Weekplan: Warmup

Philip Bille Inge Li Gørtz

Reading

Survival Guide and Programming Prerequisites. Do the test in the Programming Prerequisites.

Exercises

1 Loops The goal of this exercise is to check if you understand some basic programming concepts. Do *not* implement the functions. What do the functions `loop1`, `loop2`, `loop3` and `loop4` in Figure 1 (Java/C/C++) or Figure 2 (Python) return when

1.1 $\texttt{n} = 4$?

1.2 $\texttt{n} = 10$?

1.3 $\texttt{n} = 1000$?

1.4 as a function of `n`?

2 Recursion and Iteration A function is *recursive* if it calls itself. For instance the function `f(A,n)` in Figure 1/ 2 is recursive. Solve the following exercises.

2.1 What does `f(A,n)` compute if `A` is an array of integers of length `n`? You should *not* implement it.

2.2 Rewrite `f(A,n)` to be *iterative*, ie. make a function that computes the same as `f(A,n)` but without calling itself.

3 Introduction to CodeJudge Solve the following exercises.

3.1 Go to CodeJudge and read the tutorial. See how to access CodeJudge on the course homepage.

3.2 [*w*†] Implement a function to add two numbers and test it in CodeJudge.

4 [†] **Christmas Presents** The three siblings Anna, Laura, and Oscar, just got their Christmas presents. Anna gets jealous if both Laura and Oscar has more presents than her. Laura gets jealous if Anna has more presents than her, and Oscar gets jealous if Anna or Laura (or both) has more presents than him.

Give an algorithm, that given the number of presents each of the three siblings has, returns the names of the siblings that gets jealous. Implement your algorithm and test it on CodeJudge.

5 Linearthritis You have recently hired 128 programmers for your new high-tech startup company. Unfortunately, one of them is suffering from the feared Linearthritis disease that makes everybody near the person write slow programs. In order to identify the diseased programmer you have rented a special room that you can use to determine if the diseased programmer is with in a group of your programmers. It is extremely expensive to rent this room and the process needed to test a group is complicated (long and exhausting programming tests are necessary). Therefore you would like to minimize the number of times you have to use the room in order to find the diseased. Solve the following exercises.

5.1 Show you can find the diseased programmer using at most 7 tests.

5.2 How many tests do you need if you have n programmers instead of 128?

5.3 [$*$] Assume that you rent $k > 1$ rooms you can use to test k groups of programmers simultaneously. How many *rounds* of tests are enough to identify the diseased programmer? In each round you can test k groups in parallel.

6 Zombie Duels You have an army of n brainless zombies. You want to find the strongest and the weakest zombie in the army. By pairing up two zombies in a cage with a big chunk of brain matter you can quickly determine which of the two are the strongest. Unfortunately, zombies wear out in this process, so you want to minimize the number of duels needed. Solve the following exercises.

6.1 Explain how to find the strongest zombie using at most $n-1$ duels.

6.2 [$*$] Explain how to find both the strongest and the weakest zombie using at most $3n/2$ duels.

6.3 [$**$] Explain how to find both the strongest and second strongest zombie using at most $n+\log_2 n$ duels.

7 [$**$] **Ants on a Stick** Suppose that you have 100 ants on a stick of length 100 cm. At the start, each ant is placed at some position on the stick, pointing either toward the left or right end of the stick. Then, all ants begin to move simultaneously. The ants all move at a speed of 1 cm per second. If an ant bumps into another ant they both immediately reverse directions and continue at the same speed, and if an ant reaches the end of the stick it falls off the stick. What is maximum duration of time before all ants have fallen off the stick over all possible initial placements of the ants?

```
int loop1(int n) {
   int x = 0;
   for(int i = 0; i < n; i++) {
      for(int j = 0; j < n; j++) x++;
   }
   return x;
}

int loop2(int n) {
   int x = 0;
   for(int i = 0; i < n; i++) x++;
   for(int j = 0; j < n; j++) x++;
   return x;
}

int loop3(int n) {
   int x = 0;
   for(int i = 0; i < n; i++) {
      if (i == n-1) for(int j = 0; j < n; j++) x++;
   }
   return x;
}

int loop4(int n) {
   int x = 0;
   for(int i = 0; i < n; i++) {
      for(int j = i; j < n; j++) x++;
   }
   return x;
}

int f(int[] A, int n) {
   if(n == 0) return 0;
   else return f(A, n - 1) + A[n-1];
}
```

Figure 1: Loops and recursion in Java/C/C++. To obtain a valid C/C++ program for the function `f`, `[]` should be changed to `*`

```
def loop1(n):
   x = 0
   for i in range(n):
       for j in range(n):
           x += 1
   return x

def loop2(n):
   x = 0
   for i in range(n):
      x += 1
   for j in range(n):
      x += 1
   return x

def loop3(n):
   x = 0
   for i in range(n):
      if (i == n-1):
         for j in range(n):
            x += 1
   return x

def loop4(n):
   x = 0
   for i in range(n):
      for j in range(i,n):
         x += 1
   return x

def f(A, n):
   if (n == 0):
      return 0
   else:
      return f(A, n - 1) + A[n-1]
```

Figure 2: Loops and recursion in Python.