1. 정렬 알고리즘 동작 방식

[Bubble Sort]
인접한 두 원소의 크기를 비교하여 더 큰 원소가 나중에 위치하도록 자리를 교환한다. 배열의 첫 원소부터 마지막 원소까지 1사이클을 수행하면, 배열 내 제일 큰 원소가 제일 마지막(A[n-1])에 위치하게 된다. 범위를 1 줄여서 마지막 원소를 제외하고 두 번째 사이클을 수행하면, 배열 내 두번째로 큰 원소가 마지막에서 2번째 자리(A[n-2])에 위치하게 된다. 이렇게 범위를 1씩 줄여가며 같은 과정을 n-1번 반복하면 정렬이 완료된다.

[Insertion Sort]
k-1번째 원소까지 정렬된 상태고, k번째 원소를 정렬해야 하는 상황이라고 하자. 이 원소의 값을 X라고 하면, k보다 작은 p에 대해 $A[p-1] \leqq X \leqq A[p]$ 를 만족하는 인덱스(p)를 찾는다. k부터 p까지의 원소를 한자리씩 큰 자리로 이동시키고, 비어있는 A[p]자리에 X를 삽입한다. 이렇게 되면 k번째 원소까지 정렬이 이루어지게 된다. 이 과정을 k의 값을 1부터 n-1까지 1씩 증가시키며 반복하면 전체 배열에 대해 정렬이 완료된다.

[Heap Sort]
Heap 자료구조를 활용한다. A[i]입장에서, A[2i], A[2i+1]은 자식 노드이고, A[i]는 이 두 원소의 부모노드이다. 모든 노드에서 부모노드의 원소가 자식노드의 원소보다 큰 관계를 유지하는 것이 바로 Heap 구조이다. 배열의 모든 원소를 대상으로 Heap 구조를 생성하면 첫 자리(A[0])에 가장 큰 원소가 위치하게 된다. 이 상태에서 A[0]과 A[n-1]의 위치를 바꾸면, 제일 마지막 자리에 가장 큰 원소가 위치한다. 마지막 원소를 제외한 나머지 원소들로 다시 Heap 구조를 만들면, A[0] 자리에 두번째로 큰 원소가 위치한다. 이를 다시 현재 상황의 마지막 원소인 A[n-2]와 위치를 바꾼다. 이렇게 점점 배열의 크기를 줄여가며 Heap 구조를 만들어 나가면 전체 배열에 대해 정렬이 완료된다.

[Merge Sort]
배열을 절반으로 분할하여 왼쪽 배열과 오른쪽 배열로 나누고, 각 배열에 대해 Merge Sort를 재귀적으로 수행한다. 계속 분할을 반복하면 결국 길이가 1인 배열들이 남게 되는데, 이때부터 merge가 수행된다. 길이가 1인 두 원소를 merge하여 길이가 2인 배열이 되고, 이를 다시 길이가 1혹은 2인 배열과 merge하여 2차 배열을 만든다. 각 merge가 수행될 때, 두 배열의 제일 첫 원소부터 마지막 원소까지 차례대로 비교하여, 병합된 배열 내에 원소 크기 순서대로 자리하게 만든다. 모든 배열이 merge되어 원래 배열의 길이를 갖게 될 때까지 이 과정을 반복하면 정렬이 완료된다.

[Quick Sort]
배열 내 한 원소를 pivot으로 삼아, 배열 내 전체 원소를 pivot과 비교하여 각각 작은 배열, 큰 배열로 분류한다. 분류된 작은 배열, 큰 배열에 각각 재귀적으로 Quick Sort를 수행한다. 난수 입력이 주어질 경우 원소의 인덱스와 크기는 관련이 없으므로 마지막 원소를 첫 pivot으로 삼아 시행한다. Merge Sort와 마찬가지로 배열의 크기가 1이 될때까지 분할을 수행하고, 이후 다시 합친다. [작은배열]+[pivot]+[큰배열] 형태로 합쳐나가면 정렬을 유지하며 합칠 수 있다. 모든 배열이 합쳐져 원래 배열의 길이를 갖게 될 때까지 과정을 반복하면 정렬이 완료된다.

[Radix Sort]
가장 낮은 자리수부터 비교 및 정렬을 수행하여 제일 큰 자리수까지 반복한다. –9부터 9까지의 수를 이름으로 갖는 Queue를 생성하고, 현재 비교를 수행하고 있는 자리의 수에 해당하는 Queue에 현재 원소를 push한다. 모든 원소를 각각에 걸맞은 Queue에 넣고 나면, -9의 Queue부터 9의 Queue까지 차례대로 모든 원소를 pop한다. 이후 자리수를 하나 올려 동일한 과정을 반복하고, 제일 큰 자리수까지 비교하면 정렬이 완료된다. 제일 작은 자리수부터 시작하고 큐를 사용하기 때문에, 현재 비교하고 있는 자리수 이하의 수는 계속 정렬이 유지된다.

2. 동작 시간 분석

[사전실험 결과 및 실험방법 기준통일 과정]
입력 배열의 길이가 10, 100일 때에는 모든 방식에서 0 혹은 1ms이 출력되어 눈에 띄는 차이가 없다. 실험은 길이 1000부터 수행한다. 또한 길이가 동일하다면 음수든 양수든 차이가 없다. 예를 들어 길이가 1000이라면, $0 \sim 1000$ 입력이나 $-500 \sim 500$ 입력이나 차이가 없었다. 본 실험에서는 –range/2 ~ range/2 구간으로 비교를 진행한다. 마지막으로 정렬 순서에 따른 결과 차이는 없다. 정렬 순서를 각각 다르게 테스트해보았을 때, 동일한 길이와 범위 내에서는 순서 상관 없이 결과 차이가 크게 나지 않았다. 따라서 정렬방법의 순서는 무시한다. 길이는 1000, … ,10^6 까지 10배씩 늘려서 총 4개이고, 범위는 10, … , 10^9까지 100배씩 늘려서 5개에 0까지 (모든 원소가 동일한 배열) 총 6개이다. 동일한 길이, 범위에서 각각 3번씩 수행하여 평균값을 조사한다. 편의상 정렬의 앞글자만을 따 B, I, H, M, Q, R로 서술하겠다.

[실험 결과] (시간, 단위 ms)

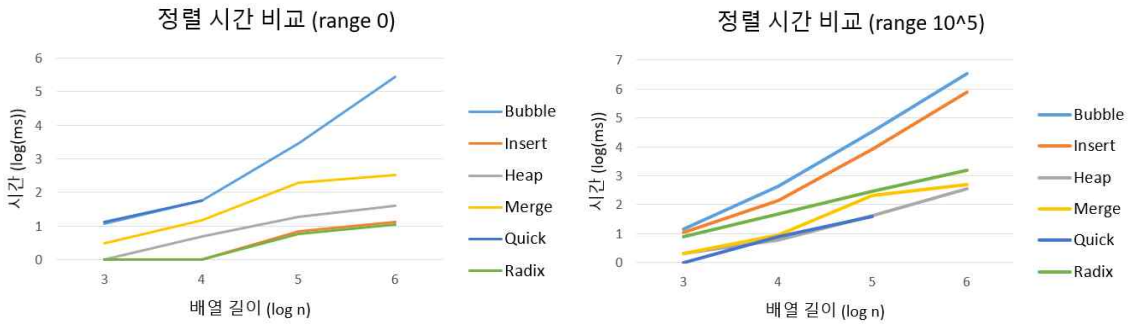
n	Range	Bubble	Insertion	Heap	Merge	Quick	Radix
10^3	0	12	0	1	3	13	1
	10^1	15	15	4	3	3	3
	10^3	16	14	3	3	2	7
	10^5	14	11	2	2	1	8
	10^7	16	13	1	2	2	16
	10^9	15	11	2	3	2	13
10^4	0	56	1	5	15	57	1
	10^1	376	123	6	9	28	12
	10^3	411	118	8	9	6	34
	10^5	424	141	6	9	8	48
	10^7	482	142	8	11	6	52
	10^9	408	137	7	10	6	59
10^5	0	2,965	7	19	195	StackOverflow	6
	10^1	31,766	7,252	37	208	444	90
	10^3	32,958	8,197	44	221	42	213
	10^5	33,589	8,526	43	214	39	297
	10^7	34,900	8,044	43	189	47	361
	10^9	34,909	8,371	45	216	42	472
10^6	0	280,233	13	41	328	StackOverflow	11
	10^1	3,318,256	723,117	219	438	StackOverflow	318
	10^3	3,367,423	792,824	319	458	StackOverflow	886
	10^5	3,329,332	806,307	354	496	StackOverflow	1,570
	10^7	3,366,078	787,276	350	524	StackOverflow	2,365
	10^9	3,334,562	780,337	307	493	StackOverflow	2,582

[범위에 따른 결과 비교]
길이가 동일할 때, 0 범위를 제외하고는 범위 증가에 따른 시간 변화는 크게 나타나지 않았다. 따라서 범위가 0일 때와 그렇지 않을때를 구분해서 보아야 한다.

1) 범위가 0인 경우 (배열의 모든 원소가 동일)
그렇지 않을 때에 비해 M은 차이가 없었다. 원소의 특성과 상관없이 항상 동일한 규모로 분리와 병합을 수행하기 때문일 것이다. Q는 오히려 범위가 작을 때 더 오래 걸렸다. 마지막 원소를 기준으로 분리하는데, 범위가 작다면 pivot이 가운데쯤이 아닌 양 극단으로 치우칠 확률이 높아 더 오래 수행하기 때문이다. B, I, H에서는 급격히 시간이 단축되었으며 특히 I에서 그 차이가 압도적이었다. R에서는 범위가 0인 것이 독특한 경우는 아니었고, 범위 증가에 따라 수행시간이 선형적으로 증가하는 경향을 보였다. 이는 자리수만큼 비교를 수행하는 R의 특성 때문이다.

2) 범위가 0이 아닌 경우
앞서 말한대로 R은 범위가 증가하면 수행시간도 증가하는 경향을 보인다. Q의 경우 길이에 비해 범위가 매우 작을 경우 수행시간이 눈에 띄게 증가하지만, 10^3이후에는 수행시간에 가시적인 차이가 없다. 나머지 네 방식의 경우 범위 증가에 따른 시간 변화는 없다고 볼 수 있다.

[길이에 따른 결과 비교]



위 도식은 배열 길이 n과 정렬 시간(ms)을 log-log scale로 나타낸 그래프다. 앞서 말한대로 길이가 0인 경우와 아닌 경우로 양상이 구분되기 때문에 두 경우를 구분하여 나타내었고, 0이 아닌 경우끼리는 큰 차이가 없어 범위를 10^5로 정하여 나타내었다. Q에서 stack이 overflow된 경우는 그래프에 표시되지 않았다. 모든 경우에서 B가 눈에 띄게 오래 걸렸음을 확인할 수 있다. n이 10배 될 때마다 되면 정렬시간은 100배 증가하였다. M에서는 범위가 0인 경우와 아닌 경우 그래프 양상이 동일하였다.

1) range 0

R, I에서 약 10ms 이내의 압도적으로 빠른 결과를 보인다. 원소 수만큼만 1번씩 비교하고, switching이 발생하지 않기 때문이다. Q의 경우 B와 비슷한 결과를 보였는데, 이는 모든 원소가 동일한 배열에 매우 취약함을 나타낸다. M은 range에 따른 차이가 없으며, H는 더 단축된 결과를 보여준다.

2) range 10^5

각 정렬의 평균적인 정렬능력을 보여주는 조건이라고 생각한다. B, I 는 $O(n^2)$, H, M, Q는 $O(n\log n)$, R은 $O(n)$이다. B, I에서는 n이 10배 증가하면 정렬시간이 약 100배 증가하는 것을 확인 할 수 있는데, 이는 $O(n^2)$임을 보여주는 결과라 할 수 있다. 그래프에서는 R가 다른 정렬 대비 오랜 시간이 걸렸지만, 한계 증가량은 타 정렬에 비해 작은 것을 확인할 수 있다. 본 실험에서는 input의 길이를 10^6으로 제한하였지만, 그 이후로 더 크기를 증가시킨다면 결국 R의 소요시간이 제일 작을 것임을 예상해볼 수 있다. 대부분의 경우에서 Q가 제일 적게 소요되었는데, 이름에 걸맞는 수행 능력을 보여주었다고 할 수 있다.