Technical University of Denmark

Written examination, December 12, 2017	
Course name:	Mathematical Software Programming
Course number:	02635
Aids allowed:	All aids allowed
Exam duration:	4 hours
Weighting:	80/100

Final exam
Mathematical Software Programming

This exam contains a total of 18 questions: 14 multiple choice questions (questions 1–14) and 4 programming questions (questions 15–18). Your exam answers must be submitted electronically as a **PDF document**. You may include your code in the document along with your answers or submit the code separately in a ZIP file.

1. (4 points) Properties of floating-point arithmetic.

   (a) The associative property of multiplication holds for floating-point arithmetic, *i.e.*,

   $$a(bc) = (ab)c$$

   where a, b, and c are floating-point numbers.

   A. True
   B. False

   (b) The distributive property of multiplication holds for floating-point arithmetic, *i.e.*,

   $$a(b + c) = ab + ac$$

   where a, b, and c are floating-point numbers.

   A. True
   B. False

2. (2 points) What is the condition number of $f(x) = \cos(x)$?

   A. $|x \tan(x)|$
   B. $|\tan(x)|$
   C. $|\sin(x)|$
   D. $|x \sin(x)|$

   $$\left| \frac{x f'(x)}{f(x)} \right|$$

3. (2 points) Given n numbers $x_1, x_2, \ldots, x_n$ where $\sum_{i=1}^{n} |x_i| \approx 500$ and $|\sum_{i=1}^{n} x_i| \approx 0.01$, what is approximately the worst-case relative error for sequential floating-point summation of the numbers if the unit round-off is u and $nu < 0.1$?

   A. $5.3nu$
   B. $53nu$
   C. $530nu$
   D. $5300nu$
   E. $53000nu$

   $$\frac{|\hat{S}_n - S_n|}{|S_n|} \leq c \cdot 1.06nu$$

   $$c = \frac{\sum_{i=1}^{n} |x_i|}{|\sum_{1} x_i|}$$

4. (4 points) A programmer wrote the following C function for evaluating quadratic functions of the form

$$f(x) = x^T Ax$$

where x is a vector of length n and A is a symmetric matrix of order n:

```
double quad_form(double **A, double *x, unsigned int n) {
    unsigned int i,j;
    double res = 0.0;
    double *y = malloc(n*sizeof(*y));
    if (y==NULL) return NAN;
    for (i=0;i<n;i++) {
        y[i] = 0.0;
        for (j=0;j<n;j++)
            y[i] += A[i][j]*x[j];
        res += y[i]*x[i];
    }
    return res;
}
```

There is a serious problem with this implementation. What is it?

A. The function returns `res` which is a local variable.

B. The function does not give the correct answer.

C. The function leaks memory.

D. The array `y` is too small, leading to undefined behavior.

E. The array `y` is not properly initialized.

5. (2 points) Improved locality generally leads to

A. improved numerical accuracy

B. worse numerical accuracy

C. fewer cache hits

D. fewer cache misses

6. (4 points) The theoretical improvement in speed of execution of a task executed on p processors can be expressed as

$$S(p) = \frac{T(1)}{T(p)} = \frac{\cancel{fT(1)} + (1 - \cancel{f)T(1)}}{(f/p)\cancel{T(1)} + (1 - f)\cancel{T(1)}}$$

where $T(p)$ is the execution time on p processors (real time) and f is the so-called parallel fraction of the task. For example, if 50% of a task can be parallelized, then $f = 0.5$. Suppose we measure $T(1) = 100$ and $T(4) = 37$.

(a) What is the value of the parallel fraction f of the task?

A. $f = 0.37$

B. $f = 0.71$

C. $f = 0.84$

D. $f = 0.91$

$$\frac{37}{100} = \frac{f}{p} + 1 - f$$

(b) What is the theoretical execution time on 8 processors?

A. $T(8) = 20.4$

B. $T(8) = 26.5$

C. $T(8) = 29.0$

D. $T(8) = 37.9$

7. (4 points) The real-valued function

$$f(x) = \frac{ax + b}{cx + d}$$

is a so-called linear fractional function. We will assume that $ad - bc \neq 0$ and $c \neq 0$.

(a) What is the condition number of f?

A. $\left|\frac{x(ad-bc)}{ac(x+b/a)(x+d/c)}\right|$

B. $\left|\frac{(ad-bc)}{ac(x+b/a)(x+d/c)}\right|$

C. $\left|\frac{x}{(x+b/a)(x+d/c)}\right|$

D. $\left|\frac{(ad-bc)}{ac(x+d/c)}\right|$

$$\left|\frac{x * f'(x)}{f(x)}\right|$$

(b) Catastrophic cancellation is likely to occur when

A. $ax + b \approx cx + d$

B. $ax + b \approx bx + a$

C. $x \approx a/b$ or $x \approx c/d$

D. $x \approx -b/a$ or $x \approx -d/c$

8. (4 points) A half-precision floating point number occupies 16 bits and has the following representation

s	$e_1 \dots e_5$	$d_1 d_2 \dots d_{10}$

where s is the sign bit, d_i is the ith bit of the mantissa, and e_i is the ith bit of the exponent. Thus, a half-precision floating point number can be represented as

$$x = (-1)^s \cdot (d_0.d_1 d_2 \dots, d_{10})_2 \cdot 2^E = (-1)^s \cdot \sum_{i=0}^{10} d_i 2^{E-i}$$

where $E \in \{-14, -13, \dots, 14, 15\}$ is a decimal representation of the exponent.

(a) What is the unit round-off for the half-precision floating point format?

A. $u = 2^{-10}$
B. $u = 2^{-11}$
C. $u = 2^{-12}$
D. $u = 2^{-13}$

(b) Recall the following relative error bound for the sequential summation of n floating-point numbers $x_1, \dots, x_n$

$$\epsilon_{\text{rel}} \leq \frac{\sum_{i=1}^n |x_i|}{|\sum_{i=1}^n x_i|} \cdot 1.06 \cdot nu, \quad nu < 0.1.$$

Now suppose $x_1, \dots, x_n$ are nonnegative numbers. What is approximately the largest value of n for which we can guarantee a relative error of at most 10% using half-precision arithmetic?

A. $n \approx 96$
B. $n \approx 102$
C. $n \approx 193$
D. $n \approx 205$

$1 \cdot 1.06 \cdot nu \leq 0.1$

$n \leq \frac{0.1}{1.06} \cdot 2^{11}$

2048

9. (6 points) Given a vector $x = (x_1, \ldots, x_n)$ of length n, we can compute the sum $s_n = \sum_{i=1}^{n} x_i$ recursively using the following C function:

```
double rsum(double *x, unsigned int n) {
    unsigned int m = n/2;
    if (m>1)
        return rsum(x,m)+rsum(x+m,n-m);
    else if (m==1 && n-m>1)
        return x[0]+rsum(x+m,n-m);
    else if (m==1 && n-m==1)
        return x[0]+x[1];
    else
        return x[0];
}
```

(a) What type of recursion is this?

A. Single recursion

B. Multiple recursion

(b) What is the time complexity of computing s_n using `rsum(x,n)`?

A. $O(1)$

B. $O(\log(n))$

C. $O(n)$

D. $O(2^n)$

(c) Excluding the space required to store x, what is the space complexity of computing s_n using `rsum(x,n)`?

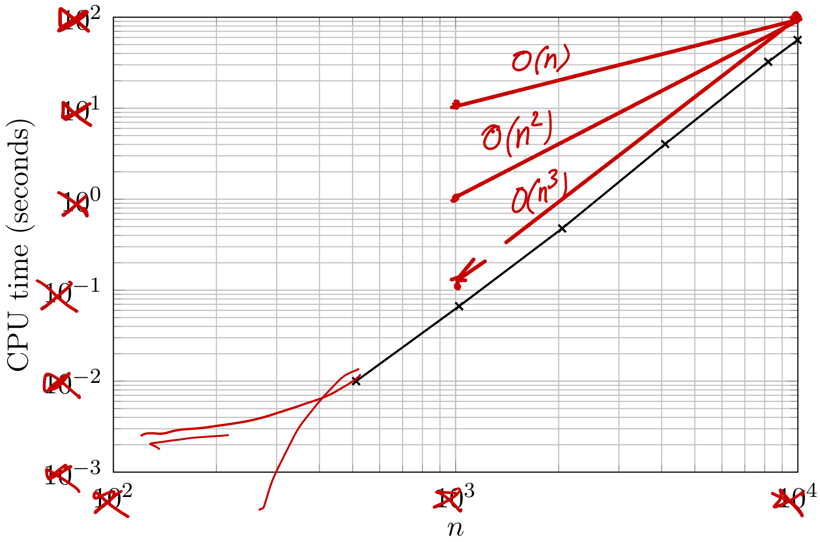
A. $O(1)$

B. $O(\log(n))$

C. $O(n)$

D. $O(2^n)$

10. (4 points) The following plot shows the CPU time required by some algorithm to solve a certain problem as a function of its dimension n.



What is the time complexity of the algorithm?

A. $O(1)$

B. $O(n)$

C. $O(n^2)$

D. $O(n^3)$

11. (2 points) The purpose of loop unrolling is to

A. reduce execution time

B. improve numerical accuracy

C. improve code readability

D. avoid while loops

12. (2 points) In the C++ programming language, the datatype `int&` is

A. the address of an `int`

B. a pointer to an `int`

C. a reference to an `int`

D. the size of an `int` in bytes

13. (2 points) The `std::vector` class template is a container that represents a

    A. linked list

    B. stack

    C. static array

    D. dynamic array

14. (4 points) A programmer wrote the following function to check if two vectors x and y of length n are orthogonal within numerical precision:

```c
int orthogonal(unsigned int n, double *x, double *y) {
  double dot = 0.0;
  for (unsigned int i=0; i<n; i++)
    dot += x[i]*y[i];
  if (fabs(dot) <= 1.06*n*DBL_EPSILON)
    return 1;    // x and y are orthogonal
  else
    return 0;    // x and y are not orthogonal
}
```

    Assuming that all floating-point numbers are normal and that there is no under- or overflow, will this implementation work as intended?

    A. Yes

    B. No

15. (8 points) Write a C function that evaluates the function

$$f(n) = \log(n!)$$

    with domain $\mathbb{N}_0$ (the set of nonnegative integers).

    Hint: We have that $0! = 1$.

    Use the following function prototype:

```c
double logfactorial(unsigned int n);
```

    Test your code with $n \in \{0, 1, 2, 5, 50, 100, 1000\}$.

16. (8 points) Given n linear functions of $x \in \mathbb{R}$

$$g_i(x) = a_i x + b_i, \quad i = 1, \dots, n,$$

we seek to compute the maximum of the n functions, *i.e.*,

$$f(x) = \max_{i=1,\dots,n} (a_i x + b_i).$$

Implement a function that evaluates $f(x)$ given x and two length n arrays with the coefficients $a_1, \dots, a_n$ and $b_1, \dots, b_n$.

Use the following prototype:

```
double linear_max(
    const double x,
    const double *a, /* array with a coefficients */
    const double *b, /* array with b coefficients */
    unsigned int n
);
```

17. (8 points) Given a nonzero vector $v \in \mathbb{R}^n$, the matrix

$$T = I - 2\frac{vv^T}{\|v\|_2^2}$$

is a so-called Householder matrix.

(a) Write a function that takes a vector x and a vector v as inputs (both of length n) and computes $x := Tx$ (i.e., the function should overwrite x with the result Tx). The function should have the following prototype:

```
int householder(const double *v, double *x, unsigned int n);
```

The return value should be zero if successful. In case of an error, the function should return a nonozero value and the array `x` should remain unchanged.

(b) What is the time-complexity of computing the matrix-vector product Tx?

18. (10 points) A real-valued univariate polynomial of degree n

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

can be represented using the following data structure

```
struct polynomial {
    unsigned int n;
    double *coef;
};
```

where `coef` points to the first element of an array that stores the $n+1$ coefficients $(a_0, a_1, \ldots, a_n)$. You may assume that $a_n \neq 0$.

(a) The value of the polynomial p at x can be evaluated using Horner's method: initialize $b_n := a_n$ and compute

$$\begin{aligned} b_{n-1} &:= a_{n-1} + b_n x \\ b_{n-2} &:= a_{n-2} + b_{n-1} x \\ &\vdots \\ b_0 &:= a_0 + b_1 x \end{aligned}$$

which yields $b_0 = p(x)$. Write a function that evaluates $p(x)$ using Horner's method.

Your function should have the following prototype:

```
double poly_eval(struct polynomial *p, double x);
```

(b) The product

$$q(x) = p_1(x) p_2(x)$$

of two polynomials $p_1(x)$ and $p_2(x)$ of degree n_1 and n_2, respectively, is itself a polynomial $q(x)$, and the degree of $q(x)$ is $n_1 + n_2$. Write a function that takes two polynomials p_1 and p_2 as inputs and computes q.

Your function should have the following prototype:

```
struct polynomial * poly_mul(
    const struct polynomial *p1,
    const struct polynomial *p2
);
```