Mid-Term Exam

The Art of Computer Programming

6 November 2026

You have 1.5 hours to answer this mid-term exam. You are allowed 10 A4 sheets (i.e., 20 pages) with the content of your choice. No electronic devices or any other material is allowed. Do not forget to add your name on all your answer sheets and to number them. The exam is graded out of 20 points.

The exam consists of 6 exercises, with a total of 16 questions. Exercises are independent and can be done in any order. When you write code, minor syntax errors are not penalized: what matters is that your reasoning is correct and your code is clear. "Justify briefly" means giving the main argument in one or two sentences (for a complexity, e.g., how many times each loop runs): no formal proof is expected.

Exercise A: A Recursive Function (3 points)

Consider the following Python function, where `n` is a nonnegative integer. As a reminder, `//` and `%` compute, respectively, the quotient and the remainder of the integer division.

```
def f(n):
    if n == 0:
        return 0
    return n % 2 + f(n // 2)
```

Q1. (1 point) List the successive calls to `f` made when computing `f(13)`, and give the value it returns.

Q2. (1 point) Write 13 in binary. In general, what does `f(n)` compute, in terms of the binary representation of `n`? Justify briefly.

Q3. (1 point) Give, with the $\Theta(\cdot)$ notation, the time complexity of `f(n)` as a function of n. Justify briefly.

Exercise B: Memory in C (3.5 points)

Q4. (2 points) What are the values of the three elements of the array `a` at the end of the following C program? (No explanation needed.)

```c
void g(int x, int *y) {
  x = 10;
  *y = 20;
}
```

```c
int main(void) {
  int a[3] = {1, 2, 3};
  int *p = a + 1;
  g(a[0], p);
  p++;
  *p = 30;
}
```

Q5. (1.5 points) The following C program compiles, but it has errors in its use of memory. Identify two of them: for each, say where it is, what goes wrong when the program runs, and how to correct it.

```c
#include <stdio.h>
#include <stdlib.h>

int *triples(int n) {
  int *r = malloc(n * sizeof(int));
  for (int i = 0; i <= n; i++)
    r[i] = 3 * i;
  return r;
}
```

```c
int main(void) {
  for (int k = 1; k <= 1000; k++) {
    int *s = triples(k);
    printf("%d\n", s[k - 1]);
  }
}
```

Exercise C: Complexity (3.5 points)

Q6. (1.5 points) Give, with the $\Theta(\cdot)$ notation, the time complexity of each of these two Python functions as a function of n. Justify briefly.

```python
def h1(n):
    c = 0
    for i in range(n):
        for j in range(i, n):
            c += 1
    return c
```

```python
def h2(n):
    c = 0
    i = 1
    while i <= n:
        for j in range(i):
            c += 1
        i = 2 * i
    return c
```

Q7. (1 point) The following Python function returns the elements of a list `t` of n integers without their duplicates, keeping the order of their first occurrences.

```python
def without_duplicates(t):
    seen = []
    for x in t:
        if x not in seen:
            seen.append(x)
    return seen
```

What is its worst-case time complexity, and on which kind of input is this worst case reached? Explain how to modify it to get $\Theta(n)$ on average, with the same result.

Q8. (1 point) A dynamic array has initially a capacity of 1. Whenever it is full and a new element is inserted at its end, a new block of twice the capacity is allocated and all its elements are copied there. We insert 20 elements in an initially empty array. In total, how many times is an element copied from an old block to a new one?

Exercise D: Algorithm Design (4 points)

A *run* of a sequence is a block of consecutive equal elements of that sequence. We are interested in the length of the longest run of a sequence of integers. For instance:

- On input $(5, 5, 2, 2, 2, 7, 2, 2)$, the answer is 3.
- On input $(1, 1, 2, 2, 2)$, the answer is 3.
- On input $(1, 2, 3)$, the answer is 1.
- On input $()$, the answer is 0.

Q9. (3 points) Write a function that takes as input a sequence of integers and returns the length of its longest run, either in clear pseudocode or in Python, C, or C++. It should make a single pass over the sequence.

Q10. (1 point) Give the worst-case time complexity and the additional space complexity of your function, in terms of the length n of the sequence. Justify briefly.

Exercise E: A Linked List in C (3 points)

A singly linked list of integers is represented by a pointer `first` to its first node, which is `nullptr` for an empty list. The following function inserts a value at the front of a list:

```c
typedef struct Node {
  int value;
  struct Node *next;
} Node;
```

```c
void list_push_front(Node **first, int value) {
  Node *n = malloc(sizeof(Node));
  if (n == nullptr) exit(1);
  n->value = value;
  n->next = *first;
  *first = n;
}
```

Q11. (1 point) Draw the list obtained after executing the following code, and give the value of `first->next->value`.

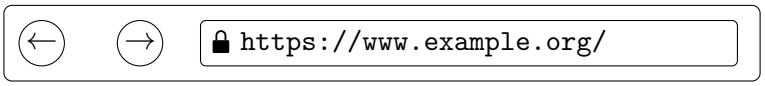
```c
Node *first = nullptr;
list_push_front(&first, 5);
list_push_front(&first, 8);
list_push_front(&first, 13);
```

Q12. (1 point) Write a C function `int list_count(const Node *first, int x)` that returns the number of times the value `x` appears in a list.

Q13. (1 point) We want to use a singly linked list as a queue, keeping a pointer to its first node and a pointer to its last node. At which end of the list should elements be added, and at which end should they be removed, so that both operations take $\Theta(1)$? Why not the other way round?

Exercise F: A Class for Browsing History (3 points)

A Web browser has a back button and a forward button:



back forward

- back returns to the page that was displayed before the current one; it can be pressed several times to go further back.
- forward cancels the last press of back, returning to the page it left.
- Visiting a new page, e.g., by following a link, makes forward impossible until back is pressed again.

The following C++ class models this behavior: `current` is the page being displayed, `past` is the stack of pages that back can return to, and `future` is the stack of pages that forward can return to. For a `std::vector v`, `v.back()` is its last element, `v.pop_back()` removes it, and `v.clear()` removes all elements.

```
#include <string>
#include <vector>

class History {
private:
  std::string current;
  std::vector<std::string> past;
  std::vector<std::string> future;

public:
  History(std::string start)
      : current(start) {}

  void visit(std::string page) {
    past.push_back(current);
    current = page;
    future.clear();
  }

  void back() {
    if (!past.empty()) {
      future.push_back(current);
      current = past.back();
      past.pop_back();
    }
  }
};
```

Q14. (1 point) Give the final contents of the properties `current`, `past`, and `future` of `h` after the following code is executed. (No explanation needed.)

```
History h("home");
h.visit("a"); h.visit("b"); h.back(); h.visit("c"); h.back();
```

Q15. (1 point) Write a method `forward` that undoes the effect of the last `back`, and does nothing if there is no page to go forward to.

Q16. (1 point) To limit memory use, we now keep at most k pages in `past`, where k is a property of the class. Whenever `past` already has k elements, `visit` first removes the oldest one, by calling `past.erase(past.begin())`. What is the worst-case complexity of `visit` now, as a function of k (the strings being considered of bounded length)? Which other collection could be used for `past` in order to make all the operations on it $\Theta(1)$, and why?