Aufgabenblatt 6: Sortieren durch Einfügen

```
> restart;
```

Einfügen in einen sortierten Binärbaum

Die Prozedur insert übernehmen wir aus der Zentralübung:

```
> insert := proc(baum::evaln,k::numeric)
    if eval(baum) = LEER
       then baum := [k, [LEER,LEER]];
    else
       if eval(baum)[1] > k
       then
          insert(baum[2,1],k);
       else
          insert(baum[2,2],k);
       end if;
    end if;
    eval(baum);
  end:
```

Die folgende Prozedur erhält eine Liste als Parameter und liefert einen sortierten Binärbaum als Ergebnis:

```
> listebaum := proc(liste)
     local i,baum;
     baum := LEER;
     for i in liste do
        insert(baum,i):
     end do;
     baum;
  end;
```

listebaum := **proc**(*liste*)

local *i*, *baum*;

baum := *LEER*; **for** *i* **in** *liste* **do** insert(*baum*, *i*) **end do**; *baum*

end proc

```
> zufall := rand(1..100):
> r := listebaum( [ seq( zufall(), i=1..10) ] );
```

r := [82, [[71, [

[64, [[39, [[22, [[10, [*LEER*, *LEER*]], *LEER*]], *LEER*]], [69, [*LEER*, *LEER*]]]],

[77, [*LEER*, *LEER*]]]]], [98, [[86, [*LEER*, *LEER*]], *LEER*]]]]

```
>
```

Auslesen des Binärbaums:

Die nächste Prozedur erhält einen sortierten Binärbaum und erzeugt aus diesem eine sortierte Liste.
Die wesentliche (rekursive!) Idee ist:

- schreibe erst alle Elemente des linken Teilbaums in die Liste
- hänge das Knotenelement hinten an die Liste an
- hänge die Elemente des rechten Teilbaums hinten an die Liste an

Das liefert folgende Prozedur:

```
> baumliste := proc(baum)
    if baum = LEER
    then return [];
    else
       return [ op( baumliste(baum[2,1]) ),
                baum[1],
                op( baumliste(baum[2,2]) ) ];
    end if;
  end;
```

baumliste := **proc**(*baum*)

if *baum* = *LEER* **then return** [ ]

else return [op(baumliste(*baum*[2, 1])), *baum*[1], op(baumliste(*baum*[2, 2]))]

end if

end proc

```
> baumliste(r);
```

[10, 22, 39, 64, 69, 71, 77, 82, 86, 98]

```
>
```

Sortieren:

Hintereinander ausgeführt ergeben die beiden Funktionen einen Sortieralgorithmus:

```
> baumsort := proc(liste)
     baumliste(listebaum(liste));
  end;
```

baumsort := **proc**(*liste*) baumliste(listebaum(*liste*)) **end proc**

```
> baumsort( [ seq( zufall(), i=1..10) ] );
```

[8, 17, 50, 56, 58, 61, 62, 64, 75, 86]

Laufzeiten von baumsort

Zunächst generieren wir eine Funktion `zufall`, die Zufallszahlen zwischen 1 und 10000 liefert:

```
> zufall := rand(1..10000);
```

zufall := **proc**()

local *t*;

global *_seed*;

_seed := irem(427419669081 ∗ *_seed*, 999999999989); *t* := *_seed*; irem(*t*, 10000) + 1

end proc

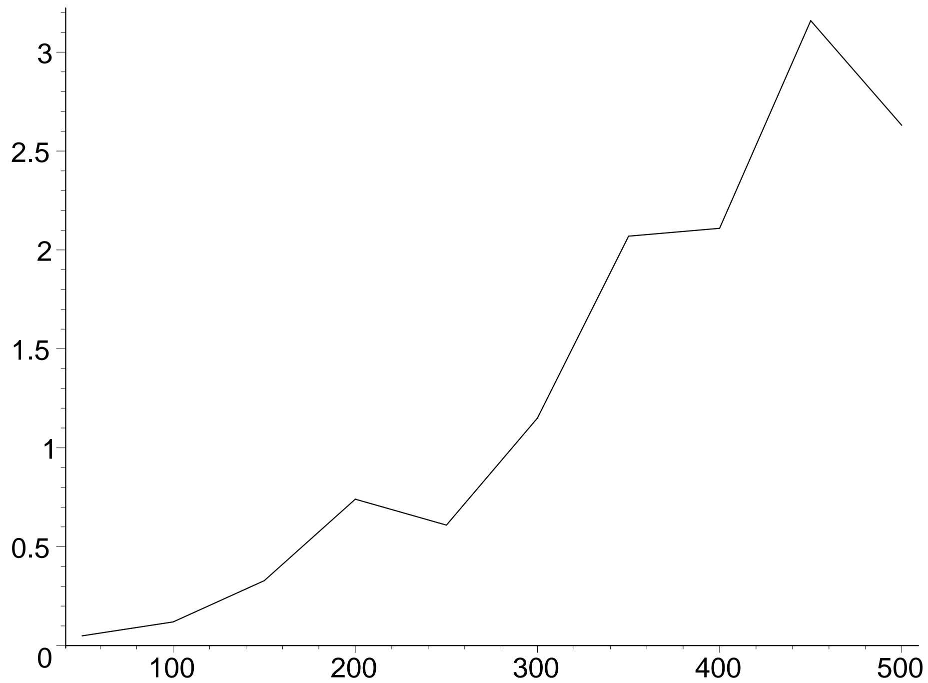
Die Prozedur laufzeit entspricht denen aus dem vorherigen Arbeitsblatt

```
> laufzeit := proc(n)
    local j,liste, mtime, feld;

    liste := [ seq(zufall(j), j=1..n) ];
    mtime := time(baumsort(liste));

    [n,mtime];
  end:
> iz := [seq(laufzeit(i*50),i=1..10)];
```

$iz := [[50, .050], [100, .120], [150, .329], [200, .740], [250, .609], [300, 1.150],$
$[350, 2.070], [400, 2.109], [450, 3.159], [500, 2.630]]$

Der Laufzeit-Plot sieht eher nach einem quadratischen Verhalten aus, als nach dem erwarteten $n \log(n)$ Verhalten. Wichtig wären deshalb Kenntnisse, wie gut Maple die Baumoperationen durch die Listen implementiert.

```
> with(plots):
> pointplot(iz,style=line);
Warning, the name changecoords has been redefined
```




```
>
```