레포트

과목: 알고리즘 설계와 분석(CSE3081-01)

학번: 20221052

이름: 심규재

실험 환경

사용된 기기: 삼성 노트북 컴퓨터 NT900X5L

프로세서: Intel(R) Core(TM) i5-6200U CPU @ 2.30GHz 2.40 GHz

RAM: 8.00GB

시스템 종류: 64비트 운영 체제, x64 기반 프로세서

OS type: Windows 10 Home

Version: 20H2

실험 조건

-모든 실험에서 testcasegen의 숫자 범위는 10000으로 고정 후 실험했음.

-모든 실험에서 실행시간의 측정값은 소수점 5번째 자리에서 반올림 했음.

-실험#1(평균치)의 데이터는 실험#1-1과 실험#1-2의 측정값의 평균으로 산출했음.

-실험#2(내림차순)에 사용된 입력 데이터는 표에서 해당되는 개수, 범위, 시드로 testcasegen에서 생성된 데이터를 mp2_20221052.c의 1번 알고리즘(Insertion Sort)에서 order = -1로 설정한 새로운 c 프로그램(reverse.c)로 내림차순 정렬한 데이터를 사용하였음.

-측정시간이 정상적으로 출력되지 않는 경우에 대해 측정값을 비워두고 에러 메시지를 기재했음.

입력 조건/정렬 알고리즘 별 소요시간 데이터

<표1>

실험#1-1(랜덤) Testcasegen 시드: 1	100개	1000개	10000개	100000개
(1)InsertionSort	0.000025	0.001715	0.141438	10.380402
(2)QuickSort	0.000015	0.000141	0.001869	0.023230
(3)MergeSort	0.000017	0.000299	0.003383	Segmentation fault (core dumped)
(4)OptimizedQuicksort	0.000014	0.000202	0.003160	0.023625

<표2>

실험#1-2(랜덤) Testcasegen시드: 100	100개	1000개	10000개	100000개
(1)InsertionSort	0.000020	0.001184	0.120562	11.397377
(2)QuickSort	0.000013	0.000195	0.002502	0.023257
(3)MergeSort	0.000016	0.000164	0.003307	Segmentation fault (core dumped)
(4)OptimizedQuicksort	0.000014	0.000198	0.001809	0.027456

<표3>

실험#1 평균치	100개	1000개	10000개	100000개
(1)InsertionSort	0.000023	0.001449	0.131000	10.888890
(2)QuickSort	0.000014	0.000168	0.002186	0.023244
(3)MergeSort	0.000017	0.000232	0.003345	측정값 없음
(4)OptimizedQuicksort	0.000014	0.000200	0.002485	0.025541

<표4>

실험#2(내림차순) Testcasegen 시드: 100	100개	1000개	10000개	100000개
(1)InsertionSort	0.000054	0.003368	0.256469	20.668295
(2)QuickSort	0.000028	0.004146	0.200199	9.327495
(3)MergeSort	0.000020	0.000162	0.001141	Segmentation fault (core dumped)
(4)OptimizedQuicksort	0.000045	0.001684	0.103834	4.824500

실험 데이터 분석

<표1>에서 전체적으로 우수한 성능을 갖는 것은 2번(알고리즘)이다. 특이 사항으로는 100개 사이즈의 입력 데이터에서 4번이 유일하게 3번 알고리즘보다 살짝 빨랐는데 이는 입력데이터의 특이성으로 해석 할 수 있다. 3번에서 발생된 에러 메시지(segmentation fault (core dumped))를 검색해보니 메모리 초과 등의 사유로 발생한다고 적혀있어, 우선은 그대로 기재했다. 3번을 충분히 큰 입력데이터를 처리하는데 사용하려면 가정 보급용 사양 이상의 하드웨어가 필요로 하다.

<표2>에서 전체적으로 우수한 성능을 가리기 어렵다. 2번은 100개와 100000개의 입력에서, 3번은 1000개의 입력에서, 4번은 10000개의 입력에서 각자 우수한 성능을 보였다. 3번에 대한 에러는 동일하다.

<표3>에서 <표1>과 <표2>의 데이터를 평균 낸 수치를 볼 수 있다. 100개 입력값에서 2번과 4번이 동일한 성능을 낸 것을 제외하고 다른 모든 입력에서 우수한 성능을 보인다. 에러가 있었던 3번의 100000개의 입력 데이터에 대해서는 성능비교에서 예외로 두었다.

<표4>에서 3번이 전반적인 입력 데이터에서 우수한 성능을 보이나, 이전 실험 결과와 마찬가지로 100000개의 데이터에서는 에러를 출력한다. 특이 사항으로는 2번 Quicksort 알고리즘을 개량한 4번 알고리즘이 100개 이후의 입력 값에서 모두 2배 정도 빠른 성능 차이를 보인다는 점이다.

종합한 결과는 다음과 같다.

-1번은 전체적으로 느리다.

-2번은 전체적으로 빠르다.

-3번은 내림차순 정렬된 입력에서 가장 빠르고, 10만개 이상의 데이터를 처리할 수 없다.

-4번은 전체적으로 2번의 하위호환이지만 내림차순 정렬된 입력에서 2번보다 2배 나은 성능을 보인다.

(4)OptimizedQuicksort 설계 시 고려된 점

Testcasegen을 분석한 결과 입력의 범위(모든 실험에서는 10000으로 고정 되었던)를 받아와 0을 기준으로 (-입력범위 ~ 0 ~ +입력범위) 까지의 값들을 입력 개수 만큼(100,1000,10000,100000) 생성하는 것을 알 수 있었다.

이러한 숫자 생성 역시 정규분포를 따를 것이라 가정하고, Quicksort가 중간 값을 pivot으로 할 때 최고의 성능을 보인다는 점에서 기인하여, Quicksort의 시작시 초기 pivot 값을 0으로 고정하고 그 이후로 일반적인 Quicksort를 재귀호출 하는 알고리즘이 (4)OptimizedQuicksort 이다.

그 구현 방식은 단순히 입력 데이터를 배열에 저장한 다음 배열 맨 끝에 0의 값을 갖도록 추가하고, pivot을 맨끝으로 바꾼 것이다.

알고리즘 설계 시 2번과 4번은 성능이 거의 비슷하거나 4번이 조금 더 우수할 것으로 예상했으나 실제로는 내림차순 입력이 아닌 데이터들에서는 2번이 우수했다.

반면 내림차순 입력 데이터에서는 1000개 이상의 입력 데이터에서 2배 정도의 빠르기를 보인다.