<Report 1. Test results of programming part>

2016314364 박수헌

1.a.

```
BUBBLE SORT (A[1,2,... n])
  for i = 1 to n
      for j = 1 to n-i
          if A[j] > A[j+1]
              // swap A[j] and A[j+1]
              temp = A[j]
              A[j] = A[j+1]
              A[j+1] = temp
```

1.b.1) int A[100] : filled by rand()%1000

```
Microsoft Visual Studio 디버그 콘솔
1) filled by rand()
before sorting :
784 494 132 217 945 3 749 782 746 817 578 755 896 856 224 838 155 946 779 738 759 93 23 261 992 567 223 39 769 692 123 657 788 172 295 475 220 71
 532 549 833 482 691 828 555 90 968 546 960 62 466 82 220 886 441 480 558 425 33 938 901 664 548 42 346 926 54 289 896 78 793 449 123 690 653 270
 406 900 969 118 704 174 48 111 770 493 779 818 249 841 221 432 465 453 684 440 744 471 587 176

after sorting :
# of comparisons : 4950
3 23 33 39 42 48 54 62 71 78 82 90 93 111 118 123 123 132 155 172 174 176 217 220 220 221 223 224 249 261 270 289 295 346 406 425 432 440 441 449
 453 465 466 471 475 480 482 493 494 532 546 548 549 555 558 567 578 587 653 657 664 684 690 691 692 704 738 744 746 749 755 759 769 770 779 779
782 784 788 793 817 818 828 833 838 841 856 886 896 896 900 901 926 938 945 946 960 968 969 992
```

1.b.2) int A[100] : already sorted

```
2) already sorted
before sorting :
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 5
2 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 10
0

after sorting :
# of comparisons : 4950
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 5
2 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 10
0
```

1.b.3) int A[100] : reversely sorted

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

after sorting :
# of comparisons : 4950
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 5
2 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 10
0
```

2.1) int A[100] : filled with rand()%1000

```
1) filled by rand()
before sorting :
446 574 202 547 206 707 954 241 291 136 974 396 294 836 554 31 12 769 600 204 489 232 686 638 768 181 493 883 792 276 90
7 116 657 501 212 760 248 692 381 302 236 120 544 808 428 573 928 211 714 529 198 278 352 73 887 316 773 315 853 138 325
 165 171 230 83 21 271 694 610 743 265 59 98 828 416 975 46 336 408 612 366 582 516 264 506 139 16 185 384 955 534 874 9
33 826 361 932 505 3 242 935

after sorting :
# of comparisons : 545
975 974 955 954 935 933 932 928 907 887 883 874 853 836 828 826 808 792 773 769 768 760 743 714 707 694 692 686 657 638
612 610 600 582 574 573 554 547 544 534 529 516 506 505 501 493 489 446 428 416 408 396 384 381 366 361 352 336 325 316
315 302 294 291 278 276 271 265 264 248 242 241 236 232 230 212 211 206 204 202 198 185 181 171 165 139 138 136 120 116
98 83 73 59 46 31 21 16 12 3
```

2.2) int A[100] : already sorted (from 100 down to 1)

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

after sorting :
# of comparisons : 356
100 99 98 97 96 95 94 93 92 91 90 89 88 87 86 85 84 83 82 81 80 79 78 77 76 75 74 73 72 71 70 69 68 67 66 65 64 63 62 61
 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21
 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
```

2.3) int A[100] : reversely sorted(from 1 too 100)

```
3) reversely sorted
before sorting :
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

after sorting :
# of comparisons : 316
100 99 98 97 96 95 94 93 92 91 90 89 88 87 86 85 84 83 82 81 80 79 78 77 76 75 74 73 72 71 70 69 68 67 66 65 64 63 62 61
 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21
 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
```

3. Results

```
Iteration 1
268 251 907 571 427 322 653 11 519 109 729 431 706 10 339 874 608 137 935 730
679 580 718 407 991 516 340 322 770 788 188 783 618 774 547 668 6 322 308 169
934 63 453 24 739 415 718 156 444 897 262 833 210 615 834 389 975 459 44

Iteration 2
268 251 907 571 427 322 653 11 519 109 729 431 706 10 339 874 608 137 935 730
679 580 718 407 991 516 340 322 770 788 188 783 618 774 547 668 6 322 308 169
934 63 453 24 739 415 718 156 444 897 262 833 210 615 834 389 975 44

Iteration 3
268 251 907 571 427 322 653 11 519 109 729 431 706 10 339 874 608 137 935
730 679 580 718 407 991 516 340 322 770 788 188 783 618 774 547 668 6 322
308 169 934 63 453 24 739 415 718 156 444 897 262 833 210 615 834 389 44
```

4.1) Matrix multiplication using standard algorithm

```
4x4
Matrix A :
570 560 600 456

948 999 878 376

738 441 674 546

932 684 592 572

Matrix B :
462 388 223 950

904 749 618 484

515 519 155 939

439 829 357 750

Matrix C (in standard algorithm) :

# of computations (addition) : 48
# of computations (multiplication) : 64
1278764 1330024 728982 1717940

1958306 1883461 1099108 2490558

1326424 1419093 736504 1956930

1604908 1655368 926512 2201344
```

```
8x8
Matrix A :
149 583 284 472 428 200 216 709

350 734 146 566 947 328 235 370

548 28 329 666 355 552 654 31

11 69 127 433 596 919 246 544

857 735 913 223 698 441 729 316

762 797 939 338 417 506 850 856

835 799 962 967 559 504 191 647

241 63 349 312 21 703 537 876

Matrix B :
121 29 847 141 451 256 590 548

25 558 159 829 878 416 246 410

309 220 730 388 504 597 389 407

788 232 617 179 667 856 246 20

809 171 157 867 512 579 137 468

341 910 296 286 856 574 735 493

375 177 379 763 355 938 0 611

953 655 318 995 207 53 797 535

Matrix C (in standard algorithm) :

# of computations (addition) : 448
# of computations (multiplication) : 512
1663425 1259434 1151166 1997535 1650812 1457049 1228625 1255905

1870528 1327516 1321450 2278110 2179055 1943227 1248803 1509917

1443697 957496 1596551 1342850 1774629 2055933 1101087 1313462

1789728 1505285 1011981 1694079 1710116 1610157 1370070 1268008

1869499 1544260 2263726 2726252 2699809 2618927 1768068 2305593

2313039 1994717 2475756 3199514 2867712 2751090 2205332 2555203

2492575 1917862 2648267 2744750 3127310 2774616 2263986 2168974

1677348 1503457 1354894 1778084 1532492 1529144 1587955 1459355
```

4.2) Matrix multiplication using recursion

4x4 :

```
Matrix C (in recursion) :
partial maxtix is constructed
649656 1176125

560453 1048553

____________________
1872141 1326398

1724724 1239639

____________________
566544 959237

364289 501487

____________________
1298635 1064746

623692 557857

____________________
649656 1176125 1872141 1326398

560453 1048553 1724724 1239639

566544 959237 1298635 1064746

364289 501487 623692 557857
```

Number of multiplications when multiplying 2x2 matrix is 8, and the number of additions when multiplying 2x2 matrix is 4.

Therefore multiplying 4x4 matrices recursively means that 8 multiplications and 4 additions will happen when each partition multiplication happens.

Accordingly, in 4x4 matrices case, multiplications happen 8*8 = 64 times, and additions happen 8*4+4 = 68 times.

In 8x8 matrices case, multiplications happen 8*8*8 = 512 times, and additions happen 8*((8*4)+4)+4 = 292 times.

According to 4.1) which used a standard algorithm method, following is the number of computations comparison.

	4x4		8x8	
	Standard Algorithm	Recursion	Standard Algorithm	Recursion
Multiplication	64	64	512	512
Addition	48	68	448	292
Total	112	132	960	804

8x8 :

```
Matrix C (in recursion) :
partial maxtix is constructed
525131 758443

1026853 1261982

____________________

380282 169358

634939 334866

____________________

275400 255919

497182 665913

____________________

546113 129826

371007 456235

____________________

partial maxtix is constructed
1261193 1516471

1290162 1743931

____________________

1201541 2478575

1191950 2309344

____________________

1080263 1543798

693949 739189

____________________

961269 2153427

701414 1238131

____________________

partial maxtix is constructed
546913 706866

1022806 1225880

____________________

735448 436684

1194042 848975

____________________

228388 374407

470750 790215
```

→

```
____________________

291116 270543

666497 398453

____________________

partial maxtix is constructed
1709785 1070477

1967569 1085319

____________________

1540076 1659924

1589316 1592442

____________________

1546588 869138

982919 623763

____________________

1579832 1480058

657586 820601

____________________

partial maxtix is constructed
1630141 1754093

1045093 1080370

____________________

1238552 885371

654961 394570

____________________

699218 763932

352404 340009

____________________

342426 397430

495281 274121

____________________

partial maxtix is constructed
776081 785715

592437 596240

____________________

536593 1290195

579217 1048553

____________________

1532515 1787154

567138 895243
```

→

```
____________________

1064469 2163655

877824 1529756

____________________

partial maxtix is constructed
1510497 2019564

985841 1151125

____________________

1697161 1360810

1009334 868761

____________________

659524 844152

294566 500687

____________________

714502 560890

364903 316191

____________________

partial maxtix is constructed
977638 572440

811865 523833

____________________

681242 901057

527598 695031

____________________

2290604 1153850

860470 624306

____________________

1270600 1577338

1124814 957401

____________________

partial maxtix is constructed
1786324 2274914 1581823 2647933

2317015 3005913 1826889 2644210

1355663 1799717 1507382 2283253

1191131 1405102 1072421 1694366
```

```
2317015 3005913 1826889 2644210
1355663 1799717 1507382 2283253
1191131 1405102 1072421 1694366
————————————————————
2256698 1777343 2275524 2096608
2990375 2311199 2783358 2441417
1774976 1243545 1870948 1750601
1453669 1413978 1324083 1219054
————————————————————
2406222 2539808 1775145 2175566
1637530 1676610 1234178 1443123
2231733 2551086 1406895 2561085
919542 1235252 1373105 1803877
————————————————————
2488135 2592004 2378403 2261867
1797706 1674958 1536932 1563792
2950128 1998002 1985102 2138228
1155036 1124993 1489717 1273592
————————————————————
1786324 2274914 1581823 2647933 2256698 1777343 2275524 2096608
2317015 3005913 1826889 2644210 2990375 2311199 2783358 2441417
1355663 1799717 1507382 2283253 1774976 1243545 1870948 1750601
1191131 1405102 1072421 1694366 1453669 1413978 1324083 1219054
2406222 2539808 1775145 2175566 2488135 2592004 2378403 2261867
1637530 1676610 1234178 1443123 1797706 1674958 1536932 1563792
2231733 2551086 1406895 2561085 2950128 1998002 1985102 2138228
919542 1235252 1373105 1803877 1155036 1124993 1489717 1273592
```