SCHOOL OF COMPUTING

ASSESSMENT FOR
Special Term I, 2015/2106

CS1010X — PROGRAMMING METHODOLOGY

June 2016 Time Allowed: 2 Hours

INSTRUCTIONS TO STUDENTS

1. Please write your Student Number only. Do not write your name.
2. The assessment paper contains **FIVE (5) questions** and comprises **TWENTY-TWO (22) pages**.
3. Weightage of questions is given in square brackets. The maximum attainable score is 100.
4. This is a **CLOSED** book assessment, but you are allowed to bring **TWO** double-sided A4 sheets of notes for this exam.
5. Write all your answers in the space provided in this booklet.
6. **Please write your student number below.**

STUDENT NO: ______________________________

(this portion is for the examiner's use only)

Question	Marks	Remark
Q1		
Q2		
Q3		
Q4		
Q5		
Total		

Question 1: Parallel Computations [25 marks]

Ben Bitdiddle was sitting around one day and he decided that it would be a great idea to be able to create some functions that will be flexible enough to perform a computation on an arbitrary number of arguments. In particular, he came up with 3 functions `triple`, `add_one` and `filter_odd` that operates on an arbitrary number of arguments and returns the answers as a tuple, as shown:

```
>>> triple(1,2) # multiplies arguments by 3
(3,6)

>>> triple(1,2,3)
(3,6,9)

>>> add_one(2,3) # adds one to the arguments
(3,4)

>>> add_one(1,2,3)
(2,3,4)

>>> filter_odd(1,2,3)  # keep only the odd numbers
(1,3)

>>> filter_odd(2,4,6)
()
```

A. Give a possible implementation for the function `triple`. [3 marks]

B. Give a possible implementation for the function `add_one`. [3 marks]

C. Give a possible implementation for the function `filter_odd`. [4 marks]

Ben then decided that it would good to be able to string all the operations together and tried:

```
>>> filter_odd(add_one(triple(1,2,3)))
```

Unfortunately, this resulted in a `TypeError`. So, Ben decided to come up with a new function that he called `composen` to solve the problem. In particular, with `composen`, Ben managed to do:

```
>>> alpha = composen(triple,add_one)
>>> alpha(1,2,3)
(6, 9, 12)

>>> beta = composen(add_one,triple)
>>> beta(1,2,3)
(4, 7, 10)

>>> composen(alpha,beta)
>>> gamma(1,2,3)
(15, 24, 33)
```

D. Give a possible implementation for the function `composen`. [5 marks]

E. Now suppose:

```
>>> delta = composen(alpha,composen(filter_odd,beta))
```

What is the value of `delta(1,2,3,4,5)`? Explain. [5 marks]

Ben has a friend, Alyssa P Hacker, who heard about Ben's problem with the `TypeError`. Alyssa told Ben, "Actually, we can use just Python's Exception handling mechanism to deal with the `TypeError` using a function called `wrapper`. In particular, we can then do the following:"

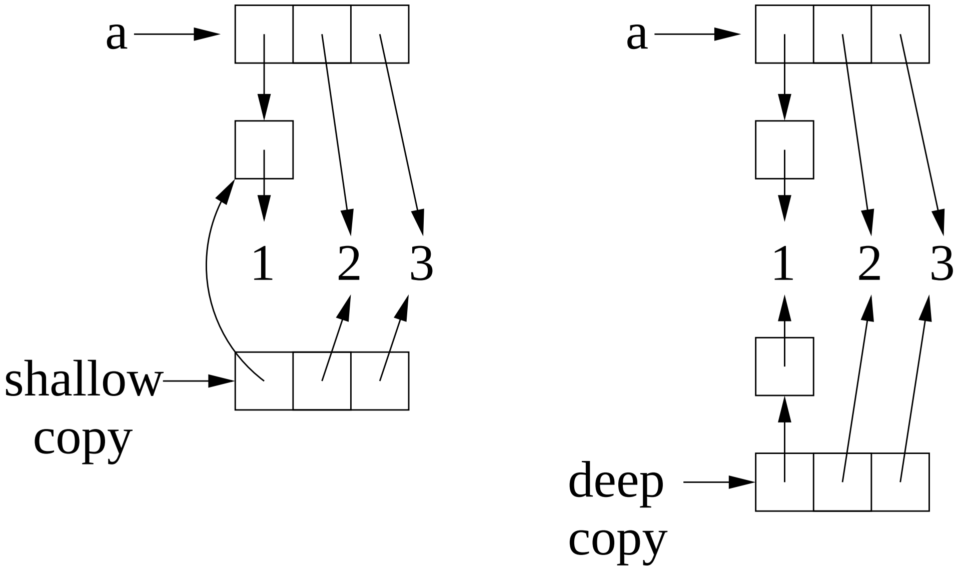
```
>>> new_triple = wrapper(triple)
>>> new_add_one = wrapper(add_one)
>>> new_add_one(new_triple(1,2,3))
(4, 7, 10)
```

F. Give a possible implementation for the function `wrapper`. [5 marks]

```
def wrapper(f):
```

Question 2: How Deep Is Your Copy? [17 marks]

We discussed shallow and deep copy in class. In particular, a shallow copy only copies the first layer, while a deep copy will copy every list or tuple element. This is illustrated in the following figure:



A. [**Warm Up**] Give a possible implementation for `deep_copy` for a list. [3 marks]

```
def deep_copy(lst):
```

B. One question after you do a deep copy is whether you actually did the copy correctly. So, what we want is to have a function `deep_copy_check` that checks whether one list is a deep copy of the other. For example:

```
>>> a = [[[1],2],3,4]
>>> b = deep_copy(a)
>>> deep_copy_check(a,b) # assuming deep_copy is correct
True

>>> b[0][0] = a[0][0]
>>> deep_copy_check(a,b)
False

>>> b[0] = a[0]
>>> deep_copy_check(a,b)
False

>>> c = deep_copy(b)
>>> deep_copy_check(a,c)
True

>>> d = c.copy()
>>> deep_copy_check(c,d)
False
```

Give a possible implementation for deep_copy_check. [6 marks]

```
def deep_copy_check(lst1,lst2):
```

C. [**Replacements**] Next, what we want to do is to replace elements within a tree consisting of trees with a function called `deep_replace`, which takes in a list and two arguments, a reference and a target and replaces all instances of reference with the target in the tree, i.e.

```
>>> a = [[[1],2],3,4]
>>> deep_replace(a,2,3)
>>> a
[[[1], 3], 3, 4]

>>> deep_replace(a,3,5)
>>> a
[[[1], 5], 5, 4]

>>> deep_replace(a,5,1)
>>> a
[[[1], 1], 1, 4]

>>> deep_replace(a,1,9)
>>> a
[[[9], 9], 9, 4]
```

Give a possible implementation for `deep_replace`. [4 marks]

```
def deep_replace(lst,a,b):
```

D. [**Counting Replacements**] Next, we don't just want to replace elements, we actually want to count the number of elements that got replaced in the process, i.e.

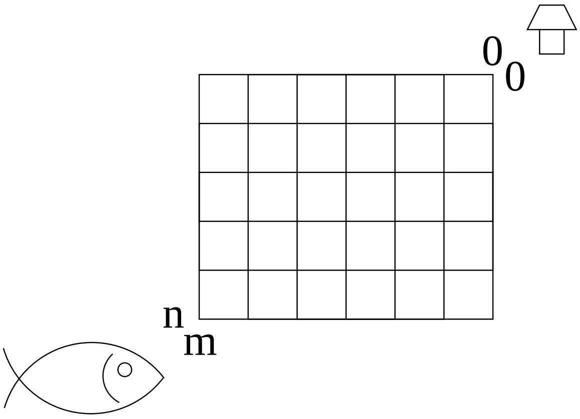
```
>>> a = [[[1],2],3,4]
>>> counting_deep_replace(a,2,3)
1
>>> a
[[[1], 3], 3, 4]

>>> counting_deep_replace(a,2,3)
0

>>> counting_deep_replace(a,3,5)
2
>>> a
[[[1], 5], 5, 4]

>>> counting_deep_replace(a,5,1)
2
>>> a
[[[1], 1], 1, 4]

>>> counting_deep_replace(a,1,9)
3
>>> a
[[[9], 9], 9, 4]
```

Give a possible implementation for `counting_deep_replace`. [4 marks]

```
def counting_deep_replace(lst,a,b):
```

Question 3: Finding Dory [24 marks]

In this question, you will help Dory (a fish) find her way home. For simplicity, we model the paths between Dory and home as an $m \times n$ grid as shown:



A. [**Warm Up**] We assume that Dory can only swim up or right and the number possible paths home is given by the function `paths`, which takes in 2 positive integers m and n as follows:

```
>>> paths(1,1)
2

>>> paths(2,2)
6

>>> paths(3,1)
4

>>> paths(3,2)
10

>>> paths(3,3)
20
```

Give a possible implementation for the function `paths`. [4 marks]

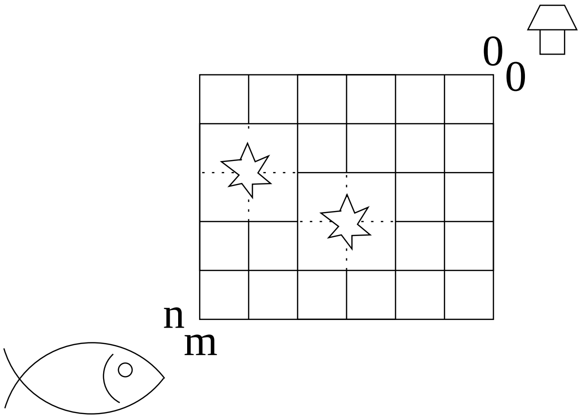
```
def paths(m,n):
```

B. What is the order of growth in time and space for the function `paths` that you implemented in Part(A)? Explain. [4 marks]

Time:

Space:

C. [**Blocked Paths**] But alas, life is never simple and Dory sometimes runs into some obstacles that blocks off some junctions. The function `blocked_paths` takes in *m* and *n* and a tuple of blocked junctions which are coordinate pairs (tuples) and returns the number of possible paths.

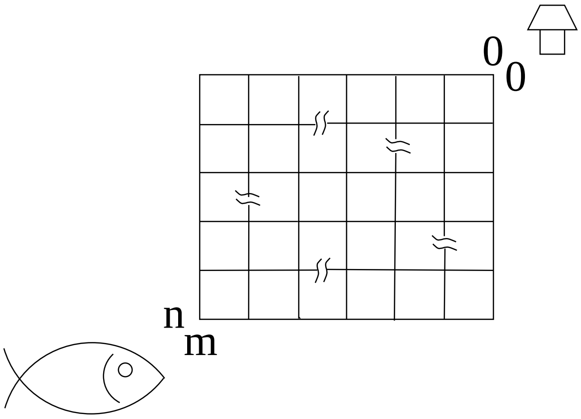


```
>>> blocked_paths(1,1,((0,0),))
0
>>> blocked_paths(1,1,((1,0),))
1
>>> blocked_paths(1,1,((0,1),))
1
>>> blocked_paths(1,1,((1,1),))
0
>>> blocked_paths(2,2,((1,1),))
2
>>> blocked_paths(2,2,((1,1),(0,1)))
1
```

Give a possible implementation for the function `blocked_paths`. [6 marks]

```
def blocked_paths(m,n,blocks):
```

D. [**Broken Paths**] Other times, obstacles only block off the path between junctions instead of the whole junction. The function `broken_paths` takes in m and n and a tuple of blocked connections which are sets of 4 coordinates $(from_x, from_y, to_x, to_y)$ which specifies which connections are broken.



```
>>> broken_paths(1,1,((1,1,1,0),))
1

>>> broken_paths(1,1,((1,1,0,1),))
1

>>> broken_paths(1,1,((1,0,0,0),))
1

>>> broken_paths(1,1,((0,1,0,0),))
1

>>> broken_paths(2,2,((0,1,0,0),))
3

>>> broken_paths(2,2,((1,1,0,1),))
4

>>> broken_paths(2,2,((1,1,0,1),(1,1,1,0)))
2
```

Give a possible implementation for the function `broken_paths`. [5 marks]

```
def broken_paths(m,n,blocks):
```

E. [**Dynamic Programming**] Give a Dynamic programming implementation for the function `broken_paths`. If your implementation in Part(D) is already DP and correct, you will get the marks to this question for free (power to you!) [5 marks]

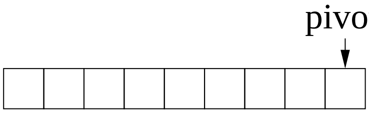
```
def dp_broken_paths(m,n,blocks):
```

Question 4: In-Place Quicksort [30 marks]

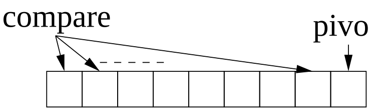
Quicksort is one of the classic sorts that we did not discuss in class. No problem. In this question, you will learn about it. How cool is that?

```
>>> a = [4,2,56,23,12,1,32,5,7]
>>> quicksort(a)
>>> a
[1, 2, 4, 5, 7, 12, 23, 32, 56]
```

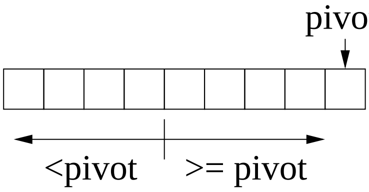
We call the last element of the list that we want to sort the *pivot*.



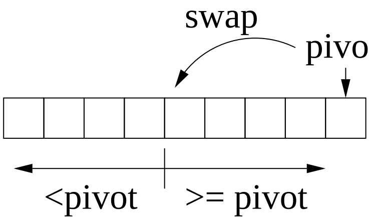
Next we compare first $n-1$ elements of the list with the pivot one at a time.



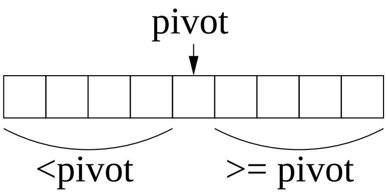
As we do so, we do some swaps so that by the time we are done, the elements are divided into 2 segments: the first segments are elments that are smaller than the pivot and the second are elements that are bigger or equal to the pivot.



We swap the pivot with the element at the boundary, so that the pivot is in between the two seqments.



We recursively apply the above process to the two segments on the left and right of the pivot. At some point, the list will get sorted.



Got it?

A. [**Sanity Check**] Explain what you understand by an *in-place* sort. [3 marks]

B. Implement the function `quicksort` that will do an in-place sort on an input list. **Hint:** If you really cannot figure out how to do this in-place, you can do it not-in-place and take a small penalty. [7 marks]

```
def quicksort(lst):
```

C. What is the order of growth in time and space for the sort you implemented in Part(B) in terms of the number of elements *n* in the average case? Explain. [4 marks]

Time:

Space:

D. What is the worst case order of growth in time? When does it happen? Explain. [3 marks]

E. Is the sort you implemented in Part(B) a *stable* sort? Explain. [3 marks]

F. Explain how you would modify the code in Part(B) to get a reverse-sorted list. Explaining in words is good enough. Don't need to write code here. [3 marks]

G. [*Quicksort in C*] Consider the following snippet of code in C:

```
#include <stdio.h>

void quicksort(int lst[],int start,int end);

void print_array(int a[], int length){
    printf("[");
    for (int i=0; i<length; i++){
        if (i!=length-1) {
            printf(" %d,", a[i]);
        } else{
            printf(" %d", a[i]);
        }
    }
    printf("]\n");
}

int main()
{
    int a[] = {4,2,56,23,12,1,32,5,7};
    quicksort(a,0,8);
    print_array(a,9);
    return 0;
}
```

Give a possible implementation for quicksort in C so that the above code works. Basically, translate your code in Part(B) to C! [7 marks]

```
void quicksort(int lst[],int start,int end)
```

Question 5: How Do We Do The Right Thing? [4 marks]

In his article *"How Will You Measure Your Life?,"* Professor Clayton Christensen recounted the following story:

> *I'd like to share a story about how I came to understand the potential damage of "just this once" in my own life. I played on the Oxford University varsity basketball team. We worked our tails off and finished the season undefeated. The guys on the team were the best friends I've ever had in my life. We got to the British equivalent of the NCAA tournament – and made it to the final four. It turned out the championship game was scheduled to be played on a Sunday. I had made a personal commitment to God at age 16 that I would never play ball on Sunday. So I went to the coach and explained my problem. He was incredulous. My teammates were, too, because I was the starting center. Every one of the guys on the team came to me and said, "You've got to play. Can't you break the rule just this one time?"*
>
> *I'm a deeply religious man, so I went away and prayed about what I should do. I got a very clear feeling that I shouldn't break my commitment–so I didn't play in the championship game.*
>
> *In many ways that was a small decision–involving one of several thousand Sundays in my life. In theory, surely I could have crossed over the line just that one time and then not done it again. But looking back on it, resisting the temptation whose logic was "In this extenuating circumstance, just this once, it's OK" has proven to be one of the most important decisions of my life. Why? My life has been one unending stream of extenuating circumstances. Had I crossed the line that one time, I would have done it over and over in the years that followed.*
>
> *The lesson I learned from this is that it's easier to hold to your principles 100% of the time than it is to hold to them 98% of the time. If you give in to "just this once," based on a marginal cost analysis, as some of my former classmates have done, you'll regret where you end up. You've got to define for yourself what you stand for and draw the line in a safe place."*

If you have come across a situation like this in your own life, describe it and tell us how you found the inner strength to do the right thing. Or if you didn't, repent now and tell us how you think you would have convinced yourself to do better if you had the chance to relive that moment. If not, think of a situation where you will be tempted to "take the slippery slope," and explain how you would convince yourself to **do the right thing** and not the convenient thing.

— END OF PAPER —

Scratch Paper

– HAPPY HOLIDAYS! –