6. **(a) Finding a tight big-O bound for matrix\_mult:**

Found this by looking at the code. Not including the constants,

…

for (int i=0; i<n; i++) { //🡨 n

for (int j=0; j<n; j++) { //🡨 n

result[i\*n+j] = 0;

for (int k=0; k<n; k++) { //🡨 n

result[i\*n+j] += a[i\*n+k] \* b[k\*n+j];

…

n\*n\*n = n3 so the big-O runtime is: O(n3).

**(b) Finding tight big-O bound for matrix\_exp1:**

This was also found by looking at the code.

…

for (int i=0; i<n; i++) { //🡨 n == n2

for (int j=0; j<n; j++) { //🡨 n

if (i==j) result[i\*n+j]=1;

else result[i\*n+j]=0;

}

}

return result;

}

double \* temp = matrix\_exp1(a,n,m-1); // recursion occurs m times

double \* result = matrix\_mult(a,temp,n); // calls O(n3)

…

if m = 0 then O(n2) 🡨 base case

if m>0 then O(mn3) 🡨 highest order so, this is the tight big-O bound.

**(c) Find tight big-O bound for matrix\_exp2:**

**…**

for (int i=0; i<n; i++) { //🡨 n ==n2

for (int j=0; j<n; j++) { //🡨 n

if (i==j) result[i\*n+j]=1;

else result[i\*n+j]=0;

}

}

return result;

}

double \* temp1 = matrix\_exp2(a,n,m/2); // recursion call takes lg(m)

…

if (m%2 == 0) {

temp2 = matrix\_mult(temp1,temp1,n); // calls O(n3)

} else {

temp2 = matrix\_mult(temp1,temp1,n); // calls O(n3)

temp1 = matrix\_mult(a,temp2,n); // calls O(n3)

…

So the tight bound is O(n3lg(m)).

(d) After playing around with the values of m and n, the time closest to 10 seconds was user: 9.908 s, where m = 5 n = 63. This was done on a Sony Vaio Pro 13 (Core i7-4500U, 1.80 GHz, 8.00 GB Ram).

(e) O(mn3), so to double if m is kept constant, we estimate that the value of n would be around 80. When we ran it, we coincidentally found 80 gave the user time of 20.824. So this was a very good estimate.

(f) My estimate would be that m= 10 would make it run twice as fast. This (g) a good guess as the runtime was 19.780s.

(g) O(n3lg(m)). I used the values n = 43 and m = 1000. The time was 10.068s.

(h) My estimate is 54, using (half of 433 and then cube root the result). This gives the result 19.896s so this was a good estimate.

(i) Using my calculation, to double the runtime you’d need an m of 1,000,000. However this was inaccurate, giving a runtime of 20.144s so this was an accurate estimate.