Q1)

a=

**public** **static** **void** square\_matrix\_multiply(**int** [][] A, **int** [][] B, **int** [][] C)

{

**for**(**int** i=0;i<A.length;i++){

**for**(**int** j= 0; j<B.length;j++){

**for** (**int** k = 0; k < B.length; k++)

C[i][j] += A[i][k] \* B[k][j];

}

}

*print\_2d\_array*(C);

}

b=

18 14

62 66

c=

//part 1(c)

//initialize elements of matrices with random integers

*initialize\_2d\_array\_random*(A\_2);

*initialize\_2d\_array\_random*(B\_2);

start\_time= System.*nanoTime*();

*square\_matrix\_multiply*(A\_2, B\_2, C\_2);

end\_time= System.*nanoTime*();

elapsed\_time= end\_time-start\_time;

System.***out***.println("Time it takes: " +elapsed\_time);

32786 32241 38724 48736 43801 43951 38516 23717 34341 36167 47868 38184 41344 33343 47315 36456

35935 31585 40913 42956 46405 32544 42354 34135 37423 43400 50796 40385 40863 45460 45228 39560

Time it takes: 22791022

d=

//part 1(d)

//initialize elements of matrices with random integers

*initialize\_2d\_array\_random*(A\_3);

*initialize\_2d\_array\_random*(B\_3);

*initialize\_2d\_array*(C\_3);

start\_time= System.*nanoTime*();

*square\_matrix\_multiply*(A\_3, B\_3, C\_3);

end\_time= System.*nanoTime*();

elapsed\_time= end\_time-start\_time;

System.***out***.println("Time it takes part d: " +elapsed\_time);

187685 169558 166205 142876 153255 133423 138989 153335 136045 172631 178321 159620 150713 171573 170199 148320 159525 154488 143011 143698 173381 150414 157801 123206 168104 154201 148730 153582 129235 183301 145151 157661 149703 166685 157556 132063 146113 165713 180168 150946 169021 156756 152605 161253 148331 173052 132962 175012 157251 159458 137429 132143 139399 156178 178561 145992 166381 160646 163662 164967 166449 141737 154691 151735

181741 173129 159675 145038 151793 130804 142509 152704 138857 174104 168681 161902 144420 171226 166549 150139 166996 153862 156581 141368 164598 164401 160380 138691 169322 174953 147291 153213 128022 183060 137776 179729 145755 159743 157554 156007 152108 160424 181166 139398 163283 156550 159672 165233 140407 176194 126163 158077 162408 169114 141580 126531 143707 164660 190606 142724 162842 163456 159888 163142 160674 138958 166271 149161

Time it takes part d: 126985395

**Increase**: 126985395-22791022= 104194373= which is quite a lot.

**Q2)**

**a-**

// part 2(a)

**public** **static** **void** square\_matrix\_multiply\_recursive(**int**[][] A, **int**[][] B, **int** A\_row\_index\_start,

**int** A\_row\_index\_end, **int** A\_col\_index\_start, **int** A\_col\_index\_end, **int** B\_row\_index\_start, **int** B\_row\_index\_end,

**int** B\_col\_index\_start, **int** B\_col\_index\_end, **int** C\_row\_index\_start, **int** C\_row\_index\_end,

**int** C\_col\_index\_start, **int** C\_col\_index\_end, **int**[][] C) {

**int** n = A\_row\_index\_end - A\_row\_index\_start + 1;

// Divide step, compute the middle points for the row and column indices of A

// and B matrix

**int** A11\_row\_index\_start = A\_row\_index\_start;

**int** A11\_row\_index\_end = (A\_row\_index\_end + A\_row\_index\_start) / 2;

**int** A11\_col\_index\_start = A\_col\_index\_start;

**int** A11\_col\_index\_end = (A\_col\_index\_end + A\_col\_index\_start) / 2;

**int** A12\_row\_index\_start = A\_row\_index\_start;

**int** A12\_row\_index\_end = (A\_row\_index\_end + A\_row\_index\_start) / 2;

**int** A12\_col\_index\_start = (A\_col\_index\_end + A\_col\_index\_start) / 2 + 1;

**int** A12\_col\_index\_end = A\_col\_index\_end;

**int** A21\_row\_index\_start = (A\_row\_index\_end + A\_row\_index\_start) / 2 + 1;

**int** A21\_row\_index\_end = A\_row\_index\_end;

**int** A21\_col\_index\_start = A\_col\_index\_start;

**int** A21\_col\_index\_end = (A\_col\_index\_end + A\_col\_index\_start) / 2;

**int** A22\_row\_index\_start = (A\_row\_index\_end + A\_row\_index\_start) / 2 + 1;

**int** A22\_row\_index\_end = A\_row\_index\_end;

**int** A22\_col\_index\_start = (A\_col\_index\_end + A\_col\_index\_start) / 2 + 1;

**int** A22\_col\_index\_end = A\_col\_index\_end;

**int** B11\_row\_index\_start = B\_row\_index\_start;

**int** B11\_row\_index\_end = (B\_row\_index\_end + B\_row\_index\_start) / 2;

**int** B11\_col\_index\_start = B\_col\_index\_start;

**int** B11\_col\_index\_end = (B\_col\_index\_end + B\_col\_index\_start) / 2;

**int** B12\_row\_index\_start = B\_row\_index\_start;

**int** B12\_row\_index\_end = (B\_row\_index\_end + B\_row\_index\_start) / 2;

**int** B12\_col\_index\_start = (B\_col\_index\_end + B\_col\_index\_start) / 2 + 1;

**int** B12\_col\_index\_end = B\_col\_index\_end;

**int** B21\_row\_index\_start = (B\_row\_index\_end + B\_row\_index\_start) / 2 + 1;

**int** B21\_row\_index\_end = B\_row\_index\_end;

**int** B21\_col\_index\_start = B\_col\_index\_start;

**int** B21\_col\_index\_end = (B\_col\_index\_end + B\_col\_index\_start) / 2;

**int** B22\_row\_index\_start = (B\_row\_index\_end + B\_row\_index\_start) / 2 + 1;

**int** B22\_row\_index\_end = B\_row\_index\_end;

**int** B22\_col\_index\_start = (B\_col\_index\_end + B\_col\_index\_start) / 2 + 1;

**int** B22\_col\_index\_end = B\_col\_index\_end;

**if** (n == 1) {

C[C\_row\_index\_start][C\_col\_index\_start] += A[A\_row\_index\_start][A\_col\_index\_start]

\* B[B\_row\_index\_start][B\_col\_index\_start];

}

**else** {

// implement the recursive part here

// Recursive call 1 for C11

*square\_matrix\_multiply\_recursive*(A, B, A11\_row\_index\_start, A11\_row\_index\_end,

A11\_col\_index\_start, A11\_col\_index\_end, B11\_row\_index\_start, B11\_row\_index\_end, B11\_col\_index\_start,

B11\_col\_index\_end, C\_row\_index\_start, C\_row\_index\_end, C\_col\_index\_start, C\_col\_index\_end, C);

C\_row\_index\_start = A11\_row\_index\_start;

C\_row\_index\_end = A11\_row\_index\_end;

C\_col\_index\_start = A11\_col\_index\_start;

C\_col\_index\_end = A11\_col\_index\_end;

// Recursive call 2 for C11

*square\_matrix\_multiply\_recursive*(A, B, A12\_row\_index\_start, A12\_row\_index\_end,

A12\_col\_index\_start, A12\_col\_index\_end, B21\_row\_index\_start, B21\_row\_index\_end, B21\_col\_index\_start,

B21\_col\_index\_end, C\_row\_index\_start, C\_row\_index\_end, C\_col\_index\_start, C\_col\_index\_end, C);

// Recursive call 1 for C12

*square\_matrix\_multiply\_recursive*(A, B, A11\_row\_index\_start, A11\_row\_index\_end, A11\_col\_index\_start,

A11\_col\_index\_end, B12\_row\_index\_start, B12\_row\_index\_end, B12\_col\_index\_start, B12\_col\_index\_end,

C\_row\_index\_start, C\_row\_index\_end, C\_col\_index\_start, C\_col\_index\_end, C);

C\_row\_index\_start = A12\_row\_index\_start;

C\_row\_index\_end = A12\_row\_index\_end;

C\_col\_index\_start = A12\_col\_index\_start;

C\_col\_index\_end = A12\_col\_index\_end;

// Recursive call 2 for C12

*square\_matrix\_multiply\_recursive*(A, B, A12\_row\_index\_start, A12\_row\_index\_end, A12\_col\_index\_start,

A12\_col\_index\_end, B22\_row\_index\_start, B22\_row\_index\_end, B22\_col\_index\_start, B22\_col\_index\_end,

C\_row\_index\_start, C\_row\_index\_end, C\_col\_index\_start, C\_col\_index\_end, C);

// Recursive call 1 for C21

*square\_matrix\_multiply\_recursive*(A, B, A21\_row\_index\_start, A21\_row\_index\_end, A21\_col\_index\_start,

A21\_col\_index\_end, B11\_row\_index\_start, B11\_row\_index\_end, B11\_col\_index\_start, B11\_col\_index\_end,

C\_row\_index\_start, C\_row\_index\_end, C\_col\_index\_start, C\_col\_index\_end, C);

C\_row\_index\_start = A21\_row\_index\_start;

C\_row\_index\_end = A21\_row\_index\_end;

C\_col\_index\_start = A21\_col\_index\_start;

C\_col\_index\_end = A21\_col\_index\_end;

// Recursive call 2 for C21

*square\_matrix\_multiply\_recursive*(A, B, A22\_row\_index\_start, A22\_row\_index\_end, A22\_col\_index\_start,

A22\_col\_index\_end, B21\_row\_index\_start, B21\_row\_index\_end, B21\_col\_index\_start, B21\_col\_index\_end,

C\_row\_index\_start, C\_row\_index\_end, C\_col\_index\_start, C\_col\_index\_end, C);

// Recursive call 1 for C22

*square\_matrix\_multiply\_recursive*(A, B, A21\_row\_index\_start, A21\_row\_index\_end, A21\_col\_index\_start,

A21\_col\_index\_end, B12\_row\_index\_start, B12\_row\_index\_end, B12\_col\_index\_start, B12\_col\_index\_end,

C\_row\_index\_start, C\_row\_index\_end, C\_col\_index\_start, C\_col\_index\_end, C);

C\_row\_index\_start = A22\_row\_index\_start;

C\_row\_index\_end = A22\_row\_index\_end;

C\_col\_index\_start = A22\_col\_index\_start;

C\_col\_index\_end = A22\_col\_index\_end;

// Recursive call 2 for C12

*square\_matrix\_multiply\_recursive*(A, B, A22\_row\_index\_start, A22\_row\_index\_end, A22\_col\_index\_start,

A22\_col\_index\_end, B22\_row\_index\_start, B22\_row\_index\_end, B22\_col\_index\_start, B22\_col\_index\_end,

C\_row\_index\_start, C\_row\_index\_end, C\_col\_index\_start, C\_col\_index\_end, C);

}

}

**Implement:**

*square\_matrix\_multiply\_recursive*(A, B,0, 1, 0,1, 0,1, 0,1,0,1,0,1, C);

System.***out***.println("This was the array");

*print\_2d\_array*(C);

**Output:** It gives me wrong output

26 48

76 10s