Assignment #5, Question 3

Michael Monagan, April 2013.

This is the "slow" algorithm. It computes U^2 each time round the loop and divides e = a-U^2 by p^k each time round the loop. A big x big multiplication and a big by big division.

If the input a has n digits (base p) then these operations make the algorithm O(n^3) assuming classical algorithms for integer multiplication and division and assuming the prime p is of fixed size.

```
> restart;
  ZSQRT := proc(a,u0,p) local U,pk,k,ek,ck,uk,i;
      U := mods(u0,p);
      i := modp(1/(2*u0),p);
      pk := p;
      for k do
          ek := a - U^2;
          if ek = 0 then RETURN(U); fi;
          if ek < 0 then RETURN(FAIL); fi;
          ck := iquo(ek,pk); # should divide exactly with 0 remainder
          uk :=  mods( ck*i, p );
          U := U + uk * pk;
          pk := p*pk;
      od;
  end:
```

This is the "good" algorithm. It computes ck = ek/p^k efficiently.

It replaces the computation of $c_k = \frac{e_k}{p^k} = \frac{a-(U^{(k)})^2}{p^k}$

$$= \frac{\left(a-(U^{(k-1)})^2 - 2\cdot U^{(k-1)}\cdot u_{k-1}\cdot p^{k-1} - {u_{k-1}}^2\cdot p^{(2\cdot k-2)}\right)}{p^k}$$

$$= \frac{\left(c_{k-1} - 2\cdot u_{k-1}\cdot U^{(k-1)} - u_{k-1}^2\cdot p^k\right)}{p}$$

```
> ZSQRT_FASTER := proc(a,u0,p) local U,ck,uk,pk,k,i,r;
      U := 0;
      i := mods(1/(2*u0),p);
      ck := a;
      pk := 1;
      uk := mods(u0,p);
      for k from 0 do
          #ck := iquo(ck-2*uk*U-uk^2*pk,p,'r');
          ck := iquo(ck-uk*(2*U+uk*pk),p,'r');
          if r<>0 then ERROR(bug) fi;
          U := U + uk*pk;
          if ck=0 then RETURN(U); fi;
          if ck<0 then RETURN(FAIL); fi;
          uk := mods( ck*i, p );
```

```
          pk := pk*p;
       od;
  end:
> R := rand(10^5000):
> b := R():
  a := b^2:
  length(a);
```

$$9998$$

```
> p := 997;
```

$$p := 997$$

```
> u0 := b mod p;
```

$$u0 := 324$$

```
> st := time(): c := ZSQRT(a,u0,p): time()-st;
```

$$0.220$$

```
> st := time(): d := ZSQRT_FASTER(a,u0,p): time()-st;
```

$$0.037$$

```
> c-d;
```

$$0$$

```
> p := prevprime(2^31);
```

$$p := 2147483647$$

```
> u0 := b mod p;
```

$$u0 := 1460298156$$

```
> tslowprev := time( ZSQRT(a,u0,p) );
```

$$tslowprev := 0.090$$

```
> tfastprev := time( ZSQRT_FASTER(a,u0,p) );
```

$$tfastprev := 0.013$$

```
> for i to 5 do
      tslow := time( ZSQRT(a,u0,p) );
      tfast := time( ZSQRT_FASTER(a,u0,p) );
      printf(" length(a)=%6d  slow=%7.3fs  %6.2fx  fast=%6.3fs
  %6.2fx\n",
            length(a),tslow,tslow/tslowprev,tfast,tfast/tfastprev);
      tslowprev,tfastprev := tslow,tfast;
      a,b,u0 := a^2,a,a mod p;
  od:
 length(a)=  9998  slow=  0.055s    0.61x  fast= 0.009s    0.69x
 length(a)= 19995  slow=  0.241s    4.38x  fast= 0.049s    5.44x
 length(a)= 39990  slow=  1.175s    4.88x  fast= 0.216s    4.41x
 length(a)= 79979  slow=  6.081s    5.18x  fast= 0.853s    3.95x
 length(a)=159957  slow= 27.824s    4.58x  fast= 3.228s    3.78x
```

The slow algorithm is not cubic because Maple is using fast algorithms for big integer operations not classical algorithms.

The fast algorithm may be quadratic. I did some more to see how close to a factor of 4 we get.

```
> for i to 2 do
      tfast := time( ZSQRT_FASTER(a,u0,p) );
      printf(" length(a)=%7d  fast=%6.3fs  %6.2fx\n",
             length(a),tfast,tfast/tfastprev);
      tfastprev := tfast;
      a,b,u0 := a^2,a,a mod p;
  od:
 length(a)= 319913  fast=12.598s    3.90x
 length(a)= 639826  fast=50.185s    3.98x
```

Well, it's getting close to a factor of 4 times longer when we double the length of the inputs.