Report of MP2 : Master of Sorting

알고리즘 설계와 분석 2분반

20211530 김희진

0. 실험 환경과 실험내용

- Experiment environment
  - CPU : Intel Core i5-1035G4 프로세서 (4코어 SMT, **최대 3.7GHz**)
  - RAM : 8GB, DDR4, 3200MHz
  - OS type : Windows11
- Experiment setup

  size가 10, 1000, 10000, 100000, 1000000인 array에 대해 4개의 sort algorithm을 이용해 increasing order로 sort한 후, 걸린 시간에 대해 비교해 볼 것이다.

  random한 수들로 이루어진 random list와 non-increasing list에 대해서 실험을 해줄 것이다. Random list를 만드는 과정은 아래와 같다.

```
./testcasegen 10 9587452 10
```

```
./testcasegen 100 9587452 10
```

```
./testcasegen 1000 9587452 10
./testcasegen 10000 9587452 10
./testcasegen 100000 9587452 10
./testcasegen 1000000 9587452 10
```

  Non-increasing list는 원하는 array의 크기 n에 대해 n부터 1까지 decreasing order로 정렬되어 있는 파일을 생성하는 testcasegen2.c를 이용해 만들었다. Non-increasing list를 만드는 과정은 아래와 같다.

```
./testcasegen2 10
./testcasegen2 100
./testcasegen2 1000
./testcasegen2 10000
./testcasegen2 100000
./testcasegen2 1000000
```

1. Compare performance of the four algorithms

   a. Insertion sort

      1) Random list

         ① 10

```
input_000000010_009587452_000000010.txt
1
10
0.000003
-7296807 -4234669 -2537109 -1546785 -1518916 -683298 -423537 916845 2746602 6
```

② 100

```
input_000000100_009587452_000000010.txt
1
100
0.000026
-9562209 -9489736 -9327920 -9121490 -8789926 -8062860 -7788054 -7491314 -740
```

③ 1000

```
input_000001000_009587452_000000010.txt
1
1000
0.002334
-9562209 -9555899 -9531845 -9519090 -9502064 -9498994 -9489736 -9457433 -945
```

④ 10000

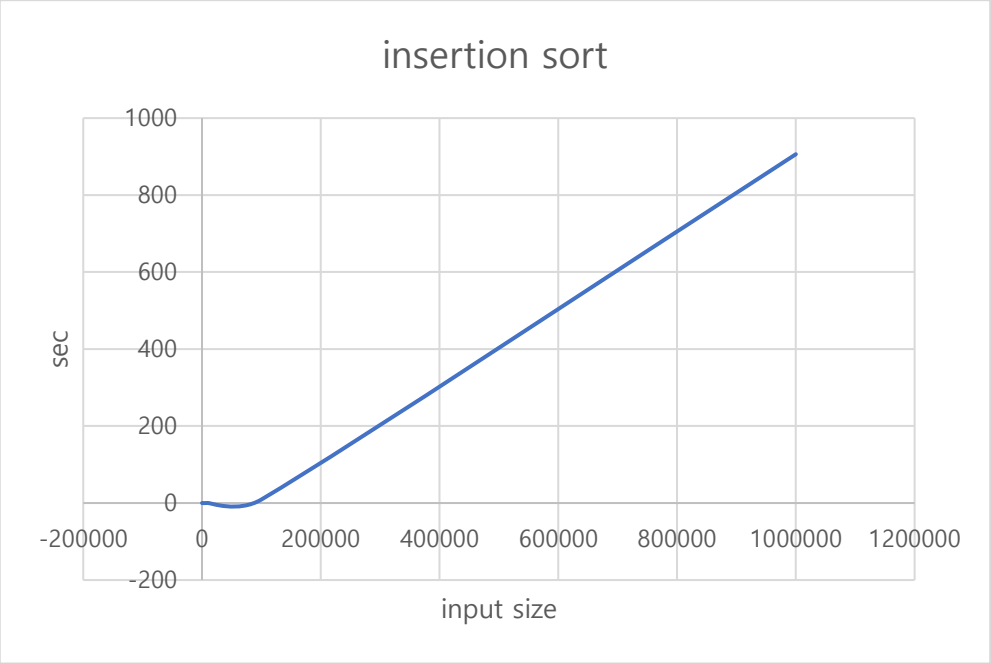
```
input_000010000_009587452_000000010.txt
1
10000
0.133334
-9584741 -9584233 -9583714 -9581645 -9580231 -9578750 -9577863 -9575347
-9575054 -9574636 -9563810 -9562220 -9562209 -9560684 -9559269 -9559031
```

⑤ 100000

```
input_000100000_009587452_000000010.txt
1
100000
9.176849
-9587179 -9587132 -9586897 -9586850 -9586723 -9586262 -9585919 -9585822
-9585734 -9585391 -9585160 -9585154 -9585052 -9584955 -9584933 -9584888
-9584741 -9584625 -9584597 -9584537 -9584443 -9584365 -9584280 -9584233
```

⑥ 100000

```
input_001000000_009587452_000000010.txt
1
1000000
906.193460
-9587451 -9587432 -9587411 -9587392 -9587370 -9587364 -9587347 -9587337 -9587
```



2) List in which the numbers are sorted in non-increasing order

① 10

```
input_10.txt
1
10
0.000004
1 2 3 4 5 6 7 8 9 10
```

② 100

```
input_100.txt
1
100
0.000052
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 1
```

③ 1000

```
input_1000.txt
1
1000
0.004664
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
```

④ 10000

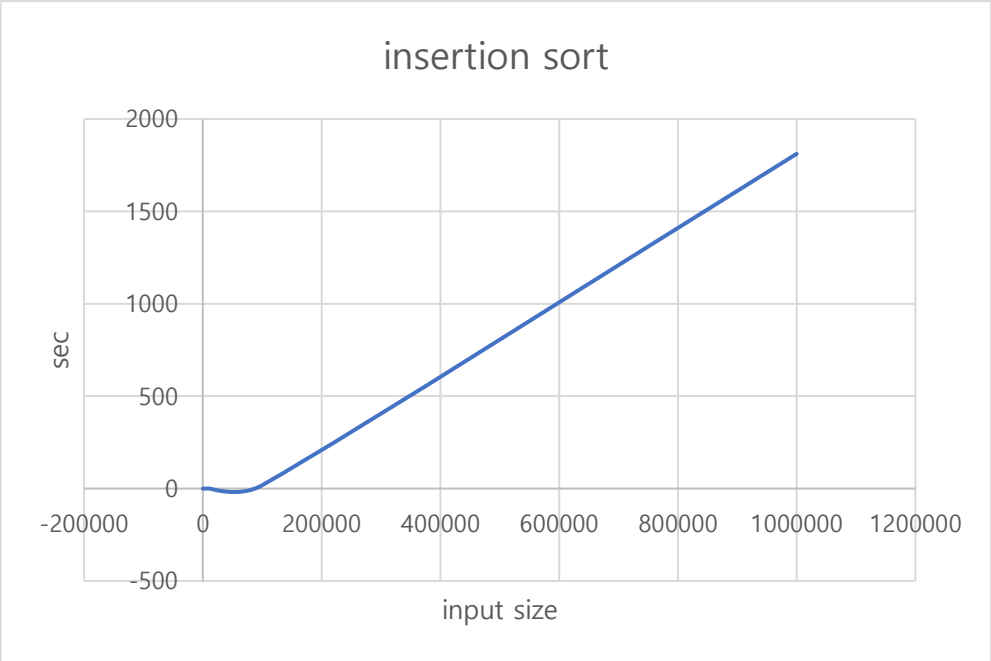
```
input_10000.txt
1
10000
0.216328
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53
54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78
79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102
```

⑤ 100000

```
input_100000.txt
1
100000
18.160590
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53
54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78
79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102
103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121
```

⑥ 1000000

```
input_1000000.txt
1
1000000
1811.592421
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 2
```



두 경우다 거의 2차함수와 비슷한 기울기로 시간이 증가하고 있지만 decreasing list일 때 훨씬 sort에 시간이 많이 듦을 확인할 수 있다.

b. Quick sort

1) Random list

① 10

```
input_000000010_009587452_000000010.txt
2
10
0.000004
-7296807 -4234669 -2537109 -1546785 -1518916 -683298 -423537 916845 2746602 6
```

② 100

```
input_000000100_009587452_000000010.txt
2
100
0.000013
-9562209 -9489736 -7296807 -7186252 -4234669 -423537 916845 -2505215 -154678
```

③ 1000

```
input_000001000_009587452_000000010.txt
2
1000
0.000106
-9562209 -9489736 -7296807 -7186252 -4234669 -423537 916845 -2505215 -154678
```

④ 10000

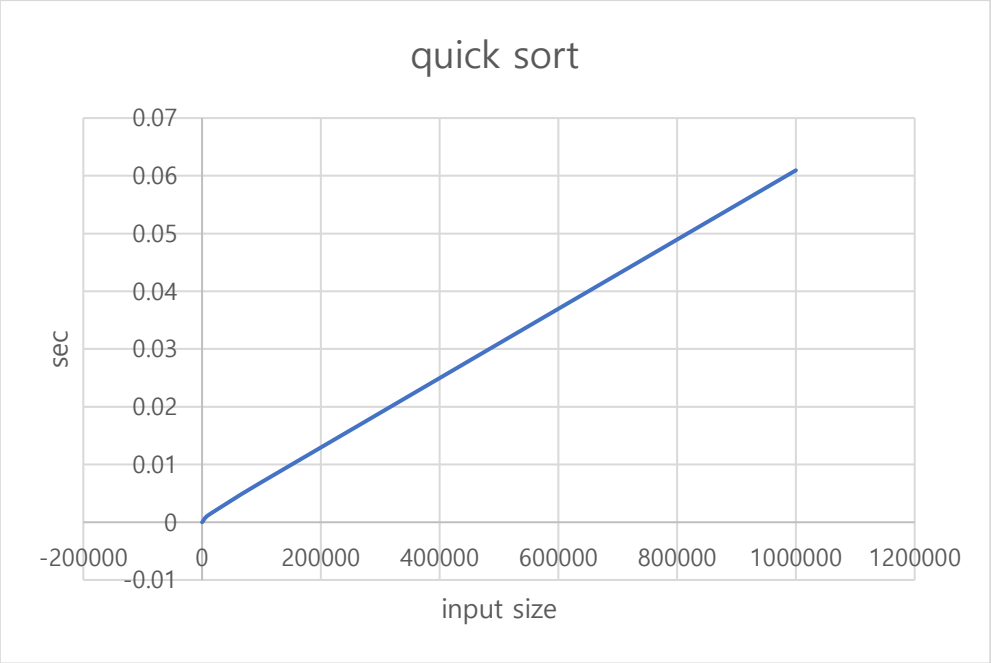
```
input_000010000_009587452_000000010.txt
2
10000
0.001191
-9584741 -9584233 -9580231 -9563810 -9562209 -9489736 -7186252 -4234669
-2505215 -1546785 6952747 -2009460 712185 6467487 7109204 8519609 -3041654
-423537 3126027 -9327920 -9121490 2249169 4384775 7976356 -3645080 -8789926
```

⑤ 100000

```
input_000100000_009587452_000000010.txt
2
100000
0.006939
-9587179 -9586723 -9585052 -9584741 -9584233 -9580231 -9563810 -9562209
-9489736 -7186252 -2505215 -2009460 712185 6952747 7109204 8519609 -3041654
-1546785 3126027 -9327920 -9121490 2249169 4384775 7976356 -3645080
-8789926 -6551612 -4698617 -3022844 -8062860 -7116013 -615789 498731 620105
```

⑥ 1000000

```
input_001000000_009587452_000000010.txt
2
1000000
0.060950
-9587451 -9587411 -9587179 -9586723 -9585052 -9584741 -9584233 -9580231
-9563810 -9562209 -9489736 -7186252 -2009460 712185 7109204 8519609
-3041654 -2505215 3126027 -9327920 -9121490 2249169 4384775 7976356
-3645080 -8789926 -6551612 -4698617 -3022844 -8062860 -7116013 -615789
498731 620105 2894976 8288365 9290712 -5428155 -3148943 -2499860 924655
```



2) List in which the numbers are sorted in non-increasing order

① 10

```
input_10.txt
2
10
0.000005
1 2 3 4 5 6 7 8 9 10
```

② 100

```
input_100.txt
2
100
0.000039
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 2
```

③ 1000

```
input_1000.txt
2
1000
0.002951
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
```

④ 10000

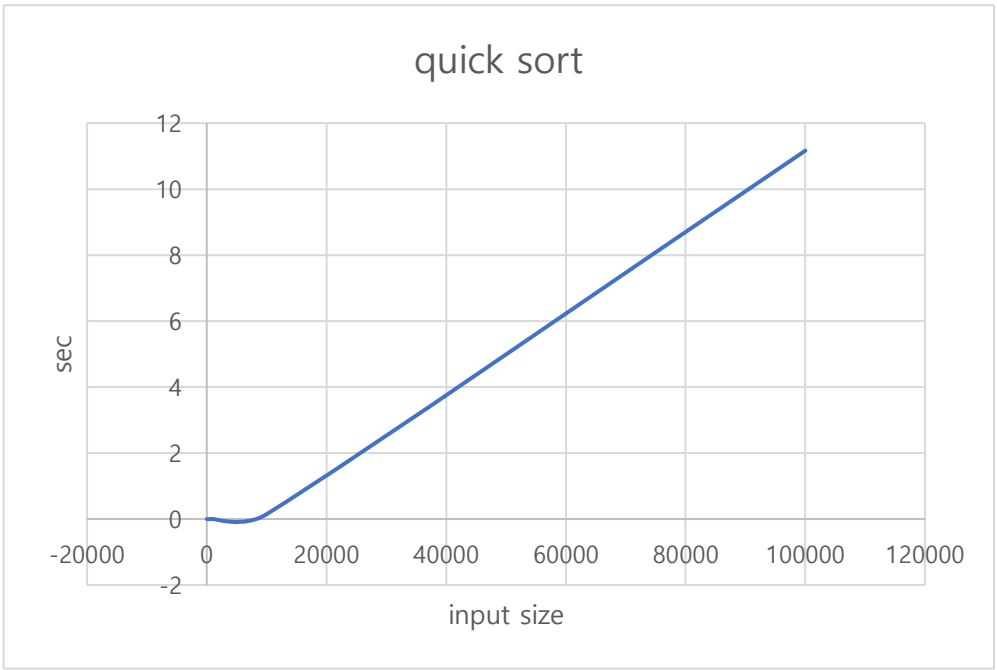
```
input_10000.txt
2
10000
0.147308
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53
54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78
79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102
103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121
```

⑤ 100000

```
input_100000.txt
2
100000
11.166097
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53
54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78
79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102
103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121
122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140
```

⑥ 1000000

overflow문제로 결과를 확인할 수 없었다.



Random list에 대해서는 일차함수 모양의 기울기로, decreasing list에 대해서는 2차함수 모양의 기울기로 시간이 증가함을 알 수 있다. Insertion sort와 마찬가지로 decreasing list에 대해서 sort할 때 시간이 훨씬 많이 걸림을 알 수 있다.

c. Merge sort

1) Random list

① 10

```
input_000000010_009587452_000000010.txt
3
10
0.000006
-7296807 -4234669 -2537109 -1546785 -1518916 -683298 -423537 916845 2746602 6
```

② 100

```
input_000000100_009587452_000000010.txt
3
100
0.000027
-9562209 -9489736 -9327920 -9121490 -8789926 -8062860 -7788054 -7491314 -740
```

③ 1000

```
input_000001000_009587452_000000010.txt
3
1000
0.000324
-9562209 -9555899 -9531845 -9519090 -9502064 -9498994 -9489736 -9457433 -945
```

④ 10000

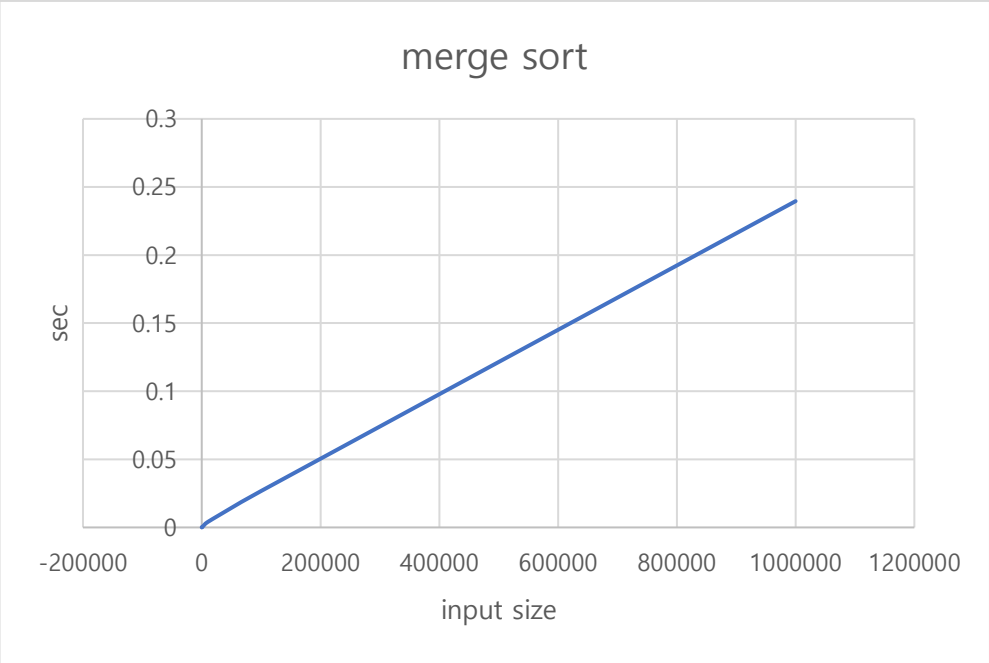
```
input_000010000_009587452_000000010.txt
3
10000
0.003980
-9584741 -9584233 -9583714 -9581645 -9580231 -9578750 -9577863 -9575347
-9575054 -9574636 -9563810 -9562220 -9562209 -9560684 -9559269 -9559031
-9558991 -9558180 -9555899 -9555348 -9554531 -9540903 -9538303 -9536275
```

⑤ 100000

```
input_000100000_009587452_000000010.txt
3
100000
0.026768
-9587179 -9587132 -9586897 -9586850 -9586723 -9586262 -9585919 -9585822
-9585734 -9585391 -9585160 -9585154 -9585052 -9584955 -9584933 -9584888
-9584741 -9584625 -9584597 -9584537 -9584443 -9584365 -9584280 -9584233
```

⑥ 100000

```
input_001000000_009587452_000000010.txt
3
1000000
0.239584
-9587451 -9587432 -9587411 -9587392 -9587370 -9587364 -9587347 -9587337
-9587313 -9587280 -9587263 -9587227 -9587225 -9587190 -9587189 -9587188
-9587179 -9587155 -9587155 -9587132 -9587049 -9587047 -9587045 -9587012
-9587005 -9586945 -9586933 -9586913 -9586910 -9586897 -9586897 -9586883
```



2) List in which the numbers are sorted in non-increasing order

① 10

```
input_10.txt
3
10
0.000005
1 2 3 4 5 6 7 8 9 10
```

② 100

```
input_100.txt
3
100
0.000019
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 2
```

③ 1000

input_1000.txt
3
1000
0.000178
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 2

④ 10000

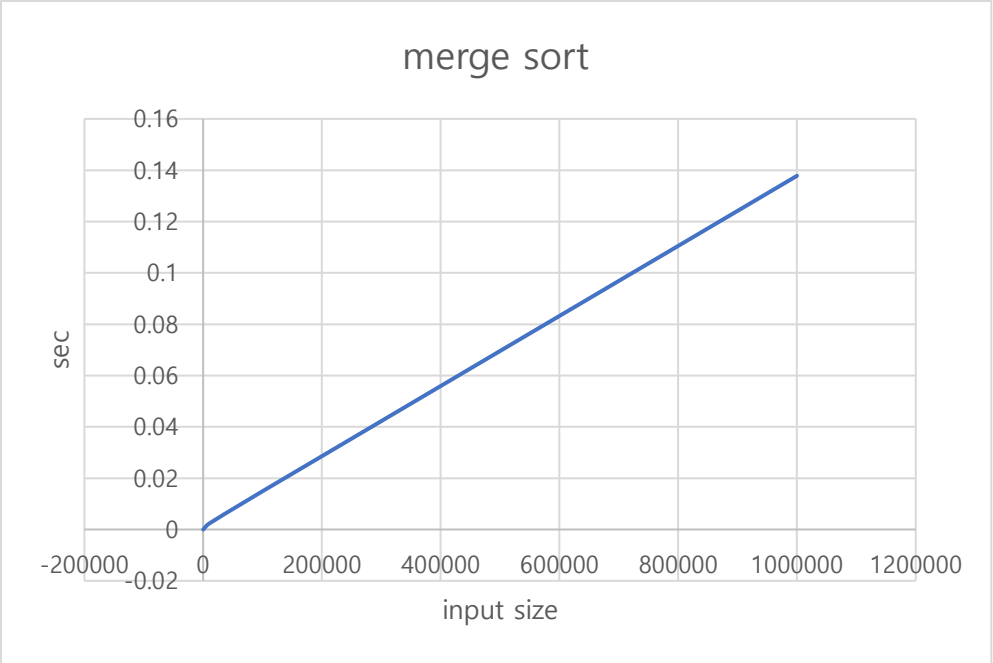
input_10000.txt
3
10000
0.002283
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53
54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78
79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102
103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121
122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140
141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159

⑤ 100000

input_100000.txt
3
100000
0.014908
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53
54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78
79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102
103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121

⑥ 100000

input_1000000.txt
3
1000000
0.137816
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53
54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78
79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102
103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121



두 경우다 일차함수와 비슷한 모양의 기울기로 시간이 증가하고 있고 random list와 decreasing list에 대해서 드는 시간이 비슷하다.

d. Hj sort(sort I think will run fastest)

1) Random list

① 10

```
input_000000010_009587452_000000010.txt
4
10
0.000003
-7296807 -4234669 -2537109 -1546785 -1518916 -683298 -423537 916845 2746602 6
```

② 100

```
input_000000100_009587452_000000010.txt
4
100
0.000014
-9562209 -9489736 -9327920 -9121490 -8789926 -7296807 -7186252 -6551612 -469
```

③ 1000

```
input_000001000_009587452_000000010.txt
4
1000
0.000121
-9562209 -9489736 -9327920 -9121490 -8789926 -7296807 -7186252 -6551612 -469
```

④ 10000

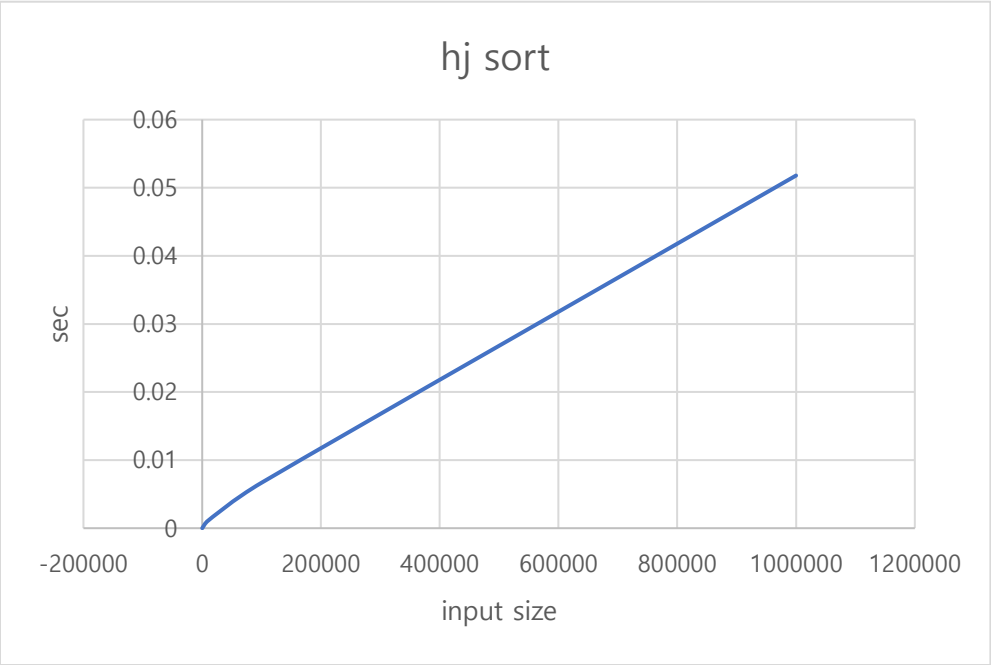
```
input_000010000_009587452_000000010.txt
4
10000
0.001111
-9584741 -9584233 -9580231 -9563810 -9562209 -9489736 -9327920 -9121490
-8789926 -7186252 -6551612 -4698617 -4234669 -3645080 -3041654 -3022844
```

⑤ 100000

```
input_000100000_009587452_000000010.txt
4
100000
0.006681
-9587179 -9586723 -9585052 -9584741 -9584233 -9580231 -9563810 -9562209
-9489736 -9327920 -9121490 -8789926 -7186252 -6551612 -4698617 -3645080
-3041654 -3022844 -2505215 -2009460 -1546785 712185 2249169 3126027 4384775
6952747 7109204 7976356 8519609 -8062860 -7116013 -5428155 -3148943
```

⑥ 1000000

```
input_001000000_009587452_000000010.txt
4
1000000
0.051789
-9587451 -9587411 -9587179 -9586723 -9585052 -9584741 -9584233 -9580231
-9563810 -9562209 -9489736 -9327920 -9121490 -8789926 -7186252 -6551612
-4698617 -3645080 -3041654 -3022844 -2505215 -2009460 712185 2249169
3126027 4384775 7109204 7976356 8519609 -8062860 -7116013 -5428155 -3148943
-2499860 -615789 498731 620105 924655 2894976 8288365 9290712 -7788054
-7491314 -5837931 -5145079 -4477004 -4347552 -320917 -150183 109349 250645
```



2) List in which the numbers are sorted in non-increasing order

① 10

```
input_10.txt
4
10
0.000003
1 2 3 4 5 6 7 8 9 10
```

② 100

```
input_100.txt
4
100
0.000021
```

③ 1000

```
input_1000.txt
4
1000
0.000545
```

④ 10000

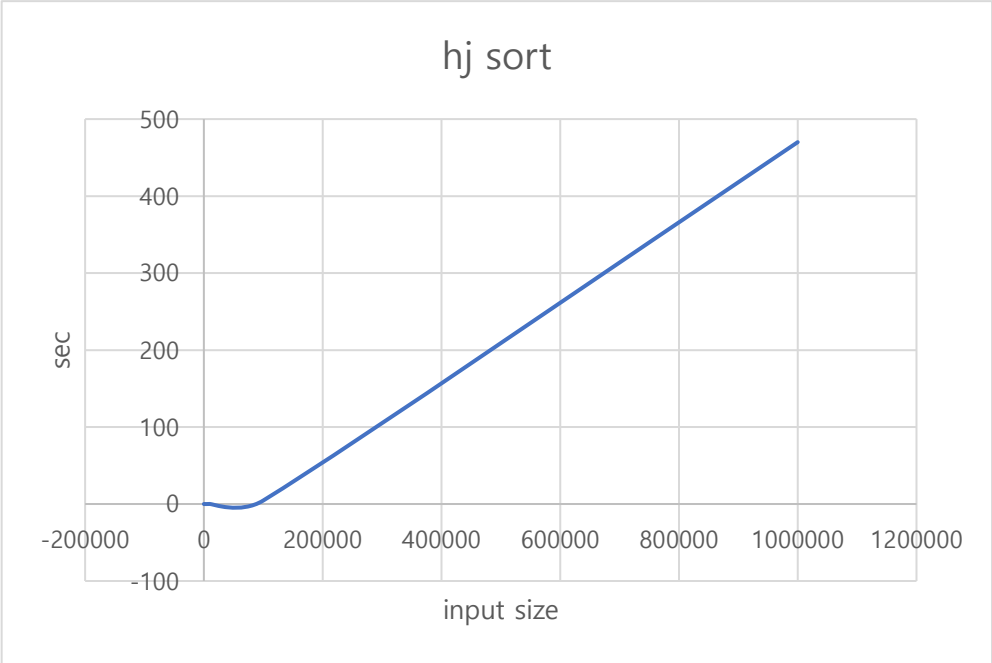
```
input_10000.txt
4
10000
0.065129
```

⑤ 100000

```
input_100000.txt
4
100000
4.539388
```

⑥ 1000000

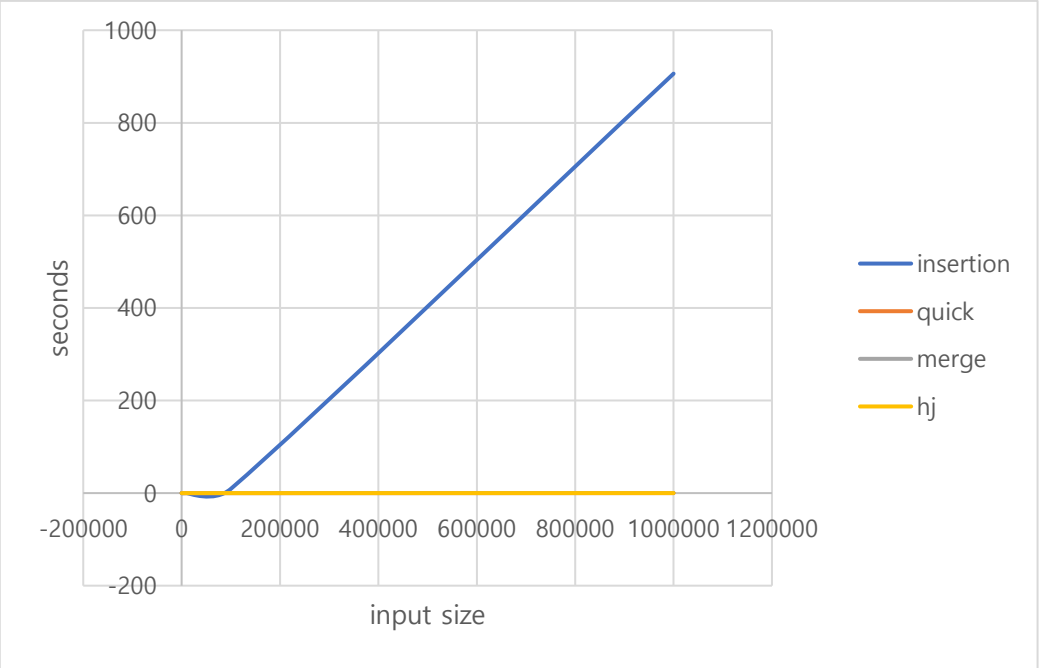
```
input_1000000.txt
4
1000000
470.353822
```



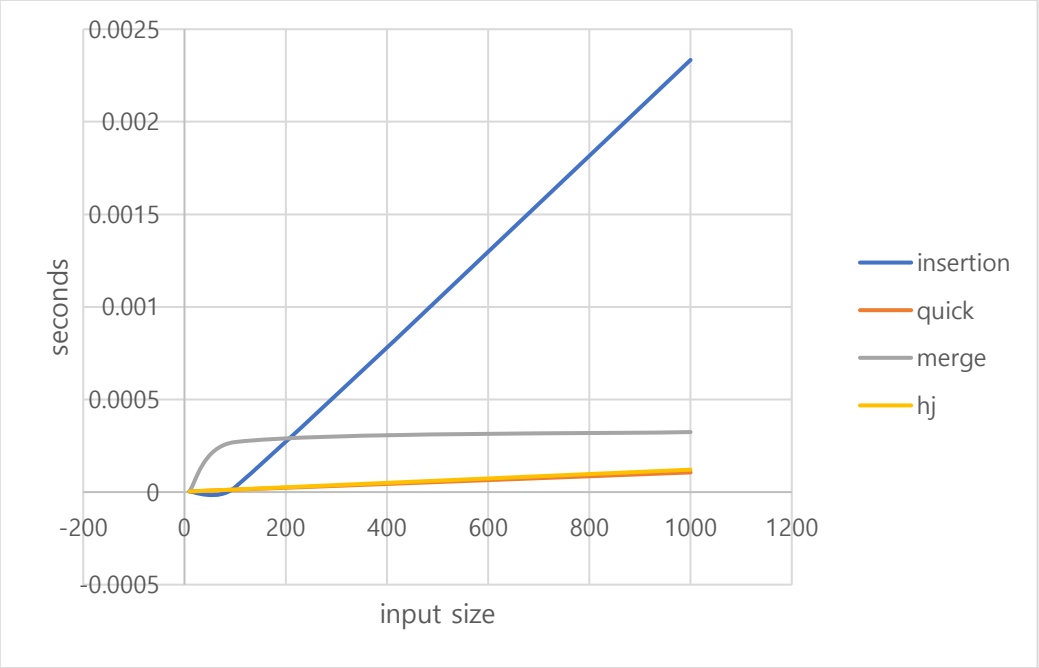
Random list에 대해서는 일차함수 모양으로, decreasing list에 대해서는 이차함수보다는 완만한 모양으로 기울기가 증가하는 형태를 띈다. 역시나, decreasing list에서 sort에 시간이 훨씬 많이 걸린다.

e. Comparation

1) Random list

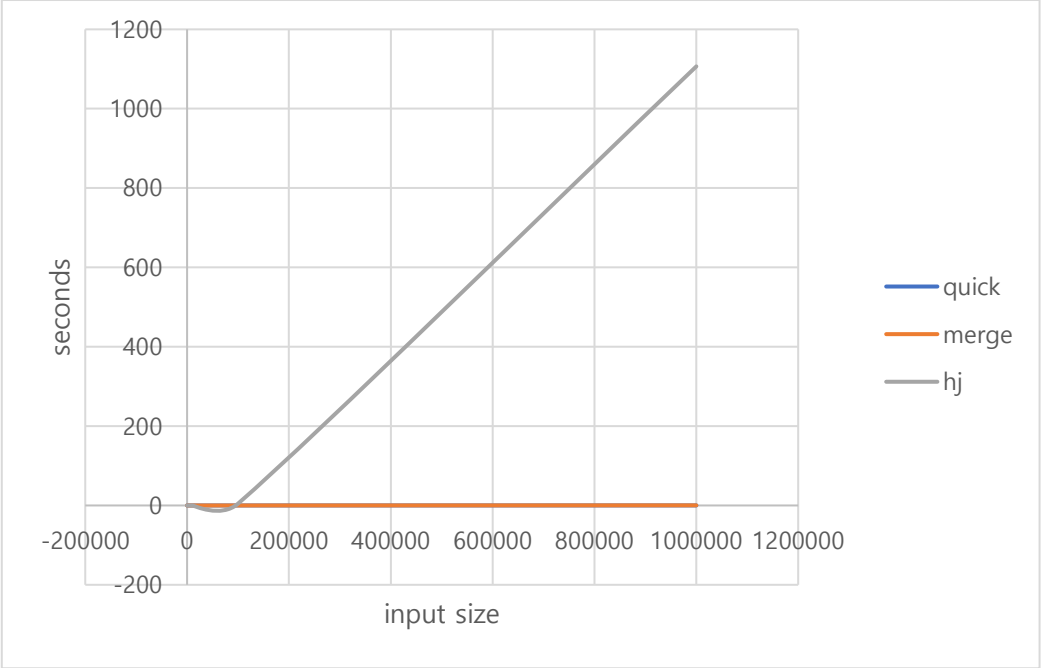


Insertion sort가 다른 3가지의 sort에 비해 input size가 커질수록 시간이 빠른 속도로 증가하고 있음을 확인할 수 있다. 이 그래프로는 insertion sort를 제외한 함수들을 비교하기 어렵기 때문에 input size를 1000까지 자른 그래프를 살펴보자.



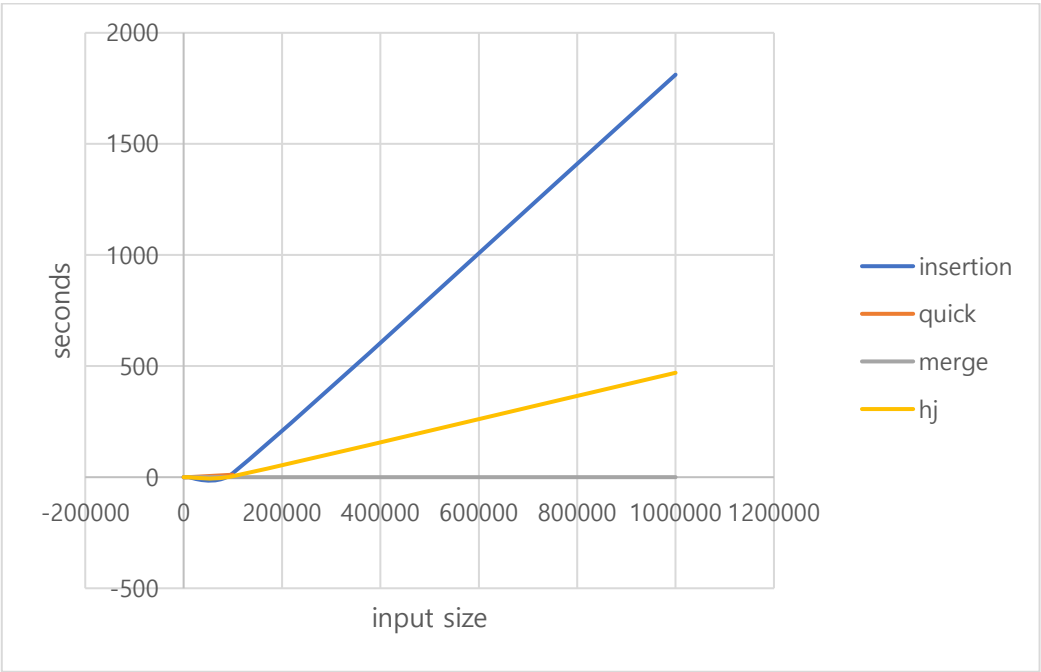
Insertion sort는 input size가 작을 때는 다른 3가지의 sort보다 더 빠른 혹은 비슷한 시간에 정렬되고 있다. Input size가 약 200이하까지 insertion sort는 merge sort보다 빠르며 약 100이하일 때는 quick sort나 hj sort보다 빠르다. 하지만 insertion sort는 input size가 커질수록 가파르게 시간이 증가하고 있다. 또한 merge sort는 hj sort와 quick sort보다 확연히 시간이 오래 걸렸으며 quick sort와

hj sort는 비슷한 시간내에 정렬되고 있다.

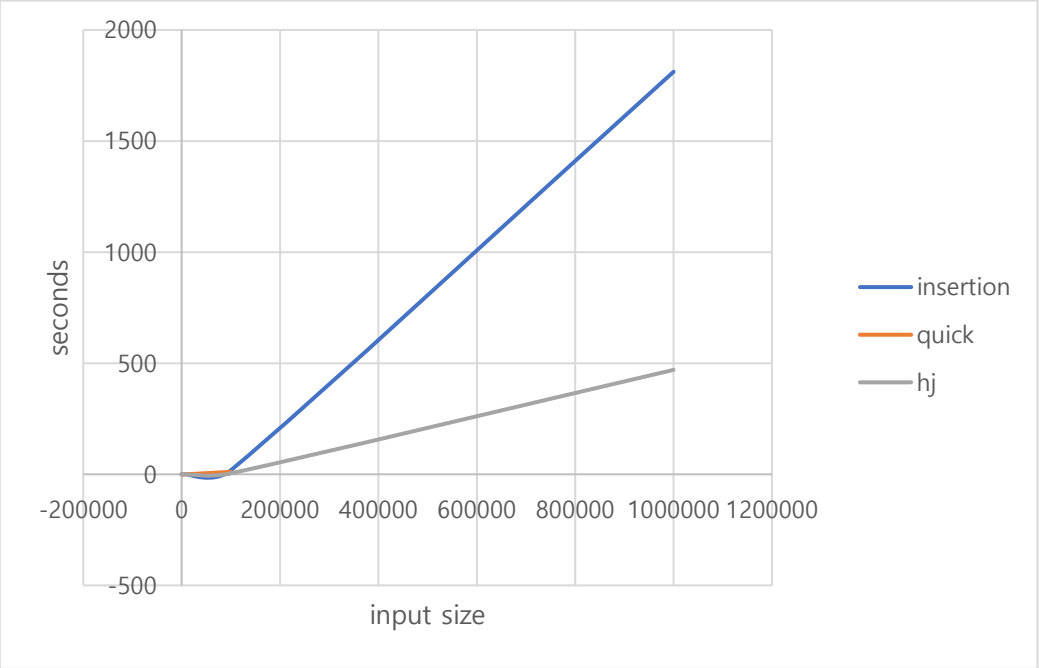


Insertion sort를 제외하고 3가지 함수에 대해 10부터 1000000크기까지의 정렬을 살펴보았을 때 merge sort는 quick, hj sort보다 더 빠른 속도로 sorting time이 증가하고 있다. quick sort와 hj sort는 비슷한 속도를 가지지만 hj sort가 조금 더 빠른 시간내 정렬하고 있음을 확인할 수 있다. 시간이 증가하는 속도는 hj sort가 더 완만하게 증가하고 있으므로 input size가 더 커진다면 quick sort와 hj sort의 차이가 더 두드러질 것으로 예상된다.

2) List in which the numbers are sorted in non-increasing order



다른 세가지 sort방식과 비교했을 때 decreasing list에 대해 merge sort가 확실히 좋은 시간성능을 가짐을 확인할 수 있다. 또한 hj sort도 insertion sort에 비해 적은 시간에 sort하고 있음을 확인할 수 있다.



Merge sort를 제외한 3가지 sort를 살펴보자. 이때 약 80~90정도까지는 4가지 함수가 비슷한 시간을 가지나 그 이후부터는 함수가 증가하는 모양이 확연히 달라진다. Quick sort는 decreasing order list에서 input size가 1000000일 때 over flow가 나서 확인하지 못했지만 다른 input size에서 insertion과 hj sort 사이의 값을 가졌으므로 1000000의 크기일 때도 그럴 것으로 예상된다. 100000일 때 세가지의 sort중에서는 hj sort가 확실히 시간적인 면에서 좋은 성능을 가졌으며 다음으로 quick sort, insertion sort순서이다.

2. Describe how you designed algorithm 4

위 그래프(분석결과)에서 알 수 있듯, insertion sort는 간단한 sort방식이기에 input size가 작을 때 다른 sort 방식보다 시간이 적게 걸린다. 그래서 quick sort와 insertion sort를 섞어서 정렬해야 하는 size가 30이하일 때는 insertion sort로 sort를 해주고 31이상일 때는 quick sort로 sort를 해주는 아이디어를 생각했다.

```
void hj_sort(int left, int right)
{
    while (left < right)
    {
        if (right - left + 1 < 30)
        {
            insertion_sort(left, right);
            break;
        }
        else
        {

            int pivot = partition(left, right);
            if (pivot - left < right - pivot)
            {
                hj_sort(left, pivot - 1);
                left = pivot + 1;
            }
            else
            {
                hj_sort(pivot + 1, right);
                right = pivot - 1;
            }
        }
    }
}
```

예상과 같이 이렇게 만든 알고리즘은 기존의 quick sort보다 빠른 시간에 sort를 할 수 있었으며, random list에 대해서는 merge sort보다도 빠른 시간에 정렬하고 있음을 확인했다.

하지만 decreasing ordered list에서 너무 좋지 않은 성능을 보아서 중간 index의 수를 pivot으로 설정해주는 최적화를 진행했다.

```
void hj_sort(int left, int right)
{
    while (left < right)
    {
        if (right - left + 1 < 30)
        {
            insertion_sort(left, right);
            break;
        }
        else
        {
            SWAP(arr[left], arr[(left+right)/2]);
            int pivot = partition(left, right);
            if (pivot - left < right - pivot)
            {
                hj_sort(left, pivot - 1);
                left = pivot + 1;
            }
            else
            {
                hj_sort(pivot + 1, right);
                right = pivot - 1;
            }
        }
    }
}
```