Your name: _____

Your student ID: _____

Instructions:

- This is an in-class quiz.
- Make sure to write your name and student ID above.
- Bring some form of ID card to the quiz. A student card is accepted. You will be required to show your id as you hand in the quiz.
- Answer the questions with pen or pencil on these sheets in class.
- Write answers to the questions exactly on the locations provided on the quiz sheets.
- You may use a calculator, but you cannot use a laptop, cellular device, or tablet. No other written material is allowed.
- Your answers should be short and without explanation unless one is explicitly requested.
- The duration of the quiz is 50 minutes + 10 minutes reading time.
- You are not allowed to communicate with anyone during the quiz and you are not allowed to look at the quiz paper of anyone else.. The only communication allowed is asking questions to the instructor at the end of the first 10 minutes as instructed. Any questions asked at that time will be answered to the whole group.
- In case of exceptional circumstances contact the course coordinator for special arrangements.

Note: all 10 questions (1a, 1b, .., 3c) receive an equal amount of points.

Quiz questions and on next page...

Question 1 Consider the following function implemented in Julia,

```
1  function myfun!(bv::Vector{Bool})
2     i, j = 1, length(bv)
3
4     while true
5        while bv[i]==false
6           i += 1
7        end
8        while bv[j]==true
9           j -= 1
10       end
11       i >= j && break
12       bv[i], bv[j] = bv[j], bv[i]
13    end
14    return bv
15 end
```

(a) What is the output of `myfun!([false, true, false, true, false, false])`?

(b) What is the worst-case order of numerical complexity of this algorithm? Formulate a short answer using Big-O notation where n is the length of the Vector `bv`.

(c) Formulate a modified version of line 14 of this function so it returns the numbers of `false`s and `true`s in bv as a tuple of two integer values.

(d) In its current version, the function myfun! modifies the argument with which the function is called. Which statement(s) could you add in line 3 to avoid this?

Question 2 Consider the following function

```
1  function myfun2(x::T) where {T <: Integer}
2     count = 0
3     for i=1:sizeof(T)*8
4        count += x & 1        # & is the bit-wise AND-operator, e.g. 12 & 10 evaluates to 8
5                              # (i.e. 0b1100 & 0b1010 evaluates to 0b1000)
6        x >>= 1               # this is equivalent to: x=x >> 1
7     end
8     return count
9  end
```

(a) What is the output of `myfun2(21)`?

(b) Modify line 3 such that the number of times the loop is executed becomes minimal without changing the output of the function.

(c) An implementation of this function in one line could be given by

```
myfun3(x::T) where {T <: Integer} = sum(x >> i & 1 for XXXX)
```

Determine the statement `XXXX` such that `myfun3` produces the same output as `myfun2`.

Question 3 The function `adaptive_quadrature` defined below approximates the integral of the function `fun` on the interval `(a,b)`.

```
1  quad(f, x1, x2)= (f(x1) + f(x2)) * (x2 - x1) / 2
2
3  function recurse(f, x1, x2, whole, tol)
4     mid = (x1 + x2) / 2
5     left = quad(f, x1, mid, tol)
6     right = quad(f, x2, mid, tol)
7     if abs(whole-(left + right)) / abs(whole) > tol
8        return recurse(f, x1, mid, left, tol) + recurse(f, mid, x2, right, tol)
9     else
10       return left + right
11    end
12 end
13
14 adaptive_quadrature(fun, a, b, tol=0.01)=recurse(fun, a, b, quad(fun, a, b), tol)
```

(a) Write a modified version of line 7, so the resulting function implements the trapezoid rule/Riemann sum with a grid size of `0.2` or smaller.

(b) The first time you run this code using `adaptive_quadrature(sin, 0, pi)`, it throws the error

```
ERROR: MethodError: no method matching quad(::var"#11#12", ::Int64, ::Float64, ::Float64)
The function ‘quad‘ exists, but no method is defined for this combination of argument types.

Closest candidates are:
quad(::Any, ::Any, ::Any)
@ Main Untitled-2:1

Stacktrace:
[1] recurse(f::Function, x1::Int64, x2::Irrational{:pi}, whole::Float64, tol::Float64)
@ Main ./Untitled-2:5
[2] adaptive_quadrature(fun::Function, a::Int64, b::Irrational{:pi}, tol::Float64)
@ Main ./Untitled-2:14
[3] top-level scope
@ REPL[1]:1
```

Fix the issue, i.e. write the line(s) which you need to change, and the content of the modified line(s).

(c) Once this is fixed, you run the same command again. The code runs but doesn't terminate. First, find the problem (write the line number which has the problem), then suggest a modified version of this line to fix the issue.

Do not forget to write your name and student number on the front page.