MP2 Report

20200422 이수빈

실험 환경은 MacBook Air M1(2020)이었으며, 코어 개수는 8(4 성능 및 4 효율), memory size는 16GB, OS type과 version은 macOS Ventura 13.4.1이었다. 다음과 같은 input file들을 사용해 성능 test를 진행하였다. 원소의 개수를 N, 해당 파일의 어떤 원소를 n이라 할 때,

- input file 1: N = 10^1, 0 <= n < 10
- input file 2: N = 10^2, -123123123 < n < 123123123
- input file 3: N = 10^4, -123123123 < n < 123123123
- input file 4: N = 10^6, -123123123 < n < 123123123

시간 측정 시점은 input file을 open하기 직전부터 output file의 4번째 줄에 측정한 시간을 기록하기 직전까지이다.

1. Average Case (random sequence)

Algorithm	N=10^1	N=10^2	N=10^4	N=10^6
Insertion Sort	237	404	81264	7.09506e+08
Quick Sort	257	442	13322	1.0102e+06
Merge Sort	255	440	15277	1.08572e+06
Optimized Quick Sort	239	403	12790	960476

(ms)

1-1. Input의 개수가 10^1개, 10^2개인 경우

모든 알고리즘의 시간 효율이 비슷했으나 Input의 개수가 작은 경우에는 insertion sort와 optimized quick sort가 근사한 차이로 빨랐으며, insertion sort와 optimized quick sort끼리, quick sort와 merge sort끼리 비슷한 시간이 걸린 것을 확인할 수 있었다.

1-2. Input의 개수가 10^4개, 10^6개인 경우

Input의 개수가 큰 경우에는 optimized quick sort, quick sort, merge sort, insertion sort 순서대로 빨랐다. insersiont sort는 시간복잡도가 O(n^2)인 만큼 눈에 띄게 느린 것을 확인할 수 있었고, quick sort와 merge sort는 시간 효율이 비슷했으나 quick sort가 근사하게 빨랐으며, Optimized quick sort는 그러한 quick sort와 비교하여 조금 더 빨랐다.

2. Best Case (ascending inputs)

Algorithm	N=10^1	N=10^2	N=10^4	N=10^6
Insertion Sort	186	298	7111	668834
Quick Sort	249	377	11258	780826
Merge Sort	242	374	11535	901923
Optimized Quick Sort	231	300	7665	758988

(ms)

2-1. Input의 개수가 10^1개, 10^2개인 경우

모든 알고리즘의 시간 효율이 큰 차이가 없었으나 Input의 개수가 작은 경우에는 insertion sort가 다른 알고리즘보다 조금 더 빨랐다. Quick sort와 merge sort는 측정된 시간이 매우 비슷했으며, optimized quick sort는 input의 개수가 10^1인 경우에는 insertion sort와 비슷했고 input의 개수가 10^2개인 경우에는 quick sort, merge sort와 비슷했다.

2-2. Input의 개수가 10^4개, 10^6개인 경우

Input의 개수가 큰 경우에는 insertion sort, optimized quick sort, quick sort, merge sort 순서대로 빨랐다. 1번의 average case에서는 시간복잡도가 O(n^2)인 insertion sort가 가장 느렸으나 base case에서는 시간복잡도가 O(n)인 insertion sort가 가장 빠른 것을 확인할 수 있었고, 시간복잡도가 O(nlogn)인 나머지 알고리즘의 순서는 그대로였다. Optimized quick sort의 경우 average case에서는 quick sort와 근사한 차이밖에 나지 않았으나 base case에서는 확연히 빨랐다.

3. Worst Case (descending inputs)

Algorithm	N=10^1	N=10^2	N=10^4	N=10^6
Insertion Sort	250	434	140053	1.32289e+07
Quick Sort	240	428	10974	841113
Merge Sort	216	358	9102	244597
Optimized Quick Sort	245	422	10759	924701

(ms)

Input의 개수에 관계 없이 merge sort가 가장 빨랐고, merge sort와 optimized quick sort가 비슷했으며, insertion sort가 가장 느렸다. Worst case에서 merge sort의 시간복잡도는 O(nlogn)이고 나머지 알고리즘의 시간복잡도는 O(n^2)이다.

4. algorithm 4 설명

Algorithm 4는 quick sort algorithm을 조금 변형하여 optimized quick sort를 만들었다. Input size가 작을 때 비교적 좋은 효율을 내는 insertion sort를 보고, quick sort의 base case를 n=1일 때가 아닌 n=16일 때로 변경하였고, 재귀를 멈추면 해당 base case에 대해 insertion sort를 하는 방식으로 구현하였다. 즉, input size가 클 때는 quick sort의 시간복잡도를, 재귀 방식의 divide를 거쳐 input size가 작아졌을 때에는 insertion sort의 시간복잡도를 따르게 함으로써 시간 효율을 최적화했다.