Cüneyt EREM 21202398 Cs-202

1)

Array: [25,9,3,7,11,0,2,20,28,5,16]

a-) Insertion sort

25,9,3,7,11,0,2,20,28,5,16

9,25,3,7,11,0,2,20,28,5,16

3,9,25,7,11,0,2,20,28,5,16

3,7,9,25,11,0,2,20,28,5,16

3,7,9,11,25,0,2,20,28,5,16

0,3,7,9,11,25,2,20,28,5,16

0,2,3,7,9,11,25,20,28,5,16

0,2,3,7,9,11,20,25,28,5,16

0,2,3,7,9,11,20,25,28,5,16

0,2,3,5,7,9,11,20,25,28,16

0,2,3,5,7,9,11,16,20,25,28

b-) Selection Sort

25,9,3,7,11,0,2,20,28,5,16

0,9,3,7,11,25,2,20,28,5,16

0,2,3,7,11,25,9,20,28,5,16

0,2,3,7,11,25,9,20,28,5,16

0,2,3,5,11,25,9,20,28,7,16

0,2,3,5,7,25,9,20,28,11,16

0,2,3,5,7,9,25,20,28,11,16

0,2,3,5,7,9,11,20,28,25,16

0,2,3,5,7,9,11,16,28,25,20

0,2,3,5,7,9,11,16,20,25,28

0,2,3,5,7,9,11,16,20,25,28

0,2,3,5,7,9,11,16,20,25,28

c-) Buble Sort

25,9,3,7,11,0,2,20,28,5,16 -> **9,25**,3,7,11,0,2,20,28,5,16

9,**25,3**,7,11,0,2,20,28,5,16 -> 9,**3,25**,7,11,0,2,20,28,5,16

9,3,7,**25,11**,0,2,20,28,5,16 -> 9,3,7,**11,25**,0,2,20,28,5,16

……..

9,3,7,11,0,2,20,**25,28**,5,16 **->** 9,3,7,11,0,2,20,**25,28**,5,16

9,3,7,11,0,2,20,25,**28,5**,16 -> 9,3,7,11,0,2,20,25,**5,28**,16

9,3,7,11,0,2,20,25,5,**28,16** -> 9,3,7,11,0,2,20,25,5,**16,28**

Second step;

9,3,7,11,0,2,20,25,5,16,28 -> **3,9**,7,11,0,2,20,25,5,16,28

3,**9,7**,11,0,2,20,25,5,16,28 -> 3,**7,9**,11,0,2,20,25,5,16,28

3,7,**9,11**,0,2,20,25,5,16,28 -> 3,7,**9,11**,0,2,20,25,5,16,28

3,7,9,**11,0**,2,20,25,5,16,28 -> 3,7,9,**0,11**,2,20,25,5,16,28

3,7,9,0,**11,2**,20,25,5,16,28 -> 3,7,9,0,**2,11**,20,25,5,16,28

3,7,9,0,2,**11,20**,25,5,16,28 -> 3,7,9,0,2,**11,20**,25,5,16,28

3,7,9,0,2,11,**20,25**,5,16,28 -> 3,7,9,0,2,11,**20,25**,5,16,28

3,7,9,0,2,11,20,**25,5**,16,28 -> 3,7,9,0,2,11,20,**5,25**,16,28

3,7,9,0,2,11,20,5,**25,16**,28 -> 3,7,9,0,2,11,20,5,**16,25,28**

……

0,2,3,5,7,9,11,16,20,25,28

d-) Merge Sort

25,9,3,7,11,0,2,20,28,5,16

25,9,3,7,11,0 2,20,28,5,16

25,9,3,7,11,0 2,20,28,5,16

25,9,3 7,11,0 2,20 28,5,16

25,9,3 7,11,0 2,20 28,5,16

25,9, 3 7,11, 0 2, 20 28,5, 16

25, 9, 3 7, 11, 0 2, 20 28, 5, 16

9,25, 3 7,11, 0 2,20 5,28, 16

3,9,25 0,7,11 2,20 5,16,28

0,3,7,9,11,25 2,5,16,20,28

0,2,3,5,7,9,11,16,20,25,28

e-) Quick Sort

25,9,3,7,11,0,2,20,28,5,16

5,3,7,11,0,2,20,28,25,16

5,3,7,11,0,2,**16**,28,25,20

<16,,,,,,,,,,,,,,,,, >16

Same for others

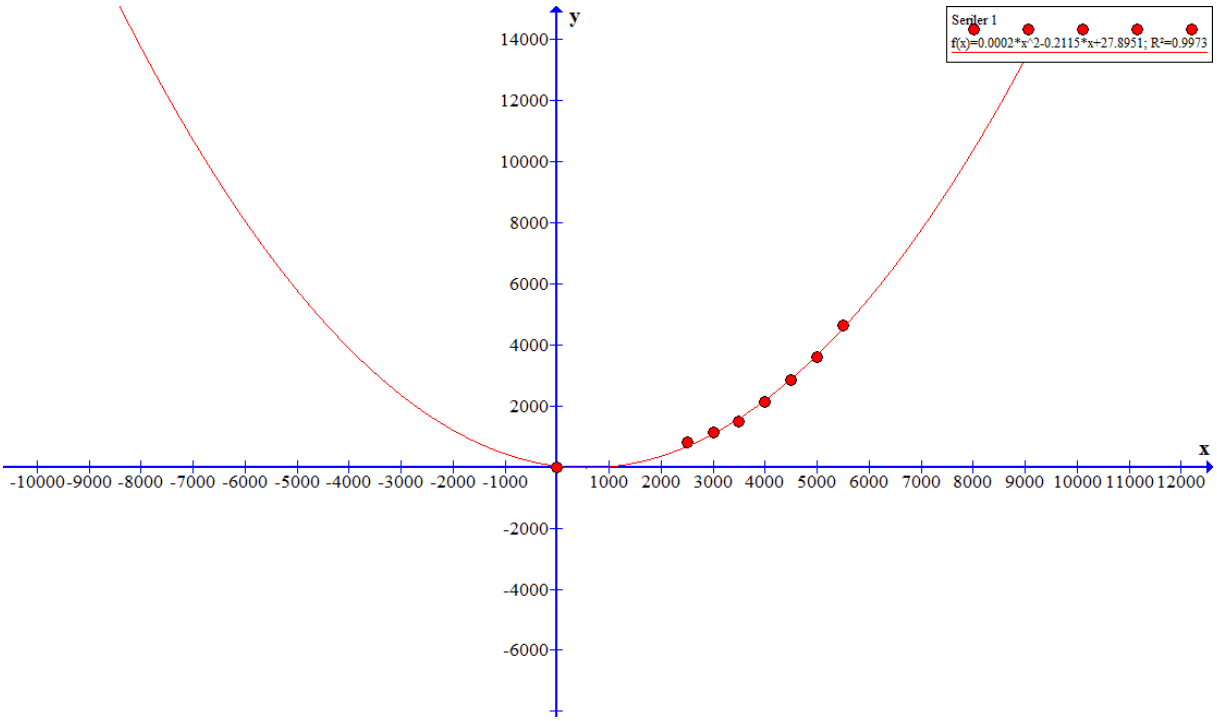
0, 2, 7, 11, 5, 3, 16, 20, 25, 28

……..

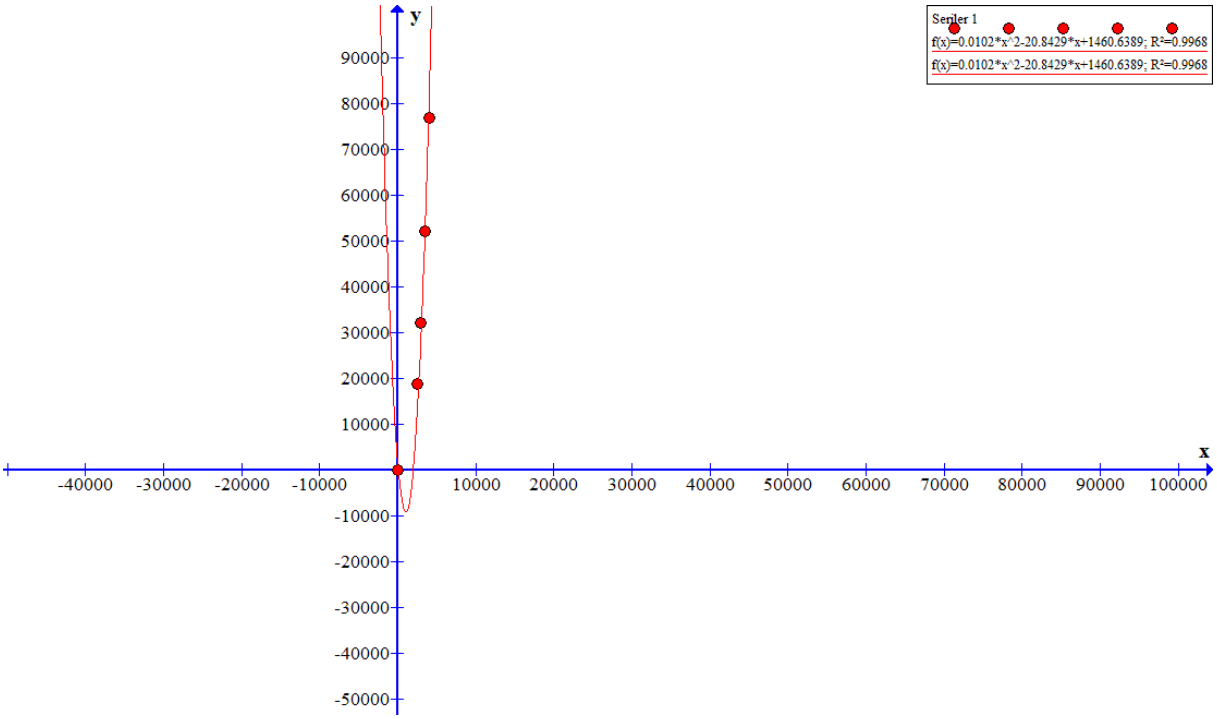
0,2,3,5,7,9,11,16,20,25,28

3-]

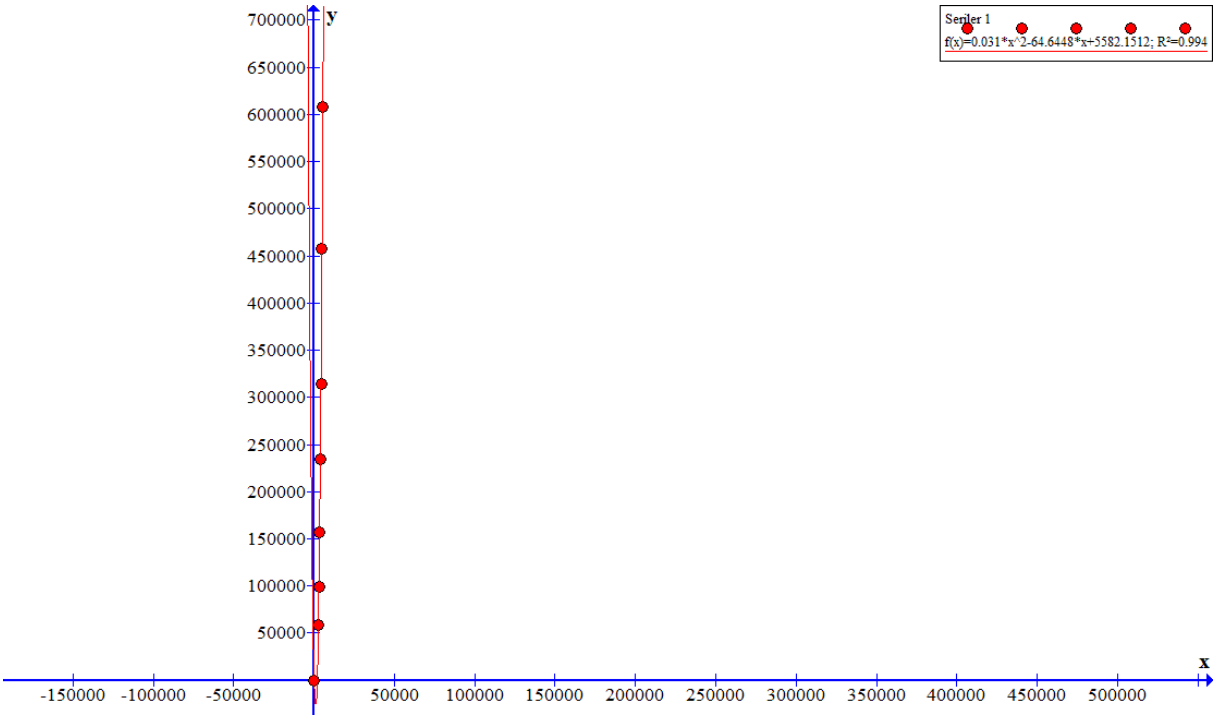
Elapsed times for **<u>quick sort algorithm</u>** with random numbers:



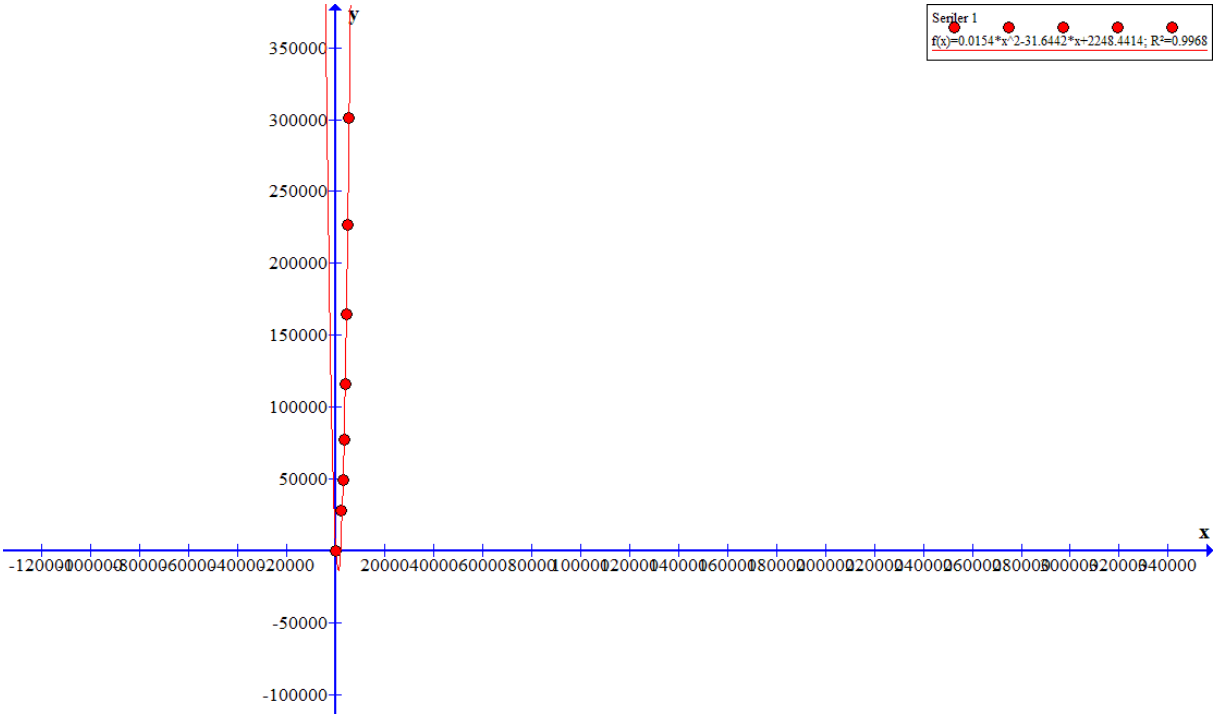
Elapsed times for **<u>insertation sort algorithm</u>** with random numbers:



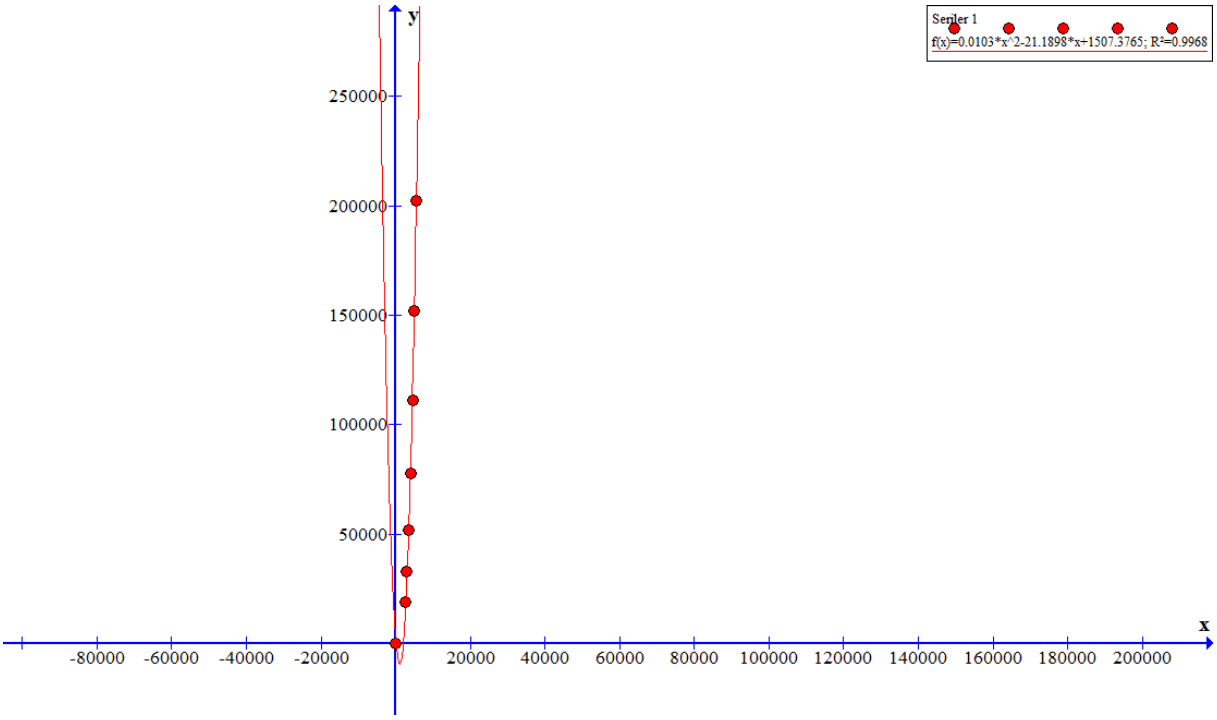
Elapsed times for **quick sort algorithm** with descending numbers:



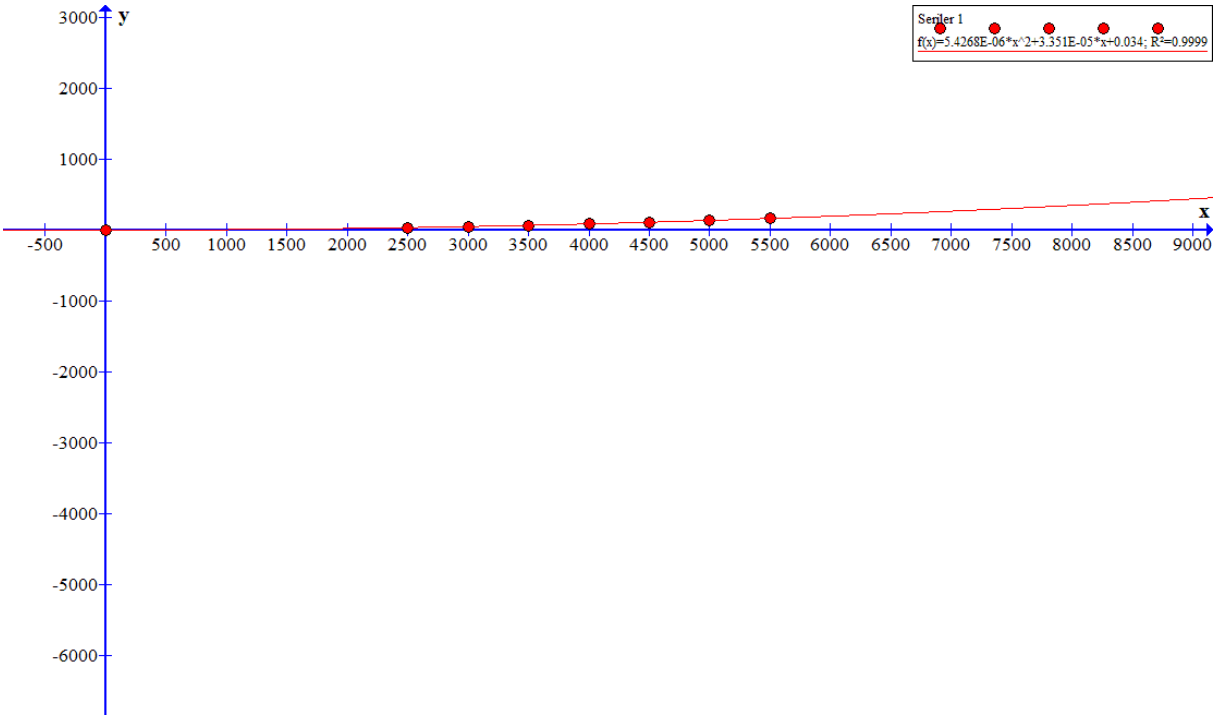
Elapsed times for **insertation sort algorithm** with descending numbers:



Elapsed times for **quick sort algorithm** with ascending numbers:



Elapsed times for **insertation sort algorithm** with ascending numbers:



If algorithms use randomly generated numbers, then they theoretic results will close to experimental results.

According to theoretical results, quick sort performs in nlog(n), insertation sort perform in n^2 time. First graphic of insertation sort is similar to theoretical result (n^2). First graphic of quick sort does not seem like nlog(n). This will be because of not to use great values or inefficiency of the algorithm.

If algorithms use ascending and descending numbers, both of them perform n^2 as expected with the theoretical results.

4)

```
#include <iostream>
#include <string>
using namespace std;
int minimum(int array[], int size) {
   if (size == 1) {
      return array[0];
   }
   else {
      return (array[size] < minimum(array, size - 1)) ? array[size] : minimum(array, size - 1);
   }
}
int main()
{
   int array[5] = {4, 7, 5, 3, 10};
   cout << "min: " << minimum(array, 5) << " . \n";
}

//         T(n) = T(n-1) + O(1)        ->         O(n)
```

5)

-

log(n!) = log(1) + log(2) + ... + log(n-1) + log(n)

upper bound;

log(1) + log(2) + ... + log(n) <= log(n) + log(n) + ... + log(n) = n*log(n)

lower bound;

log(1) + ... + log(n/2) + ... + log(n) >= log(n/2) + ... + log(n)

>= log(n/2) + ... + log(n/2)

= n/2 * log(n/2)

So;

n/2 * log(n/2) <= lg(n!) <= n lg n

➔ 0(n lg n)

-

T(1) = 0, n>2

T(n) = 2T(n/2) + n

= 2(2T(n/4) + n/2) + n = 4t(n/4) + 2n

=4(2T(n/8) + n/4) + 2n = 8t(n/4) + 3n

……

T(n) = n*T(1) + log(n)*n

(2^k=n)

➔ 0(n*logn)

-

show,

f(n) = 4n^5 + 3n^2 + 1, is order of 0(n^5) ?

c > 0 and n0 >= 1 such that 4n^5 + 3n^2 + 1 <= cn^5 for every num n >= n0. So, c = 5, n0 = 1.

2)

time: 0

0 2 3 5 7 9 11 16 20 25 28

comparison: 26

movement: 46

time: 0

0 2 3 5 7 9 11 16 20 25 28

comparison: 27

movement: 78

time: 0

0 2 3 5 7 9 11 16 20 25 28

comparison: 18

movement: 27

insertion sort size: 20000

comparison: 100418065

movement: 100458063

time elapsed: 359

merge sort size: 20000

comparison: 100678937

movement: 101032527

time elapsed: 17

quick sort size: 20000

comparison: 264649

movement: 210858

time elapsed: 3

insertion sort size: 30000

comparison: 223314136

movement: 223374134

time elapsed: 762

merge sort size: 30000

comparison: 223722802

movement: 224268598

time elapsed: 26

quick sort size: 30000

comparison: 447914

movement: 331851

time elapsed: 5

insertion sort size: 40000

comparison: 398169149

movement: 398249147

time elapsed: 1348

merge sort size: 40000

comparison: 398730783

movement: 399478075

time elapsed: 35

quick sort size: 40000

comparison: 639665

movement: 462270

time elapsed: 6

insertion sort size: 50000

comparison: 625873634

movement: 625973632

time elapsed: 2104

merge sort size: 50000

comparison: 626591918

movement: 627542560

time elapsed: 44

quick sort size: 50000

comparison: 803596

movement: 603435

time elapsed: 8

insertion sort size: 60000

comparison: 898812248

movement: 898932246

time elapsed: 3050

merge sort size: 60000

comparison: 899689350

movement: 900841174

time elapsed: 53

quick sort size: 60000

comparison: 984347

movement: 753774

time elapsed: 10

descending order;

insertion sort size: 2500

comparison: 3123737

movement: 3128735

time elapsed: 11

merge sort size: 2500

comparison: 3138489

movement: 3185543

time elapsed: 2

quick sort size: 2500

comparison: 3099868

movement: 7485

time elapsed: 7

insertion sort size: 3000

comparison: 4498490

movement: 4504488

time elapsed: 16

merge sort size: 3000

comparison: 4516566

movement: 4574296

time elapsed: 2

quick sort size: 3000

comparison: 4483826

movement: 8973

time elapsed: 11

insertion sort size: 3500

comparison: 6123233

movement: 6130231

time elapsed: 21

merge sort size: 3500

comparison: 6144573

movement: 6213039

time elapsed: 2

quick sort size: 3500

comparison: 6070082

movement: 10470

time elapsed: 14

insertion sort size: 4000

comparison: 7997986

movement: 8005984

time elapsed: 27

merge sort size: 4000

comparison: 8022162

movement: 8101792

time elapsed: 3

quick sort size: 4000

comparison: 7962054

movement: 11982

time elapsed: 19

insertion sort size: 4500

comparison: 10122729

movement: 10131727

time elapsed: 35

merge sort size: 4500

comparison: 10150877

movement: 10241343

time elapsed: 4

quick sort size: 4500

comparison: 10056168

movement: 13467

time elapsed: 24

insertion sort size: 5000

comparison: 12497475

movement: 12507473

time elapsed: 42

merge sort size: 5000

comparison: 12529479

movement: 12631089

time elapsed: 4

quick sort size: 5000

comparison: 12395646

movement: 14964

time elapsed: 29

insertion sort size: 5500

comparison: 15122228

movement: 15133226

time elapsed: 52

merge sort size: 5500

comparison: 15157984

movement: 15270842

time elapsed: 5

quick sort size: 5500

comparison: 15032232

movement: 16461

time elapsed: 35

ascending order;

insertion sort size: 2500

comparison: 0

movement: 4998

time elapsed: 0

merge sort size: 2500

comparison: 13672

movement: 61806

time elapsed: 2

quick sort size: 2500

comparison: 3092809

movement: 7428

time elapsed: 7

insertion sort size: 3000

comparison: 0

movement: 5998

time elapsed: 0

merge sort size: 3000

comparison: 16839

movement: 75806

time elapsed: 3

quick sort size: 3000

comparison: 4459609

movement: 8931

time elapsed: 11

insertion sort size: 3500

comparison: 0

movement: 6998

time elapsed: 0

merge sort size: 3500

comparison: 20081

movement: 89806

time elapsed: 3

quick sort size: 3500

comparison: 6070439

movement: 10407

time elapsed: 14

insertion sort size: 4000

comparison: 0

movement: 7998

time elapsed: 0

merge sort size: 4000

comparison: 23747

movement: 103806

time elapsed: 3

quick sort size: 4000

comparison: 7939877

movement: 11904

time elapsed: 18

insertion sort size: 4500

comparison: 0

movement: 8998

time elapsed: 0

merge sort size: 4500

comparison: 26680

movement: 118614

time elapsed: 5

quick sort size: 4500

comparison: 10017549

movement: 13383

time elapsed: 23

insertion sort size: 5000

comparison: 0

movement: 9998

time elapsed: 0

merge sort size: 5000

comparison: 29835

movement: 133614

time elapsed: 5

quick sort size: 5000

comparison: 12375738

movement: 14847

time elapsed: 30

insertion sort size: 5500

comparison: 0

movement: 10998

time elapsed: 0

merge sort size: 5500

comparison: 33092

movement: 148614

time elapsed: 4

quick sort size: 5500

comparison: 14934921

movement: 16326

time elapsed: 35