2023-12753 여은수

A. 정렬 알고리즘의 동작 방식

1. Bubble sort

Bubble sort의 동작 방식은 이웃한 2가지 원소를 비교한 후 만일 앞의 원소가 뒤의 원소보다 더 값이 크다면 2가지 원소의 위치를 바꾸는 작업을 반복하는 방식이다.

2. Insertion sort

Insertion sort의 동작 방식은 bubble과 비슷하지만 조금 다르게 작동한다. Index i번째의 원소를 key에 저장하고 이를 index i-1번째까지 원소와 비교한다. 이때 insertion sort과정에서 i-1번쨰까지는 정렬되어 있을 것이다. 따라서 key value보다 큰 값이 등장한다면 key값을 그 index에 삽입하고 뒤의 index는 한자리씩 뒤로 미루는 방식으로 작동한다.

3. Heap sort

Heap sort는 먼저 Heapify를 통해서 주어진 value를 heap tree로 만든다. Heapify의 동작 방식은 다음과 같다. 먼저 주어진 값과 자식들의 크기를 비교해서 최대값을 찾고 최대값이 부모 위치에 없다면 이를 부모 위치에 넣는 방식으로 부모가 항상 자식보다 큰 값을 유지하도록 정리하는 방법이다. 이러한 heapify를 통해서 heapsort는 가장 큰 값을 배열의 뒤로 넣고 나머지 값을 heapify한 후 이를 반복해서 정렬된 값을 얻는다.

4. Merge sort

수업 시간에서는 tmp를 이용하여 merge sort를 구현하였다. 이를 이용하여 주어진 value를 반으로 나누고 이 나눈 value array를 Merge하는 과정을 반복하여 Merge sort를 재귀적으로 구현하였다.

5. Quick sort

Quick sort의 동작방식은 partition을 이용해 입력된 값을 나누어 이를 반복하는 방식이다. 이는 위의 merge sort와 비슷하지만 partition을 이용한다는 점에서 다르다. Partition을 통해서 pivot value보다 작은 값과 크거나 같은 값으로 나누어 이를 각각 quicksort를 돌려 합친다.

6. Radix sort

Radix sort의 작동 방식은 최대값을 통한 countsort를 이용하는 방식이다. 먼저 주어진 value의 최대값을 구한다. 이를 바탕으로 새로운 array를 만든다. Array의 index가 19인 count array를 만들어서 count된 빈도를 저장한다. 이를 바탕으로 output배열에 이를 채워서 주어진 값을 오름차순으로 정렬하는 방식이다.

B. 동작 시간 분석

1. Random 예시 설정

먼저 동작 시간을 분석하기 위한 random 예시의 종류에 대해 생각해 보았다.

Random input code: r N1 N2 N3

a. N1>>N3-N2+1

b. N1=N3-N2+1(같거나 유사할 때)

c. N1<<N3-N2+1

각 경우에 따라서 어떤 sort method가 더 빠른 방법인지 비교하고 이것이 어떠한 영향일지 생각해보고자 한다.

2. Case a: 중복된 입력 값이 많을 때

시행	Bubble	Insertion	Heap	Merge	Quick	Radix
r 5000 -100 100	30	14	3	2	1	3
r 50000 -100 100	4250	700	16	26	13	9
r 150000 -100 100	34513	8158	30	45	86	24
r 15000000 -1000000 1000000	-	-	637	-	-	880

3. Case b: 중복된 입력 값이 거의 없을 때

시행	Bubble	Insertion	Heap	Merge	Quick	Radix
r 200 -100 100	1	0	1	2	3	0
r 2000 -1000 1000	12	9	2	2	1	1
r 20000 -10000 10000	525	134	7	7	3	6
r 200000 -100000 100000	-	-	42	48	30	43

4. Case c: 중복된 입력 값이 없다고 봐도 무방할 때

시행	Bubble	Insertion	Heap	Merge	Quick	Radix
r 5000 -10000 10000	38	19	4	7	2	2
r 5000 -100000 100000	39	11	5	2	3	2
r 50000 -1000000 1000000	7318	770	12	30	11	11

5. 동작 시간 분석

위의 결과를 통해서 우리는 3가지 사실을 알 수 있다.

1. Case a일 때는 radix sort와 heapsort가 비슷한 동작시간을 보여준다.
2. Case b일 때는 Quick sort가 가장 빠르다.
3. Case c일 때는 Quick sort와 Radix sort가 거의 비슷한 동작 시간을 보여준다.

즉 동작 시간을 분석하자면 다음과 같이 정리할 수 있다.

가) 중복된 수의 개수가 많을수록 heap sort와 radix sort가 빨라진다.

이는 Case a에서 중복된 수가 증가할수록 Radix sort가 다른 방법보다 걸리는 시간의 비율이 더 낮아지는 것을 통해서 알 수 있다.

나) 정렬하는 수의 개수가 증가할수록 걸리는 동작시간이 길어진다.

이는 Case b와 c에서 N1 과 N3-N2+1의 비율을 유지하되 정렬하는 수의 크기만 바뀌었을 때를 통해서 알 수 있다.

다) 수의 범위의 자릿수가 작으면 radix sort가 일반적으로 다른 방법보다 빠르다.

이는 Case a, b, c모두에서 찾아볼 수 있다.

C. Search 동작 방식

Search는 주어진 조건에 따라 구현하였다.

1. 최대 자릿수

   Max digit을 의미하도록 변수명을 kmd로 설정하였다. 위에서의 결과를 바탕으로 최대 자릿수가 kmd보다 작으면 Radix sort가 더 효율적이라고 판단하여 Radix sort를 시행하도록 코드를 작성하였다.

   이때 자릿수가 6일때부터 Radix sort보다 heapsort가 빨라지는 것을 확인하여 kmd=5로 설정하였다.

2. Hash table 충돌이 일어나는 횟수

   충돌 횟수가 많으면 중복이 많이 일어난다고 판단했다. 위의 결과에 따라서 Hashtable에 중복된 수가 많아도 Radix sort와 Heapsort의 동작시간이 비슷한 것을 확인했다. 하지만 만일 최대 자릿수가 매우 커지게 되면 radix sort의 동작시간이 heap sort의 동작 시간보다 느려졌다. 따라서 이때는 heap sort를 이용하도록 코드를 작성하였다.

   Heapsort와 Quicksort를 비교하여 kcr을 결정하였다. Quick sort과정에서 partition이 제대로 이루어지지 않을 때는 겹치는 값을 가지고 있을 때이다. 이러한 과정이 각 partition이 진행된 이후 다음 partition에 또 발생하면 quick sort의 작동시간은 매우 느려질 것이다. 즉 대략 값의 절반 정도가 충돌될 때 heap sort를 사용하도록 kcr(k_collision_rate)=0.5로 설정하였다.

3. 이미 정렬되어 있는 비율

   이미 정렬되어 있는 비율을 k_sorted_ratio의 의미로 ksr이라고 하였다. 만일 미리 정렬되어 있는 value가 주어진다면 insertion sort가 가장 빠르게 동작할 것이다. Best case time complexity로 작동하기 때문에 O(n)이다. 이때 Average case time complexity는 O(n^2)이다. 이를 결정하기 어려워 어림잡아 ksr가 대략 0.8일 때를 기준으로 코드를 작성하였다.

D. Search 동작 시간과 비교

1. 동작 시간의 비교

Search를 이용하지 않았을 때를 모든 정렬을 시행하고 최적인 정렬을 찾는 방법이라고 하자. 그렇다면 가장 시간이 오래 걸리는 Bubble sort보다 분명 걸리는 시간이 클 것이다. 따라서 Bubble sort와 Search를 비교해보자.

시행	Bubble	Search
r 5000 -100 100	40	(Radix) 1
r 5000 -1000000 1000000	51	(Heap) 2
r 2000000 -1000000 1000000	45039	(Quick) 449

2. 동작 시간 분석

앞선 3가지의 시행을 통해서 우리는 다음과 같은 사실을 알 수 있다. 모든 정렬을 시행하고 이 결과를 바탕으로 최소 시간이 걸릴 알고리즘을 작성할 경우 시간 복잡도는 Bubble sort의 시간 복잡도인 O(n^2)보다 크거나 같을 것이다. 하지만 Search를 이용하면, radix, heap, insertion, quick sort중 1가지를 선택하여 시행하기 때문에 시간복잡도가 O(n)과 O(nlogn)사이 일 것이다. 따라서 Search는 근사적으로 선형적인 수행시간

이 걸리는 알고리즘임을 알 수 있다. 따라서 모든 시행에 있어서 search를 이용하는 것이 걸리는 시간을 줄일 수 있다.