**알고리즘 과제 2 보고서**

20140132 조민호

- Experiment environment

아래와 같은

CPU : 2.3 GHz Intel Core i5

Memory : 8GB 2133 MHz LPDDR3

OS type version : macOS mojave 10.14.6

같은 환경에서 setup 해서 실험했다.

- Experiment Setup

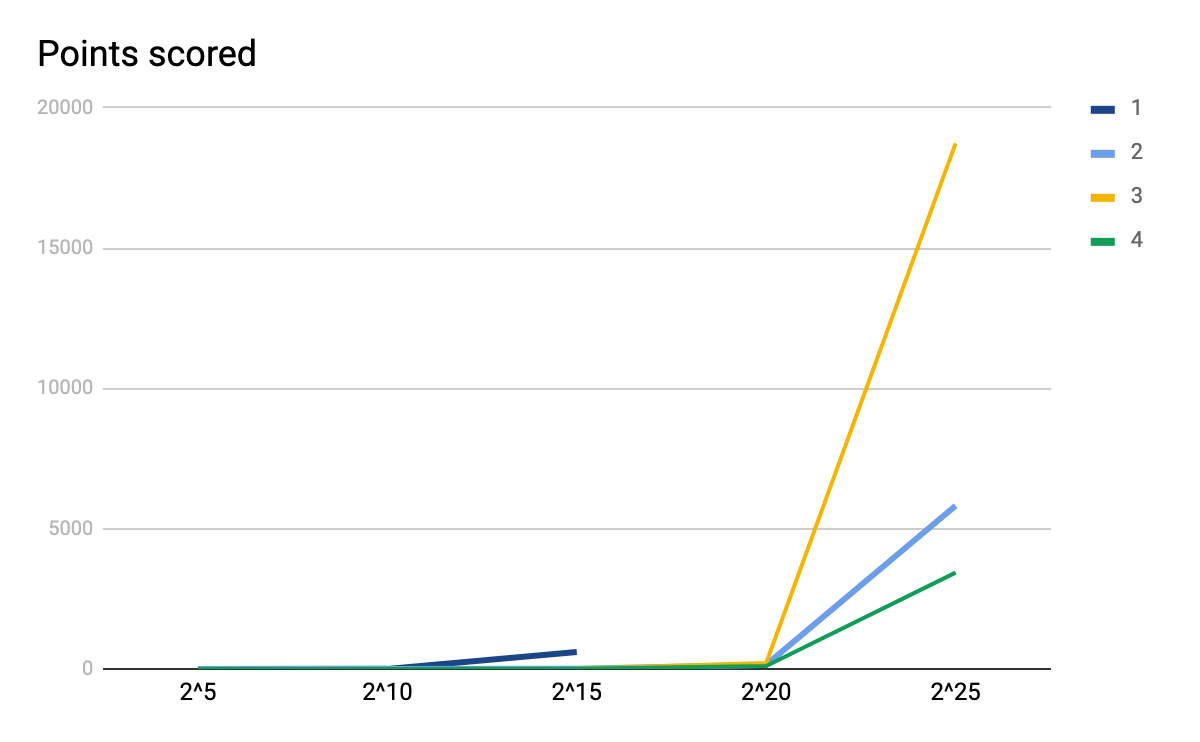
1. 기본적인 실험

input은 -1000000과 1000000사이의 정수이다. 1번 알고리즘은 insertion sort를, 2번은 quicksort를, 3번은 heap sort를, 4번은 radix sort를 활용한 sorting을 해 보았다. input별로 시간을 나타내면(단위 ms) 아래와 같았다. 데이터가 커지면 커질수록 복잡도가 폭발적으로 커진다.

먼저 random하게 만들어진 데이터의 평균을 구하여 시간을 재 보았다. 이 때 다섯번 random한 input을 만들어 실행해 보았다. 배수 변화를 확인하기 위하여 2^5의 배수로 테스트 해보았다. size는 크기가 2^5=32,2^10=1024, 2^15=32768,2^20=1048576, 2^25=33554432을 테스트 해보았다.

1. Average Time

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Insertion | quick | heap | radixplus |
| 2^5 | 0.007 | 0.011 | 0.013 | 0.026 |
| 2^10 | 0.932 | 0.146 | 0.164 | 0.157 |
| 2^15 | 602.660 | 4.419 | 6.232 | 3.858 |
| 2^20 | Nan | 144.541 | 211.501 | 105.246 |
| 2^25 | Nan | **5814.264** | **18733.381** | **3430.567** |



input size 가 커졌을 때 반영되는 시간 복잡도를 확인하고 이해할 수 있도록 실험을 해보았다. 결과적으로 input size가 점점 커지면서 quick sort와 heap sort가 같은 O(nlogn) time complexity를 가짐에도 불구하고 heap sort가 더 빨리 증가했다. average time complexity가 O(nlogn)에 가깝지만 0.011, 0.146, 4.419, 144.541, 5814.264 처럼 거의 2^5배만큼 데이터가 커질때마다 40배 가까이 데이터가 커졌다. 이 수는 2^5보다 조금 큰 수이다. 이를 통하여 5\*2^5에 가까운 수라고 추론할 수 있고, quick sort는 O(nlogn)에 가까워 짐을 추론할 수 있다. 또한 quick sort가 average case에서는 같은 time complexity를 가지는 heap sort 보다 조금 더 빨랐다.

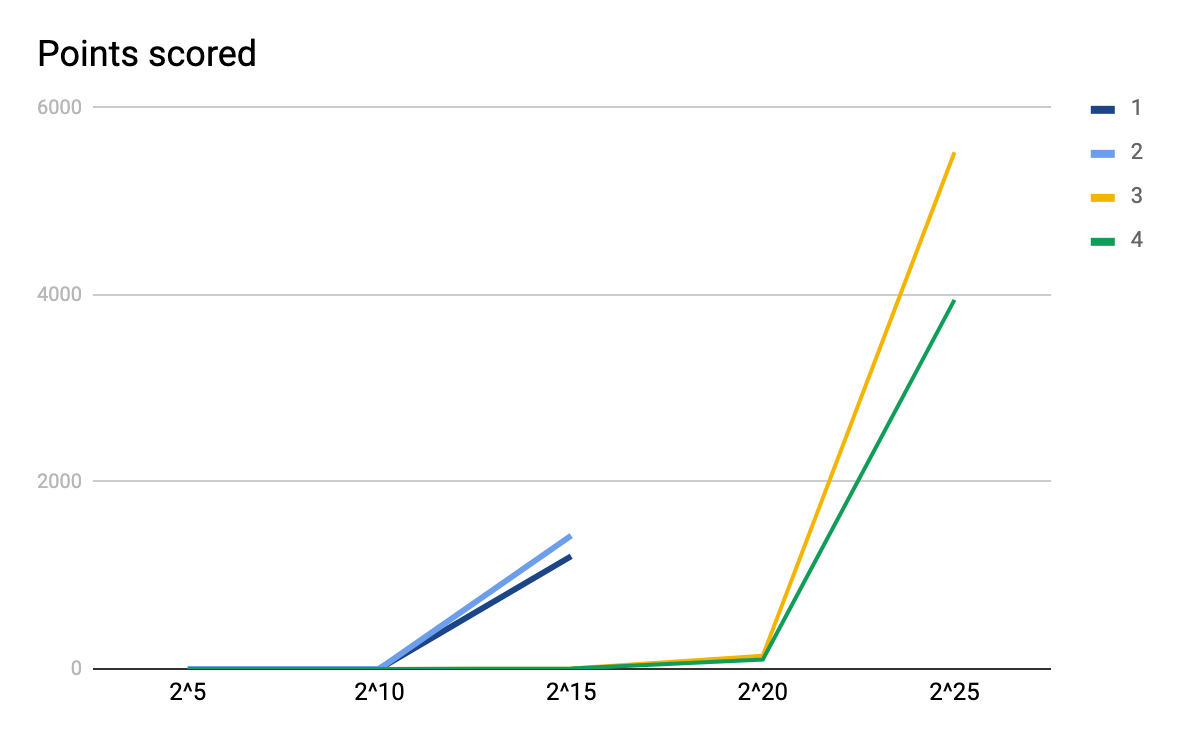
반면, heap sort는 0.021, 0.163,4.735,141.641 처럼 약 약 40배에 가깝게 데이터가 커졌다. 이 수는 2^5보다 조금 큰 수이다. 이를 통하여 5\*2^5에 가까운 수라고 추론할 수 있고, heap sort는 무관하게 O(nlogn)에 가까워 짐을 추론할 수 있다.

radixplus의 경우에는 가장 큰 max 값과 input에 의하여 time complexity가 정해진다. 1000000이 Input이므로 O(log1000000\*(n\*10)(밑 6)) 정도가 되므로 2^32씩 커질 떄 약 10배 정도 커진다고 예상할 수 있다. 비슷하게 10에 3을 곱한 30배 정도씩 커짐을 알 수 있다.

1. Worst Time

n개를 시작으로 1까지 거꾸로 되어있는 worst case를 실험해보면 아래와 같았다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Insertion | quick | heap | radixplus |
| 2^5 | 0.010 | 0.013 | 0.021 | 0.02 |
| 2^10 | 2.047 | 2.314 | 0.163 | 0.128 |
| 2^15 | 1204.708 | 1425.443 | 4.735 | 3.689 |
| 2^20 | Nan | Nan | 141.641 | 100.605 |
| 2^25 | Nan | **Nan** | **5524.480** | **3946.139** |



결과적으로 input size가 점점 커지면서 quick sort는 worst case일 떄의 O(n^2)에 가까워졌다. average time complexity가 O(nlogn)에 가깝지만 0.013 2.314 1425.443 처럼 거의 2^5배만큼 데이터가 커질때마다 1000배 가까이 데이터가 커졌다. 2^10에 가깝고 O(n^2)에 가깝다는 것을 추론할 수 있다.

반면, heap sort는 worst case 에서도 O(nlogn) time complexity를 가지므로 quick sort가 더 빨리 증가했다. 0.021, 0.163,4.735,141.641 처럼 약 약 40배에 가깝게 데이터가 커졌다. 이 수는 2^5보다 조금 큰 수이다. 이를 통하여 5\*2^5에 가까운 수라고 추론할 수 있고, heap sort는 무관하게 O(nlogn)에 가까워 짐을 추론할 수 있다.

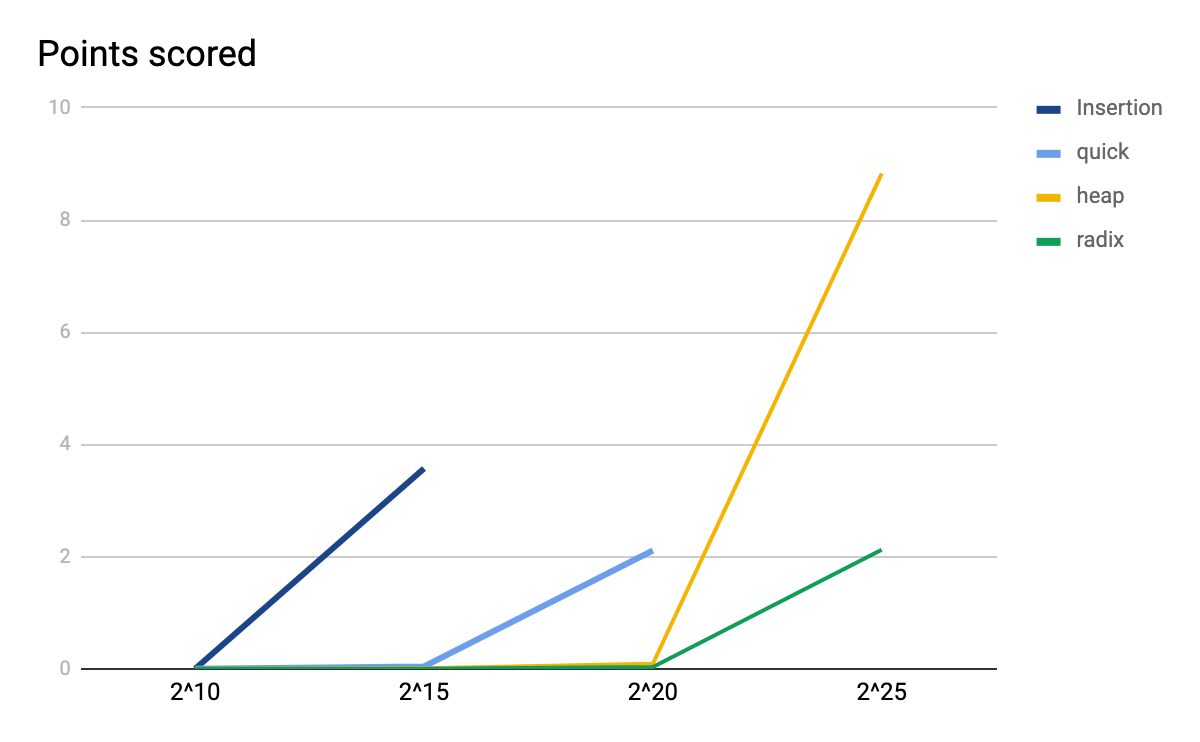
radixplus의 경우에는 worst와 상관없이 가장 큰 max 값과 input에 의하여 time complexity가 정해진다. 1000000이 Input이므로 O(log1000000\*(n\*10)(밑 6)) 정도가 되므로 2^32씩 커질 떄 약 10배 정도 커진다고 예상할 수 있다. 비슷하게 10에 3을 곱한 30배 정도씩 커짐을 알 수 있다.

추가적으로 Average Time과 Worst time을 비교해보니 조금 의아한 결과가 있었다. heap sort는 average case가 더 느린 결과가 나온 것이다. 생각해보니, average case의 Input 범위는-1000000과 1000000 사이에서 나온 랜덤한 값을 비교했었는데, worst case는 매우 작은수 부터 비교했기 때문에 comparison time이 꽤 들었을것이라고 생각했다. 특히 heap sort는 크기를 비교하는게 다른 sorting에 비해 더 많기 때문에 그렇지 않을까 추측했다. 때문에 아래 실험도 진행해 보았다.

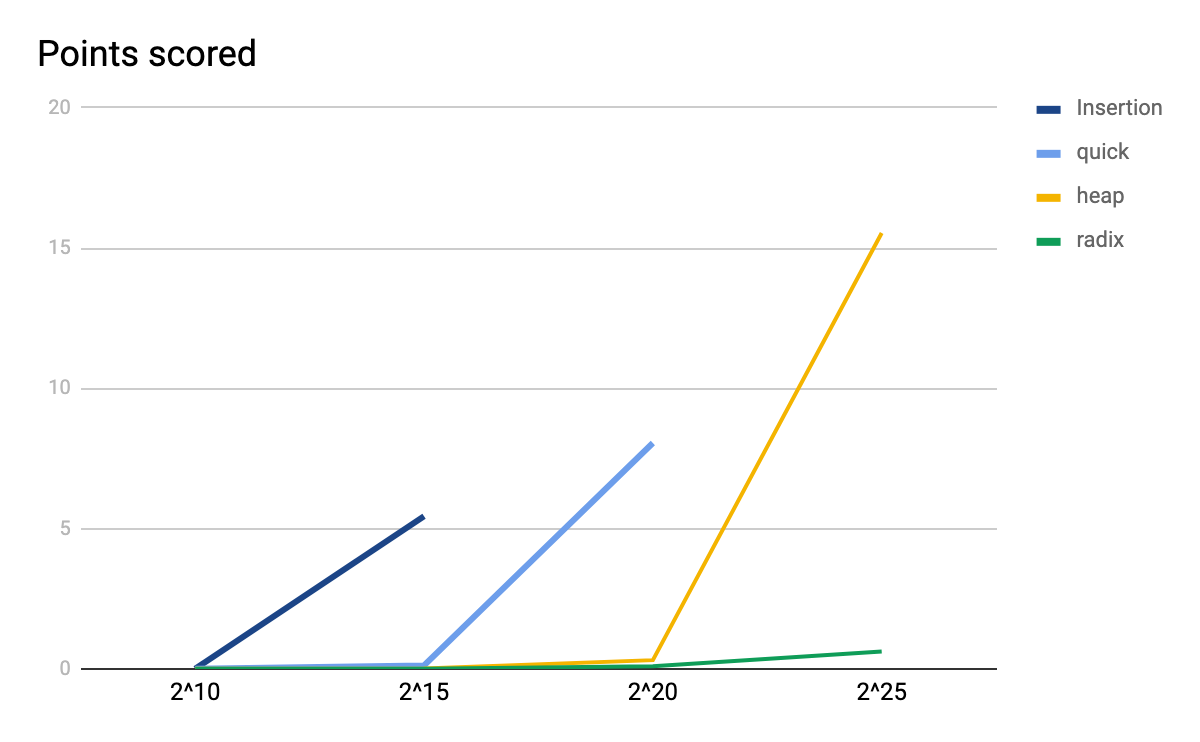
2. input 범위에 따른 속도 실험

값이 큰 것들을 비교하면 더 느려지는지 확인하기 위한 실험도 해 보았다. 간단히 average case를 -100과 100사이의 수에서 실험해 보았다. worst case 경우에도 -100과 100 사이에서 나온 random 한 순서의 데이터를 다시 역순으로 정렬해서 input으로 만들었다. 결과는 아래와 같았다. 이를 통하여 input rage가 작아질수록 더 빨리 비교함을 추론할 수 있었다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Insertion | quick | heap | radixplus |
| 2^10 | 0.002 | 0.001 | 0.001 | 0.001 |
| 2^15 | 3.573 | 0.042 | 0.010 | 0.004 |
| 2^20 | Nan | 2.110 | 0.091 | 0.035 |
| 2^25 | Nan | **Nan** | 8.829 | 2.125 |



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Insertion | quick | heap | radixplus |
| 2^10 | 0.003 | 0.001 | 0.001 | 0.005 |
| 2^15 | 5.444 | 0.124 | 0.013 | 0.008 |
| 2^20 | Nan | 8.052 | 0.312 | 0.094 |
| 2^25 | Nan | **Nan** | **15.539** | 0.624 |



3. 나의 알고리즘에 대한 설명

Input 데이터가 정수라고 가정되었으므로 radix sort가 시간을 획기적으로 줄여줄 수 있을거라 생각했다. 일반적인 정렬 알고리즘이 O(n log n) 이상의 시간 복잡도를 가지는 반면 Radix Sort 는 O(logm\*(n+b))[밑은 b, n은 개수, b는 진수, d는 가장 큰 수의 자리수)의 시간 복잡도를 지니기 때문이다. 원래 radix sort는 양수를 가정하고 만들기 때문에 음수는 양수로 만들어서 따로 계산해야 했다. 우선 입력받은 정수의 부호를 판단해서 음수이면 neg\_list에 저장하고 양수면 pos\_list에 저장했다. 이 떄 음수일때는 -1을 곱해 양수로 만든다.

이후 neg\_list를 정렬하고, pos\_list를 radix sort로 정렬하면 neg\_list를 거꾸로 돌리면서 모든요소에 다시 -1을곱해준다. 거꾸로 돌리는 이유는 절대값이 작은 값이 음수일때는 더 큰 값이기 때문이다. 이후 그 뒤에 pos\_list를 붙여주면 제대로 정렬이 된다.

radixplus의 경우에는 worst와 상관없이 가장 큰 max 값과 input에 의하여 time complexity가 정해진다. 1000000이 Input이므로 O(log1000000\*(n\*10)(밑 6)) 정도가 되므로 2^32씩 커질 떄 약 10배 정도 커진다고 예상할 수 있다. 비슷하게 10에 3을 곱한 30배 정도씩 커짐을 알 수 있다.

- Your comments on the experience

이번 실험을 통하여 quicksort와 heap sort가 같은 time complexity를 가지지만 average case에서는 quicksort가, worst case에서는 heap sort가 훨씬 빠름을 직관적으로 파악할 수 있었다.

또한 input range가 커지면서 비교 속도가 느려지고 이는 time complexity에 영향을 미침을 추론할 수 있는 시간이었다. 교수님께서 time complexity에서 comparison을 따로 계산할 필요가 있을 떄가 있다고 하셨는데 이런 때이지 않을까 생각했다.