EXERCISES UNIT 1

1. Using the *clock_gettime* function,
a) Should we use any special directives compiling or linking the program?
b) What clock identifier should we use if we want to measure real time instead of CPU time?
c) Compute the time spent in the following loop using the *clock_gettime* function.

```
for(i = 1; i <= 1000; i++)
    f(i);
```

2. When using gprof with a program compiled with the option –O3, there are functions of the program that are not shown in the tables. Why is this happening? Don't they have any cost?

3. Implement in C++ a countsort algorithm for sorting a vector of n registers of type *Persona*. *Persona* contains a key attribute (clave) with natural numbers between 1 and 10.

```
struct Persona
{
      clave: Nat
      nombre: string
}
```

```
countsort(v:Vector[1..n] de Persona) dev v2:Vector[1..n] de Persona
```

4. Sort the following list of numbers using the LSD Radixsort. Show all auxiliary lists (*bucket* vector) and the result list after each iteration.

403 16 239 821 9 342 910 524 373 145

5. Modify the Radixsort algorithm to sort strings of characters of a fixed size of 20 characters. The *digit* function must be totally defined. **Analyse** the cost of the algorithm.

6. Implement a bucketsort algorithm for sorting a vector of *n* float numbers. The float numbers are in the range]-1000, 1000[and follow a uniform distribution. Which will be the asymptotic cost of the algorithm?

7. Calculate the amortized cost of the push_back operation with an extensible vector of initial size N which expands its size by N elements each time it exhausts its capacity. Use precise expressions.

8. We have an m-bit binary counter that we use for counting from 0 to $n=2^m-1$. Each increment of the counter is done with a call to the ***inc*** function. **a)** Calculate the worst-case for counting from 0 to n using the worst-case cost of function *inc*. **b)** Calculate an upper bound for the cost of counting from 0 to n using the amortized cost of function *inc*. All costs will be calculated in number of iterations.

```
func inc(C[1..m] de {0,1})
{Pre: m >= 1}
      j ← m + 1
      repetir
            j ← j – 1
            C[j] ← 1 – C[j]
      hasta (C[j] = 1) o (j = 1)
```