Supplement 보고서

1) 목적 및 아이디어

교재에서 구현한 기본적인 알고리즘으로 각 알고리즘의 특성을 알아본 기존의 report와는 다르게 supplement report에서는 quicksort에 tuning을 가하여 기존의 알고리즘과 얼마나 실행시간의 차이가 나는지 알아보았다. 기존 실험에서 밝혀진 quicksort의 문제점은 두가지가 있는데 중복원소를 처리할 때와 이미 정렬되거나 역정렬된 배열을 정렬할 때 분할정복이 제대로 되지 않는다는 점이다. 첫번째 중복원소들을 처리하는 방법으로는 중복원소들을 따로 모아놓고, 가운데에 넣어주거나 1/2의 확률로 중복원소들을 앞에 붙일지 뒤에 붙일지 결정하는 방법이 있는데 전자의 경우 시간이 조금 더 걸리는 대신 stable sort라는 장점이 있고, 후자는 시간이 좀 더 빠르지만 stable sort가 아니라는 단점이 있다. 우리의 경우는 실행시간을 줄이는 것이 목적이기 때문에 후자의 방법을 선택하였다. 두번째 문제의 해결법으로는 가운데 원소를 기준원소로 하여 partition을 진행하는 것이다. 그렇게 한다면 역정렬, 정렬과 상관없이 partition을 같은 크기로 쪼갤 수 있을 것이다. 두 아이디어를 구현하는 것은 간단한데 첫번째 아이디어는 partition을 생성할 때 중복원소를 만나면 index j를 2로 나누어 0이면 앞의 partition에 1이면 뒤의 partition에 붙이는 방법이다. 그러면 1/2의 확률로 근사되어 partition의 크기가 비슷해진다. 두번째 idea는 median의 원소와 맨뒤 원소를 swap하여 median의 원소를 기준으로 partition을 진행하는 방법으로 구현이 가능하다. (구체적인 구현은 주석으로 추가하였다.)

2) 실험 및 결론

실험은 search에 사용했던 parameter인 충돌횟수와 sorted ratio를 사용하여 데이터 개수 50000개에 대하여 100회 수행 평균시간을 표기했다. 기존 quick sort는 충돌횟수가 증가할수록, sorted ratio가 0.5에서 멀어질수록 정렬시간이 커졌는데 tuning을 한 quicksort는 어떻게 정렬시간이 변화하는지 알아보았다. 추가적으로 나머지 sort중 가장 빨랐던 mergesort와도 비교해보았다.

충돌횟수	1000	500	100	50
Quick	24.5	13.92	5.25	3.71
TuningQ	3.02	3.04	3.73	3.8
Merge	3.38	3.56	4.01	3.75

실험1: 충돌횟수

sort_ratio	1	0.7	0.6	0.5	0.4	0.3	0
Quick	overflow	overflow	303	2.54	124.52	391.8	overflow
TuningQ	1.504	2.48	2.71	3.73	2.94	2.86	1.92
Merge	1.58	2.86	3.37	3.52	3.01	2.81	1.64

실험2: sorted_ratio

표에서 TuningQ는 Tuning을 진행한 Quicksort를 의미한다. 충돌횟수 실험을 진행한 결과 Tuning을 진행한 Quicksort는 충돌횟수와 관계없이 일정한 성능을 유지하였다. 또한 충돌횟수가 크지않은 50개부터는 Quicksort와 정렬시간이 거의 비슷했다. 따라서 중복원소가 많은 배열에서도 일정한 성능을 보여줌을 알 수 있었다. 그리고 merge sort와 비교했을 때도 거의 비슷한 성능을 보여주었는데 만약 데이터개수가 더 커진다면 배열을 되써주는 merge sort보다 확실히 더 나은 성능

을 보여줄것으로 기대된다. Sorted ratio를 변화시킨 실험에서도 마찬가지로 Tuning을 진행한 Quicksort는 일정한 성능을 보여주었는데 특기할 사항으로는 정렬되거나 역정렬된 배열들에 대해서 random에 가까운 배열들보다 더 좋은 정렬 성능을 보여주었는데 이는 median을 기준원소로 잡았을 때 정렬된 배열의 경우 partition이 정확히 반으로 나누어지기 때문인것으로 생각된다. (역정렬 배열도 마찬가지이다.)