프의 값들은 위 4개의 Sorting Algorithm의 정렬된 행렬을 정렬하는 수행시간(ms)을 측정한 값이다. 여기서 이용한 Bubble Sort는 adaptability를 위한 수정 버전이며, 이는 데이터가 이미 정렬된(최선의) 경우 별도로 정렬하는 수행을 하지 않는다. Insertion Sort 역시 데이터가 정렬된(최선의) 경우 정렬을 위한 for문을 돌지 않는다. 따라서 이러한 경우 저 두 Sorting Algorithm의 Time Complexity는 O(n)이 된다. 하지만 이 Sorting Algorithm을 실행하는 기기의 성능이 ms 단위로 시간을 재기에는 너무 빠르게 Sort를 수행하기 때문에 Time Complexity를 확인할 정도의 시간 값을 측정할 수가 없었다. 0번에서 이 Sorting Algorithm들이 수행됨을 확인하였으므로, 여기서는 이 두가지 Sorting Algorithm이 정렬된 데이터를 Sort할 때 소요되는 시간이 정렬되지 않은 데이터를 Sort할 때 소요되는 시간보다 짧다는 것은 확인할 수 있었다.  Quick Sort의 경우, 1번 문제에서는 O(nlogn)의 Time Complexity를 가짐을 확인했었으나, 정렬된 데이터를 Sorting하는 2번 문제에서는 성능이 떨어짐을 확인할 수 있다. Quick Sort는 정렬된 행렬을 Sorting하는 경우, 행렬의 크기가 증가함에 따라 수행시간의 변화량도 증가하는 양상을 보인다. 여기서 Quick Sort는 O(n^2)의 Time Complexity를 가진다.  Randomized Quick Sort의 경우, 정렬된 데이터를 입력받아도 Pivot을 랜덤한 index의 데이터와 맨 앞의 데이터를 swap한 행렬로 정하기 때문에 Quick Sort를 수행하기 전에 임의로 정렬되지 않은 행렬을 만들어 Quick Sort가 제일 성능이 나빠지는 경우(정렬된 데이터를 받는 경우)를 피한다. 즉, Quick Sort의 평균의 경우 Time Complexity인 O(nlogn)을 가진다. |