**4.2-2**

//we assume the usual notations from the textbook for the 4 “quarters” of the original matrices A and B (e.g.A1,1,…B2,2) – initially these have size n/2 each

The algorithm can be expressed in pseudo-code like below:

Strassen(A,B) //size n= power of 2

If n=1 then return AxB

Else //first, 7 recursive calls

P1 = Strassen(A1,1, B1,2 – B2,2)

P2 = Strassen(A2,1,+ A1,2 , B2,2)

P3 = Strassen(A2,1 +A2,2 , B1,1)  
 P4 = Strassen(A2,2, B2,1 – B1,1)

P5 = Strassen(A1,1 + A2,2 , B1,1 + B2,2)

P6 = Strassen(A1,2 – A2,2, B2,1 + B2,2)

P7 = Strassen(A1,1 –A2,1, B1,1 + B1,2)

//then outputting the necessary 4 “quarters” of the final resulting matrix C

C1,1 = P5+P4-P2+P6

C1,2 = P1+P2

C2,1 = P3+P4

C2,2 = P1+P5 –P3-P7

**4.2-3**

If n is not an exact power of 2, it is surely residing between two consecutive powers of 2, as any number which is not a power of 2.

Therefore, we can find m such that 2m-1 < n < 2m with m>0.

In this case, we pad with as many zeros as needed to obtain matrices of size 2m.

Afterwards, we can apply Strassen’s on the “new” matrices (i.e. with elements of 0 added).

First, it is clear the multiplication’s result is still the same (like when zeros are not padded), because the added 0’s do not modify anything.

In what regards the running time, this is now (for Strassen’s applied in the usual manner) equal to Θ[(2m)lg7] (because the “extended” n is now 2m and the same formula applies to it now).

But (2m)lg7= (2lg7)m =7m

Since 2m-1< n<2m => m=[lgn] (by [] we meant CEILING function).

Or, this means the algorithm has a tight bound equal to 7[lgn] which is the same with nlg7 (using basic logarithm properties) and this is identical with the running time when 0s were not used (when n is an exact power of 2).

**4.2-4**

We recall Strassen’s algorithm recurrence is T(n)=7T(n/2)+f(n), where f(n) is in Theta(n2)

For this algorithm, the recurrence is similar, namely T(n)=kT(n/3)+g(n), where g(n) is also in Theta(n2).

This holds because we must find k such that no. of multiplications when doing for 3x3 matrices remains better than nlg7 (which characterizes the Strassen’s routine).

Therefore, we can conclude that our k must satisfy the constraint: nlg7 (since we know that k multiplications are required). Of course, we deal mainly with the asymptotic growth rate and not with the afferent constants also.

The above conditions leads to log3klg7 which is the same with k3lg7 ~ 21.85

Therefore, the largest k for which the no. of multiplications remains in o(nlg7) is 21.

To verify, we can plug k=21 in the recurrence T(n)=21T(n/3)+ Θ(n2) and apply Case 1 of the Master Theorem (a=21, b=3, logba=log321 >2, thus the recursive term dominates g(n)) and obtain T(n)= Θ(nlog3(21))~ Θ(n2.77) which is better than Θ(nlg7)~ Θ(n2.8)

**4.2-5**

We use the recurrence adapted for all 3 cases because the 3 problems have the same underlying structure, namely a number of subproblems solved recursively and a combinations of their results (as done in Divide in Conquer paradigm).

Thus, we can write as general form the recurrence T(n)= kT(n/m) + Θ(n2)

As we have seen, this solves (as per Case 1 of Master Theorem) to Θ(nlogm(k)) because the recursive term dominates the non-recursive part (Θ(n2)).

Now, in order to compare the 3 situations, we practically need to order logm(k) for each (m,k) pair of values.

Using any tool or resource for calculation of the logarithm, we obtain:

Option 1) log68(132464)~ 2.7951284874

Option 2) log70(143640) ~2.7951226897

Option 3) log72(155424)~ 2.7951473911

We can notice that Option 2) gives the best asymptotic running time, which is better than Strassen’s also (since lg7~ 2.8073549221)

**4.2-6**

Using basic multiplication rules between matrices (A[m,n] x B[n,p]= C[m,p]), we can state that:

* [kn,n] x [n,kn] => [kn,kn] as resulting matrix (1)
* For reversed order, [n,kn] x [kn,n] => [n,n] as resulting matrix (2)

In Case 1, there are k2 “cells” in the resulting matrix and each cell is represented by a nxn matrix which is a product of two matrices. Therefore, to compute each “cell” we can use Strassen’s algorithm and this means an asymptotic running time of k2\*Θ(nlg7)= **Θ(k2nlg7)**

In Case 2, we compute the product of nxn matrices by k times (using Strassen’s) and, on top of this, we perform k-1 additions to obtain the result. Therefore, the asymptotic running time is dominated by the use of k times of Strassen’s for a total time of **Θ(knlg7).**

**4.2-7**

First, let’s see what some equivalent form of the multiplication might be.

(a+ib)(c+id)=ac+iad+ibc+i2bd =(ac-bd)+i(ad+bc) (because i2= -1)

We can “exhaust” 2 of the 3 allowed multiplications for calculating ac and bd.

On the other hand, we notice that ad+bc=ad+bc+ac+bd-ac-bd=(ad+ac)+(bc+bd)-(ac+bd)=

=a(c+d)+b(c+d)-(ac+bd)=(a+b)(c+d)-(ac+bd)

We notice that we can use what we calculate in the 2 multiplications, namely ac and bd.

In conclusion, for input a, b, c, and d, the 3 multiplications are:

a\*c, b\*d, and (a+b)\*(c+d) to obtain the required answer (note that we obtain both the real and imaginary components, namely ac-bd and ad+bc this way)

**4.5-3**

T(n)=T(n/2)+Θ(1)

We have a=1, b=2, f(n)= Θ(1) (thus it has degree 0 since it is a constant) and logba=log21=0

In this case, we can say that f(n) and nlogb(a) have the same rate of growth

Therefore, Case 2 of the Master Theorem is applicable => T(n)=Θ(nlogb(a)logn0+1)= Θ(n0logn)=>

T(n)= Θ(logn) which defines the tight bound for Binary Search algorithm.

**4.5-4**

T(n)=4T(n/2)+n2lgn

We first see whether Master Theorem can be applied and verify the conditions of the recurrence.

a=4, b=2, f(n)=n2lgn, logba=log24=2

It is obvious that f(n) dominates nlogb(a)=n2, since n2lgn = Ω(n2+ε) for a positive epsilon.

**Note** – here some might argue that f(n) is not “sufficiently” larger than n2 (i.e. by a polynomial degree ratio). However, since f(n) is Big-Omega(n2+ε), this suffices for this categorization.

In order to see whether Case 3 can be applied, we also need to check whether the regularity condition holds:

af(n/b)cf(n) where c is a positive constant <1.

4f(n/2)cf(n) => 4(n/2)2lg(n/2)cn2lgn or n2lg(n/2)cn2lgn or lg(n/2)clgn

This means that clg(n/2)/lgn=(lgn-1)/lgn =1 -1/lgn

For sufficiently large n, we have 1/lgn 🡪0, therefore 1- 1/lgn <1 (but remains positive)

In conclusion, we can find a constant c between [1-1/lgn,1) such that the regularity condition for Case 3 of the Master Theorem is satisfied.

In conclusion, we can say that T(n)=Θ(f(n))=Θ(n2lgn) is the tight upper bound for the provided recurrence

**Observation** – if the above solution is not preferred, then the upper bound of O(n2lgn) can be also obtained using structural induction method directly from the recurrence (i.e. assume T(n)kn2lgn and using the induction hypothesis in the recurrence for n/2; the inequality verifies easily this way).