instruction to candidates
(time allowed 2 hours)

- candidates will be assessed on their best four answers
- if you attempt to answer to more than the required number of questions, the marks awarded for the excess questions will be discarded (starting with your lowest mark)

Question 1

Given the following recursive function:

```
function A (n, k : in NATURAL)
    return POSITIVE is
begin
    if n = 0 or k = 0
        then return n + k;
        else return A(n − 1, k − 1) + 1;
    end if;
end A;
```

1.A Answer the following questions:

(i) List all recursive calls and values returned within each call when function A is initially called with $n = 7$ and $k = 4$. **[3 marks]**

(ii) What arithmetic function is computed by function A? **[3 marks]**

(iii) What is the exact time complexity (number of recursive calls) of function A, for arbitrary values of parameters n and k? **[3 marks]**

(iv) What happens when the line: **then return** $n + k$;
is substituted by the line: **then return** 0; **[3 marks]**

(v) What happens when the line: **else return** $A(n - 1, k - 1) + 1$;
is substituted by the line: **else return** $A(n, k - 1) + 1$; **[3 marks]**

1.B Write two recursive functions in Ada computing value 2^n (for $n \geq 1$) and having time complexities:

(i) $\Theta(2^n)$ and **[3 marks]**

(ii) $O(n)$ **[2 marks]**

Question 2

2.A Study the following *PStack* package. As you can see there are places numbered 1..5, where the missing code is labelled **missing code**.

```
package PStack is
1  –  missing code  –
        procedure push(s : in out STACK; r : in FLOAT);
        procedure pop(s : in out STACK; r : out FLOAT);
2  –  missing code  –
private
    type STACK_EL;
3  –  missing code  –
end PStack;
package body PStack is
    type STACK_EL is
        record
4  –  missing code  –
            next : STACK;
        end record;
    procedure push(s : in out STACK; r : in FLOAT) is
    begin
        s := new STACK_EL(s, r);
    end push;
    procedure pop(s : in out STACK; r : out FLOAT) is
    begin
        r := s.num;
        s := s.next;
    end pop;
5  –  missing code  –
    begin
        return s = null;
    end empty;
end PStack;
```

The missing lines are given below and are labelled with letters A to E. Work out where the missing code should go and then give the answer listing all matching pairs of the form (missing line number, appropriate letter).

A **function** $empty(s : STACK)$ **return** $BOOLEAN$ **is**
B **type** $STACK$ **is access** $STACK\_EL$;
C $num : FLOAT$;
D **type** $STACK$ **is limited private**;
E **function** $empty(s : STACK)$ **return** $BOOLEAN$; **[5 marks]**

2.B Briefly outline three advantages of using packages. **[3 marks]**

2.C Explain the function of package specification and package bodies, using as an example the *PStack* package. **[4 marks]**

2.D List two advantages and two disadvantages of using pointers. **[4 marks]**

2.E Decide whether *stack* as a data structure is more appropriate for recursive or iterative methods. **[4 marks]**

Question 3

Consider the following arithmetic expression involving only positive numbers from the range 1..8 which is given in **postfix** notation:

$$1 \ 5 \ 7 \ + \ 6 \ 2 \ * \ 8 \ - \ * \ +$$

3.A Write a piece of code in Ada evaluating postfix expressions using a stack structure. Assume that you have given functions:
$pop(E)$ – removes the top of the stack to variable E, $pop(E)$ used on empty stack generates $ERROR$ message,
$push(E)$ – places the content of variable E on the top of the stack,
$empty$ – returns $true$ when stack is empty, $false$ otherwise,
$get(C)$ – reads the next input character to variable C,
$num(C)$ – changes character in C to the corresponding number.
Make sure that your program reports also invalid expressions. **[10 marks]**

3.B Calculate the value of the expression. **[2 marks]**

3.C Create an evaluation tree for the expression and explain which traversing order corresponds to the postfix notation. **[3 marks]**

3.D Create a bracket expression representing the structure of the evaluation tree from 3.C. **[3 marks]**

3.E List nodes of the evaluation tree (3.C) in prefix order. **[2 marks]**

Question 4

4.A List three applications of heaps. **[3 marks]**

4.B Let $T(1..11) = (1, 4, 2, 5, 8, 9, 3, 7, 6, 11, 10)$.
(i) Do elements of T form a structure of a heap? To answer this question draw a binary tree formed by elements of T and check whether each internal node satisfies the heap property. **[5 marks]**
(ii) What is the time complexity of the heap construction? **[1 mark]**
(iii) What is the time complexity of the *HeapSort* algorithm? **[1 mark]**

4.C Which function is larger for almost all n (i.e. which function has higher order):

(i)

$$n^{\frac{3}{4}} \qquad \frac{n}{\log(n)}$$

(ii)

$$\log(n!) \qquad 2n$$

(iii)

$$n^{\log\log(n)} \qquad 2^{4\log(n)}$$

[6 marks]

4.D What is the value of:

$$\log^*(15) =$$

[2 marks]

4.E Which exact complexity is better for realistic values of parameter n,

$$100n \quad \textit{or} \quad \frac{n\log(n)}{2}?$$

Justify shortly all your answers. **[2 marks]**

Question 5

Given onedimensional table $T : TABLE$, where:

type $TABLE$ **is array** $(1..N)$ of FLOAT;

and T is filled with possibly both positive and negative real numbers.

5.A Design and write an Ada procedure finding a segment (slice) $T(i..j)$ in table T (for $1 \leq i \leq j \leq n$), s.t. $\sum_{k=i}^{j} T(k)$ is maximised. Ada procedure should report both left and right end of the requested segment as well as the sum of its elements. **[10 marks]**

5.B Give a proof of correctness of your procedure from 5.A **[5 marks]**

5.C What is the time complexity of your procedure from 5.A? Give some justification. **[5 marks]**

Question 6

Given

type $TABLE$ **is array** $(1..N)$ **of** $FLOAT$;
type $LONG\_TABLE$ **is array** $(1..2N)$ **of** $FLOAT$;
$T_1, T_2 : TABLE$;
$T_3 : LONG\_TABLE$;

Elements in tables T_1 and T_2 are sorted in strictly increasing order (i.e. $T_k(i) < T_k(j)$, for any $1 \leq i < j \leq N$ and $k = 1, 2$).

6.A Create an Ada procedure which fills table T_3 with elements from T_1 and T_2 in nondecreasing order. Please comment on complexity of your solution. **[5 marks]**

6.B What is the time complexity of the merging problem from 6.A when one of the input tables (T_1 or T_2) is not sorted. Can we achieve time complexity $O(N)$? If your answer is negative, please comment on why you think it is not possible. **[5 marks]**

6.C Compare time complexities of two sorting procedures *SelectionSort* and *MergeSort* when the unit operation is defined as:
(i) one comparison of two numbers **[2 marks]**
(ii) one swap of two numbers **[2 marks]**

6.D Which sorting procedure was used to sort 5 numbers from range 1..5, where sequence below starts with the original order of numbers and then continues with orders after consecutive stages of the sorting procedure:
$32514 -> 12534 -> 12534 -> 12354 -> 12345$?
Briefly justify your answer. **[2 marks]**

6.E What is the time complexity of the *CountSort* algorithm? Why and when does it beat other more general sorting algorithms? **[4 marks]**

Answers to question 1

1.A

(i) The calls and values are as follows:

$A(7,4) = 7 \qquad = A(6,3) + 1 = 6 + 1 = 7$
$A(6,3) = 6 \qquad = A(5,2) + 1 = 5 + 1 = 6$
$A(5,2) = 5 \qquad = A(4,1) + 1 = 4 + 1 = 5$
$A(4,1) = 4 \qquad = A(3,0) + 1 = 3 + 1 = 4$
$A(3,0) = 3$

(ii) Function A computes the maximum.

(iii) There are $min(n,k)+1$ calls of procedure A including first call $A(n,k)$ and last call when one of the parameters becomes 0.

(iv) In this case function will compute the minimum.

(v) In this case function will compute the sum.

1.B

(i) Function with exponential complexity:

```
function exp(n : in NATURAL)
      return NATURAL is
begin
      if n = 0
            then return 1;
            else return exp(n − 1) + exp(n − 1);
      end if;
end exp;
```

(ii) Function with linear complexity:

```
function lin(n : in NATURAL)
      return NATURAL is
begin       if n = 0
            then return 1;
            else return 2 ∗ lin(n − 1);
      end if;
end lin;
```

Answers to question 2

2.A The matching pairs are as follows:
$(1, D), (2, E), (3, B), (4, C), (5, A)$

2.B Three advantages of using packages:
1. Bringing together related parts of a program into a logical unit.
2. A package has its own environment containing local types as well as realisation of functions and procedures, which are separated from the rest of the program.
3. A package is selfcontained so it can be developed and compiled alone.

2.C Each package consists in two separate parts: specification (which is visible) and realisation (which is hidden). The visible part is accessible (from outside) and it contain declaration of types, constants, subroutines, tasks and other packages. The second part, the body of the package, is hidden, and it contains a realisation of types and subroutines defined in the specification part. Since this part is hidden the user of the package can not access the local data declared in the realisation part.
The package defined in 2.A is an implementation of a stack. In its specification part one can find declaration of 3 standard stack functions: *pop*, *push*, and *empty* which will be available for a package user. Moreover specification part contains definition of stack types. But the user of the package does not know what a stack really looks like. The only thing a user may do is declare declare objects of type $STACK$ and have parameters of type $STACK$ in calls to subprograms.

2.D Two advantages: (i) dealing with data structures which size is not known in advance and (ii) dealing with data structures with more complex topology. Two disadvantages: (i) implementing data structures in arrays is more time efficient and (ii) garbage collection is more complex.

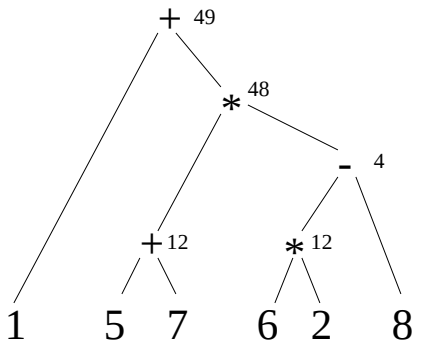
2.E A stack is a data structure which is used in the realisation of a recursive methods. A use of the stack can be hidden, when a programmer writes a recursive procedure and the stack of recursive calls is handled by the programming language. Or use of the stack can be visible when a programmer implements recursive procedure in the language not allowing recursive calls.

Answers to question 3

3.A The fragment of Ada code :
$x : CHARACTER; a, b : INTEGER;$ ...
while $get(x)$ **loop**
    case x **is**
        when $'1'..'8' => push(num(x));$
        when $'*' => pop(a); pop(b); a := a * b; push(a);$
        when $'+' => pop(a); pop(b); a := a + b; push(a);$
        when $'-' => pop(a); pop(b); a := a - b; push(a);$
    end case;
end loop;
$pop(a)$; **if** $empty$ **then return** a; **else** $ERROR$; **end if**;

3.B 49

3.C Postfix notation represents postorder traversing of the evaluation tree.



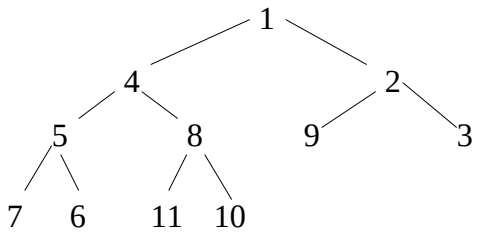
3.D (()((()())(((())())())))

3.E $+1 * +57 - *628$

Answers to question 4

4.A 1. *Selection*, 2. *Sorting*, 3. *Compression* (Huffman coding)

4.B (i) Yes, this is a heap, all children are greater then their parents.



(ii) linear $O(n)$, (iii) $O(n \log(n))$

4.C
(i) $n^{\frac{3}{4}} < \frac{n}{\log(n)}$ – because $n^{\frac{1}{4}} > \log(n)$,
(ii) $\log(n!) > 2n$ – because $\log(n!) = \Omega(n \log(n))$,
(iii)$n^{\log\log(n)} > 2^{4\log(n)}$ – because $2^{4\log(n)} = n^4$ and $n^{\log\log(n)}$ grows faster than any polynomial

4.D $log^*(15) = 3$

4.E $100n > \frac{n\log(n)}{2}$ for all realistic values of n, since inequality holds for all $n < 2^{200}$ which is much larger then the number of atoms in the solar system.

Answers to question 5

5.A The problem can be solved in linear time but also quadratic time solution will be granted 10 marks. The cubic time solution will get at most 5 marks. The quadratic solution is presented below.

$i, j, maxi, maxj : INTEGER;$
$sum, maxsum : FLOAT;$
$maxsum := T(1); maxi := 1; maxj := 1;$
for i **in** $1..N$ **loop**
    $sum := 0.0;$
    for j **in** $i..N$ **loop**
        $sum := sum + T(j);$
        if $sum > maxsum$
            then $maxsum := sum; maxi := i; maxj := j;$
        end if;
    end loop;
end loop;

Where the heaviest segment is $T(maxi, maxj)$ and its weight is $maxsum$.

5.B Let index i (external loop) stands for the left end of the slice in array T and index j (internal loop) for its right end. Then each execution of the body of an internal loop corresponds to a visit at slice $T(i..j)$, for all $1 \leq i \leq j \leq N$. The sum for the slice of the form $T(i, i)$ (i=1..N) is computed during the first iteration of the internal loop, when $i = j$. The sum for any other segment $T(i..j)$ (where $i < j$) is computed as the sum of elements in $T(i..j-1)$ (found during last iteration of an internal loop) and value $T(j)$ (added at the current iteration). The value $maxsum$ is initially set to the value $T(1)$ but it is updated whenever current value of sum (sum of elements in current $T(i..j)$) is becoming larger the $maxsum$.

5.C Time complexity of the solution is: $O(N^2)$ since we have only two nested loops with ranges not greater than N; and it is $\Omega(N^2)$ since for $i = 1..N/2$ number of iterations of internal loop is at least $N/2$. Then the time complexity is $\Theta(N^2)$.

Answers to question 6

6.A

$i, j, k : INTEGER;$
$i := 1; j := 1;$
while $i \leq N$ **and** $j \leq N$ **loop**
    if $T_1(i) \leq T_2(j)$
        then $T_3(i + j - 1) := T_1(i); i := i + 1;$
        else $T_3(i + j - 1) := T_2(j); j := j + 1;$
    end if;
end loop;
if $i > N$
    then for k **in** $j..N$ **loop** $T_3(N + k) := T_2(k)$; **end loop**;
    else for k **in** $i..N$ **loop** $T_3(N + k) := T_1(k)$; **end loop**;
end if;

Time complexity of the algorithm is linear in N since the number of elements that have been copied to table $T_3(1..2N)$ is exactly $2N$.

6.B In case one of the tables is not sorted the complexity of the problem becomes $\Theta(N \log(N))$. We have to sort elements in unsorted table (otherwise elements in T_3 would be unsorted) which requires time $\Omega(N \log(N))$. And any sorting algorithm gives solution $O(N \log(N))$.

6.C (i) in the comparison model *SelectionSort* is always quadratic and *MergeSort* has always complexity $O(n \log(n))$; (ii) in the model with swaps *SelectionSort* is faster since it always swaps at most n times and *MergeSort* always require $O(n \log n)$ swaps

6.D This is *SelectionSort*, it finds the smallest element and swaps it with the first one, then second smallest and swaps it with the second one, etc

6.E When we are dealing with (not too big) integers *CountSort* can sort them in linear time. *CountSort* is not based on comparisons between sorted numbers so the $\Omega(n \log(n))$ lower bound does not hold in this case.