A list of positive integers is called a partition of natural number \( n \) if the sum of its items is \( n \). Write a program to find

(a) a list of all partitions of a given natural \( n \). For example, if \( n=3 \) then an acceptable answer is \([3]; [1; 2]; [2; 1]; [1; 1; 1]\).

§ Part (a) is subsumed by part (c).

(b) a list of all sorted partitions of a given natural \( n \). For example, if \( n=3 \) then an acceptable answer is \([3]; [1; 2]; [1; 1; 1]\).

§ Part (b) is subsumed by part (d).

(c) the sorted list of all partitions of a given natural \( n \). For example, if \( n=3 \) then the answer is \([1; 1; 1]; [1; 2]; [2; 1]; [3]\).

§ Given a partition, to get the next partition: cut off the final item; increase the new final item by 1 and call this \( f \); join as many \( fs \) as possible without making the sum exceed \( n \).

(d) the sorted list of all sorted partitions of a given natural \( n \). For example, if \( n=3 \) then the answer is \([3]; [1; 2]; [1; 1; 1]\).

§ Given a sorted partition, to get the next sorted partition: cut off the final item; increase the sorted list of all sorted partitions of a given natural \( n \). For example, if \( n=3 \) then the answer is \([3]; [1; 2]; [2; 1]; [1; 1; 1]\).

Part (b) is subsumed by part (d).

\( A \) is a partition

\( R \) is defined as

\[
R = (L' \text{ is the sorted list of all partitions of } n)
\]

\[
= (\forall i, j: 0,\ldots,\#L': i < j \Rightarrow L'i < L'j) \wedge (\forall i: 0,\ldots,\#L':(\Sigma L'i) = n) \wedge (\forall Q: [*(\text{nat}+1)]: (\Sigma Q) = n \Rightarrow Q: L'(0,\ldots,\#L'))
\]

Introduce partition variable \( P: [*(\text{nat}+1)] \) and define

\[
A = (P \text{ is a partition}) \wedge (L \text{ is the sorted list of all partitions of } n \text{ that precede } P)
\]

\[
= (\Sigma P) = n \wedge (\forall i, j: 0,\ldots,\#L: i < j \Rightarrow Li < Lj < P) \wedge (\forall i: 0,\ldots,\#L:(\Sigma Li) = n) \wedge (\forall Q: [*(\text{nat}+1)]: (\Sigma Q) = n \& Q < P \Rightarrow Q: L(0,\ldots,\#L))
\]

Now the refinements.

\[
R \iff L := [n!]. \ P := [n*1]. \ A \Rightarrow R
\]

\[
A \Rightarrow R \iff L := L.; [P].
\]

\[
\text{if } \#P < 2 \text{ then } ok
\]

\[
\text{else } P := P[0;\ldots,\#P-2] :: [P[\#P-2]+1; (P[\#P-1] -1)*1]. \ A \Rightarrow R \ fi
\]

Here is a program using loops (Chapter 5); instead of gathering the partitions into a list, I print them (let's say !x prints the value of \( x \) and \( ?x \) reads into the variable \( x \)).

\[
\text{var } n, m: \text{int}; \ n \text{ is the length of } P \text{ and } m \text{ is a temporary}
\]

\[
!n"=". \ ?n.
\]

\[
\text{var } P: [n*\text{int}].
\]

\[
\text{for } i := 0,\ldots,n \text{ do } P i := 1 \text{ od}.
\]

\[
\text{do for } i := 0,\ldots,n \text{ do } !P i, "\" \text{ od.} \newline
\]

\[
\text{exit when } n < 2.
\]

\[
P(n-2) := P(n-2)+1.
\]

\[
m := n-1. \ n := m + P(m)-1.
\]

\[
\text{for } i := m,\ldots,n \text{ do } P i := 1 \text{ od od}
\]

The exact execution time is obtained by putting \( t := t + 1 \) in front of the recursive call, and replace \( R \) by \( t' = t + 2^n-1 \) replace \( A \Rightarrow R \) by \( t' = t + \Sigma i: 1,\ldots,\#P: 2^{\Sigma P[i:1;\#P]}-1 \).
sum too big; increase the final item to get the right sum. This solution is very similar to part (c), but the assignment
\[
P := P[0;..#P–2] ;; [P(#P–2)+1 ; (P(#P–1)–1)*1]
\]
has to be replaced by
\[
P := P[0;..#P–2] ;; [(\text{div } d f – 1)\% f : f + \text{mod } d f]
\]
and we have to modify \( R \) and \( A \).