Assignment 1
CS 9566A

Paul Vrbik
250389673

September 29, 2009

Question 1 - Karatsuba's algorithm

```
karatsuba:=proc(f,g)
local df,dg,n,x,m,F0,G0,F1,G1,b,c,a;

printf("Call karatsuba( %a , %a )\n",f,g);

    if nops(indets(f) union indets(g)) > 1 then error "univariates only"; end if;

    df,dg:=degree(f),degree(g);

    n:=max(df,dg);

    if n<1 then
        return f*g;
    end if;

    n:=2^(trunc(log[2](n))+1);

    x:=indets(f) union indets(g); #assuming = indet(g)
    x:=x[1];
    m:=n/2;

    F0:=rem(f,x^m,x,'F1');
    G0:=rem(g,x^m,x,'G1');

    a:=procname(F0,G0);
    b:=procname(F1,G1);
    c:=procname(F0+F1,G0+G1);

    return b*x^n+(c-a-b)*x^m+a;
end proc:
```

```
> f:=1-2*x^3+3*x^3;
                                        3
                              f := 1 + x

> g:=1-x-2*x^2-x^3;
                                         2    3
                         g := 1 - x - 2 x  - x

> h:=karatsuba(f,g);
Call karatsuba( 1+x^3 , 1-x-2*x^2-x^3 )
Call karatsuba( 1 , 1-x )
```

```
Call karatsuba( 1 , 1 )
Call karatsuba( 0 , -1 )
Call karatsuba( 1 , 0 )
Call karatsuba( x , -x-2 )
Call karatsuba( 0 , -2 )
Call karatsuba( 1 , -1 )
Call karatsuba( 1 , -3 )
Call karatsuba( 1+x , -1-2*x )
Call karatsuba( 1 , -1 )
Call karatsuba( 1 , -2 )
Call karatsuba( 2 , -3 )
                        2         4      2       2
                h := (-x  - 2 x) x  + (-x  - 2) x  + 1 - x

> expand(h - f*g);
                                    0
```

Question 2 - Fourier Transform

FFT

```
FFT:=proc(f,n)
local w;
    w:=exp(2*I*Pi/n);
    return h_FFT(f,n,w,indets(f)[1]);
end proc;

h_FFT:=proc(f,n,w,x) #naive implemenation
local df,Feven,Fodd,V,Vp;
printf("Call FFT( %a , %a ) with w=%a\n",f,n,w);

    if n=1 then return [f]; end if;

    df:=degree(f,x);
    Feven:=add( x^i*coeff(f,x,2*i),i=0..trunc(df/2));
    Fodd:=add( x^i*coeff(f,x,2*i+1),i=0..trunc(df/2));

    V:=procname(Feven,n/2,w^2,x);
    Vp:=procname(Fodd,n/2,w^2,x);

#   return [f(w^0),...,f(w^(n-1))];
    return [seq( V[i mod n/2 + 1] + w^i*Vp[i mod n/2 + 1], i=0..n-1 )];
end proc:
```

```
> P:=1-x-2*x^2-3*x^3;
                                          2      3
                         P := 1 - x - 2 x  - 3 x

> n:=4:
> A:=FFT(P,n);
Call FFT( 1-x-2*x^2-3*x^3 , 4 ) with w=I
Call FFT( 1-2*x , 2 ) with w=-1
Call FFT( 1 , 1 ) with w=1
Call FFT( -2 , 1 ) with w=1
Call FFT( -1-3*x , 2 ) with w=-1
Call FFT( -1 , 1 ) with w=1
Call FFT( -3 , 1 ) with w=1
                      A := [-5, 3 + 2 I, 3, 3 - 2 I]

#check
> w:=exp(2*I*Pi/n):
> B:=[ seq( eval(P,x=w^i), i=0..n-1) ]:
> map( evalc, A-B );
                               [0, 0, 0, 0]
```

Inverse FFT

```
InvFFT:=proc(A,n)
local w,f,B;
    w:=exp(2*I*Pi/n);

    f:=add( A[i+1]*x^(i), i=0..nops(A)-1 );

    B:=FFT(f,n,1/w,x);
    B:=B/n;

    return evalc(add( B[i+1]*x^i, i=0..nops(B)-1 ));

end proc:
```

```
> InvFFT(B,n);
Call FFT( -5+(3+2*I)*x+3*x^2+(3-2*I)*x^3 , 4 ) with w=I
Call FFT( -5+3*x , 2 ) with w=-1
Call FFT( -5 , 1 ) with w=1
Call FFT( 3 , 1 ) with w=1
```

```
Call FFT( 3+2*I+(3-2*I)*x , 2 ) with w=-1
Call FFT( 3+2*I , 1 ) with w=1
Call FFT( 3-2*I , 1 ) with w=1
                                   2    3
                         1 - 3 x - 2 x  - x
```

Question 3

1. $S(n) = 3S(n/2) + n^{1.2}$. For the master theorem $a = 3, b = 2, c = 1, k = 1.2$ where $\log_2 3 = 1.58 > 1.2 = k$. Therefore $\mathbf{S(n) = O(n^{\log_2 3})}$ as given by the master theorem.

2. $S(n) = 4S(n/2) + n^{1.75}$. For the master theorem $a = 4, b = 2, c = 1, k = 1.75$ where $\log_2 4 = 2 > 1.75 = k$. Therefore $\mathbf{S(n) = O(n^{\log_2 4}) = O(n^2)}$ as given by the master theorem.

3. $S(n) = 3S(n/2) + n^{1.5}$. For the master theorem $a = 3, b = 2, c = 1, k = 1.5$ where $\log_2 3 = 1.58 > 1.5 = k$. Therefore $\mathbf{S(n) = O(n^{\log_2 3})}$ as given by the master theorem.

Question 4

Let $T(1) = 1$ and $T(n) = 3T(n/2) + \ell n$ with n a power of two and ℓ constant.

$$\begin{aligned}
T(2) &= 3T(n/2) + 2\ell = 3T(1) + 2\ell = 3 + 2\ell \\
T(4) &= 3T(4/2) + 4\ell = 3T(2) + 4\ell = 9 + 6\ell + 4\ell = 9 + 10\ell \\
T(8) &= 3T(8/2) + 8\ell = 3T(4) + 8\ell = 27 + 30\ell + 8\ell = 27 + 38\ell
\end{aligned}$$

Prove (by induction) that

$$T(2^n) = 3^n + 2(3^{n-1} + 2 \cdot 3^{n-2} + \cdots + 2^{n-2} \cdot 3 + 2^{n-1})\ell. \tag{1}$$

The base case has already been given in the first part of this question. For the induction hypothesis assume

$$T(2^{n-1}) = 3^{n-1} + 2(3^{n-2} + 2 \cdot 3^{n-3} + \cdots + 2^{n-3} \cdot 3 + 2^{n-2})\ell$$

and proceed in the natural way:

$$\begin{aligned}
T(2^n) &= 3T(2^n/2) + \ell 2^n && \text{by definition} \\
&= 3T(2^{n-1}) + \ell 2^n \\
&= 3(3^{n-1} + 2(3^{n-2} + 2 \cdot 3^{n-3} + \cdots + 2^{n-3} \cdot 3 + 2^{n-2})\ell) + \ell 2^n && \text{by assumption} \\
&= 3^n + 2(3^{n-1} + 2 \cdot 3^{n-2} + \cdots + 2^{n-3} \cdot 3^2 + 2^{n-2} \cdot 3)\ell + \ell 2^n \\
&= 3^n + 2(3^{n-1} + 2 \cdot 3^{n-2} + \cdots + 2^{n-2} \cdot 3 + 2^{n-1})\ell.
\end{aligned}$$

Therefore (1) has been proved.

It is clear that RHS of (1) is dominated by 3^n in the limit. $T(2^n) = O(3^n)$ is a direct consequence of this.

Question 5 - Uniqueness of Quotient and Remainder

Suppose that we have q, g, q' and r' all generated by Euclidean division, satisfying

$$f = q \cdot g + r = q' \cdot g + r'$$

where $q \neq q'$ and $r \neq r'$. By the division algorithm neither r or r' has a term divisible by $\mathrm{LT}(g)$. However, $(q - q') \cdot \mathrm{LT}(g)$ does (any term of $(q - q')g_1$). Since $r - r' = (q - q') \cdot g$ it must also be the case that $r - r'$ has a term divisible by $\mathrm{LT}(g)$. This is only possible when $r - r' = 0$ contradicting our assumption. Therefore q and r must be unique.

Question 6 - Division Rules

Addition

Show:

$$(A_1 \text{ rem } B) + (A_2 \text{ rem } B) = (A_1 + A_2) \text{ rem } B.$$

Proof. By the division algorithm we have unique $q_1, q_2, r_1 = A_1 \text{ rem } B$ and $r_2 = A_2 \text{ rem } B$ such that $A_1 = q_1 B + r_1$ and $A_2 = q_2 B + r_2$. Since adding A_1 and A_2 gives

$$A_1 + A_2 = (q_1 + q_2)B + (r_1 + r_2)$$

$(r_1 + r_2)$ must be the unique remainder when dividing $A_1 + A_2$ by B. That is, $(A_1 + A_2) \text{ rem } B = r_1 + r_2 = (A_1 \text{ rem } B) + (A_2 \text{ rem } B)$. □

Multiplication

Show:

$$((A_1 \text{ rem } B) \times (A_2 \text{ rem } B)) \text{ rem } B = (A_1 \times A_2) \text{ rem } B.$$

Proof. By the division algorithm we have unique $q_1, q_2, r_1 = A_1 \text{ rem } B$ and $r_2 = A_2 \text{ rem } B$ such that $A_1 = q_1 B + r_1$ and $A_2 = q_2 B + r_2$. Multiplying A_1 by A_2 gives

$$\begin{aligned} A_1 \times A_2 &= q_1 q_2 B^2 + r_2 q_1 B + r_1 q_2 B + r_1 r_2 \\ &= q_1 q_2 B^2 + r_2 q_1 B + r_1 q_2 B + (r_1 r_2 \text{ div } B)B + r_1 r_2 \text{ rem } B \\ &= (q_1 q_2 B + r_2 q_1 + r_1 q_2 + r_1 r_2 \text{ div } B)B + r_1 r_2 \text{ rem } B. \end{aligned}$$

By the division algorithm $r_1 r_2 \text{ rem } B$ must be the unique remainder when dividing $A_1 \times A_2$ by B. That is, $(A_1 \times A_2) \text{ rem } B = r_1 r_2 \text{ rem } B = ((A_1 \text{ rem } B) \times (A_2 \text{ rem } B)) \text{ rem } B$. □

Question 7 - Modular Multiplication

First we compute

$$\begin{aligned} c_0 + c_1 x &= (a_0 + a_1 x)(b_0 + b_1 x) \text{ rem } (x^2 + 2) \\ &= a_0 b_0 + (a_0 b_1 + a_1 b_0)x + a_1 b_1 x^2 \text{ rem } (x^2 + 2) \\ &= (a_0 b_0 - 2a_1 b_1) + (a_0 b_1 + a_1 b_0)x \end{aligned}$$

and copy the trick of Karatsuba to do

$$\begin{aligned} c_0 &= a_0b_0 - a_1b_1 - a_1b_1 \\ c_1 &= (a_0 + a_1)(b_0 + b_1) - a_0b_0 - a_1b_1 \\ &= (a_0 + a_1)(b_0 + b_1) + a_1b_1 - c_0 \end{aligned}$$

which establishes the required modular multiplication using only three products (a_0b_0, a_1b_1 and $(a_0 + a_1)(b_0 + b_1)$).

Question 8 - Alternative Quadratic Multiplication

Let $f = f_0 + f_1x + f_2x^2$ and $g = g_0 + g_1x + g_2x^2$ and

$$\begin{aligned} h &= fg = h_0 + h_1x + h_2x^2 + h_3x^3 + h_4x^4 \\ &= f_0g_0 + (f_0g_1 + f_1g_0)x + (f_0g_2 + f_1g_1 + f_2g_0)x^2 + (f_1g_2 + f_2g_1)x^3 + f_2g_2x^4. \end{aligned}$$

We show:

$$\begin{aligned} H_0 &= F_0G_0 = f(0)g(0) = f_0g_0 = h(0) \\ H_1 &= F_1G_1 = f(1)g(1) \\ &= f_0g_0 + f_0g_1 + f_1g_0 + f_0g_2 + f_1g_1 + f_2g_0 + f_1g_2 + f_2g_1 + f_2g_2 \\ &= h(1) \\ H_{-1} &= F_{-1}G_{-1} = f(-1)g(-1) \\ &= f_0g_0 - (f_0g_1 + f_1g_0) + (f_0g_2 + f_1g_1 + f_2g_0) - (f_1g_2 + f_2g_1) + f_2g_2 \\ &= h(-1) \\ H_{x^2+2} &= F_{x^2+2}G_{x^2+2} \text{ rem } (x^2 + 2) \\ &= (f_0 - 2f_2 + f_1x)(g_0 - 2g_2 + g_1x) \text{ rem } (x^2 + 2) \\ &= f_0g_0 - 2f_0g_2 - 2f_2g_0 + 4f_2g_2 + (f_0g_1x - 2f_2g_1 + f_1g_0 - 2f_1g_2)x + f_1g_1x^2 \text{ rem } (x^2 + 2) \\ &= f_0g_0 + (f_0g_1 + f_1g_0)x - 2(f_0g_2 - f_1g_1 + f_2g_0) - 2(f_1g_1 + f_2g_1)x + 4f_2g_2 \\ &= h \text{ rem } (x^2 + 2). \end{aligned}$$

To recover h from $H_0, H_1, H_{-1}, H_{x^2+2}$ we solve:

$$\begin{aligned} H_0 &= h_0 \\ H_1 &= h_0 + h_1 + h_2 + h_3 + h_4 \\ H_{-1} &= h_0 - h_1 + h_2 - h_3 + h_4 \\ H_{x^2+2} &= (h_0 - 2h_2 + 4h_4) + (h_1 - 2h_3)x = c_0 + c_1x \end{aligned}$$

which is equivalent to solving

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 \\ 1 & 0 & -2 & 0 & 4 \\ 1 & 0 & 0 & -2 & 0 \end{bmatrix} \begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \\ h_4 \end{bmatrix} = \begin{bmatrix} H_0 \\ H_1 \\ H_{-1} \\ c_0 \\ c_1 \end{bmatrix}$$

and can be done with three divisions by two (done in constant time via bit shifting).

We require one multiplication (each) to get H_0, H_1, H_{-1} and three to do H_{x^2+2} (by Question 7.) **for a total of six multiplications**.

To apply recursively we observe that any polynomial $f = f_0 + f_1 x + \cdots + f_n x^n \in \mathcal{R}[x]$ can be written as

$$\begin{aligned} F &= (f_0 + f_1 x + \cdots + f_{m-1}x^{m-1}) + (f_m + \cdots + f_{2m-1}x^{m-1})X + (f_{2m} + f_n x^{n-2m})X^2 \\ &= a_0 + a_1 X + a_2 X^2 \end{aligned}$$

where $X = x^m$, $m = \lceil n/3 \rceil$. So now we may use our multiplication scheme to multiply any two polynomials $f, g \in \mathcal{R}[x]$ by first representing them as quadratics polynomials (as above) and recursively applying the scheme to resolve the coefficient products (which will be polynomials of degree less than m). This recursion is guaranteed to terminate as the degrees of the coefficients at each step form a strictly descending chain.

Suppose that $M(n)$ is the number of products required to multiply two polynomials of degree less than n. For this algorithm, at each step, we do six multiplications of polynomials one third of the original degree. More explicitly we have $M(n) = 6M(n/3)$. We can use the master theorem which implies, in this case, that **the complexity of this scheme is $\mathbf{O(n^{\log_3 6})}$**.

Question 9

$$\text{time to complete } < (20 \text{ min}) \times (8 \text{ questions}) = 2.7 \text{ hours}$$