

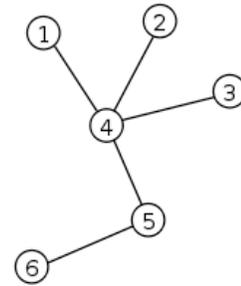
Final Exam CS10 Summer 2017

Name: Final Exam Solution Student ID: _____ Lab TA: Jobel Angela

Q1: Graph Terminology

Which of the following terms could describe the graph to the right?
(select all that apply)

- | | |
|------------------------------------------|---------------------------------------------|
| <input checked="" type="checkbox"/> Tree | <input checked="" type="checkbox"/> Acyclic |
| <input type="checkbox"/> Disconnected | <input type="checkbox"/> Directed |



Q2: Quantum Computers (Week 8 Reading)

Which of the following is true about quantum computers, according to the article “Here’s why we should be really excited about quantum computers”? (select all that apply)

- A quantum bit can exist as a 1, or a 0, or both at the same time.
- Quantum computers could revolutionize entire industries.
- Quantum computers are better than normal computers at performing calculations with many numbers.
- Quantum computers can be programmed to act similarly to human brains.

Q3: P vs. NP (Week 8 Reading)

According to the video “P vs. NP and the Computational Complexity Zoo”, which of the following best describes the P and NP classes of problems? (pick one)

- P problems’ solutions can be verified quickly; NP problems can be solved quickly.
- P problems can be solved quickly; NP problems cannot be solved quickly.
- P problems can be solved quickly; NP problems’ solutions can be verified quickly.
- P problems’ solutions can be verified quickly; NP problems’ solutions cannot be verified quickly.

Q4: What is Information Theory? (Week 8 Reading)

What do a song, a telegraph, an email, and a drawing have in common, according to the video? (pick one)

- They are made of bits.
- They are universal forms of communication.
- They can carry the same amount of information.
- They have the same information density.

Q5: UI and Design

Which of the following steps describe the iterative design cycle in order? (pick one)

- Design, Test & Evaluate, Prototype, Repeat
- Prototype, Design, Test & Evaluate, Repeat
- Prototype, Simplify, Test & Evaluate, Repeat
- Design, Simplify, Test & Evaluate, Repeat
- Design, Prototype, Test & Evaluate, Repeat

Q6: Internet & IP Addresses

Which of the following are true statements about IP addresses? (select all that apply)

- Everything that connects to the internet needs an IP address
- We ran out of unique IPv4 addresses
- DNS translates domain names like “facebook.com” to IP addresses
- We will probably run out of IPv6 addresses in the next five years

Q7: Runtime the Jewels

Student ID: _____

You will determine the runtime of the following expression in terms of the size (length) of each input separately. `func` takes linear time in terms of the length of its list input, and constant time in terms of the length of its text input.



What is the runtime complexity of the expression in terms of the length of **data A** ?

<i>constant</i> <input type="radio"/>	<i>logarithmic</i> <input type="radio"/>	<i>linear</i> <input checked="" type="radio"/>	<i>quadratic</i> <input type="radio"/>	<i>exponential</i> <input type="radio"/>	<i>something else</i> <input type="radio"/>
------------------------------------------	---------------------------------------------	---------------------------------------------------	-------------------------------------------	---------------------------------------------	------------------------------------------------

What is the runtime complexity of the expression in terms of the length of the word **word** ?

<i>constant</i> <input checked="" type="radio"/>	<i>logarithmic</i> <input type="radio"/>	<i>linear</i> <input type="radio"/>	<i>quadratic</i> <input type="radio"/>	<i>exponential</i> <input type="radio"/>	<i>something else</i> <input type="radio"/>
-----------------------------------------------------	---------------------------------------------	----------------------------------------	-------------------------------------------	---------------------------------------------	------------------------------------------------

What is the runtime complexity of the expression in terms of the length of **data B** ?

<i>constant</i> <input type="radio"/>	<i>logarithmic</i> <input type="radio"/>	<i>linear</i> <input checked="" type="radio"/>	<i>quadratic</i> <input type="radio"/>	<i>exponential</i> <input type="radio"/>	<i>something else</i> <input type="radio"/>
------------------------------------------	---------------------------------------------	---------------------------------------------------	-------------------------------------------	---------------------------------------------	------------------------------------------------

Replace **data B** with a second instance of **data A**. What is the runtime complexity in terms of **data A** ?

<i>constant</i> <input type="radio"/>	<i>logarithmic</i> <input type="radio"/>	<i>linear</i> <input type="radio"/>	<i>quadratic</i> <input checked="" type="radio"/>	<i>exponential</i> <input type="radio"/>	<i>something else</i> <input type="radio"/>
------------------------------------------	---------------------------------------------	----------------------------------------	------------------------------------------------------	---------------------------------------------	------------------------------------------------

Q8: A Game of San Francisco vs. Oakland

A man and a woman are planning a trip. They can each choose to go to either San Francisco or Oakland. Their utilities for every outcome are shown below. Which outcome is an equilibrium? (pick one)

Hint: It might help to first consider the man's optimal strategy.

- A
- B
- C
- D

		Woman's choice	
Man's choice		A. 2 0	B. 0 0
		C. 0 1	D. 1 1

Q9: Distributed Computing

According to Alex Mckinney's lecture on distributed computing, which of the following is a reason why *idempotence* is important in a distributed program? (pick one)

- Machines in datacenters may fail unexpectedly, so you must have multiple copies of the same data.
- Malicious users could take advantage of your system by sending many requests at the same time, so you must make sure only one request actually gets executed.
- Having internet connection 99.99% of the time is not guaranteed, so you must make a program that continues running when the network is down.
- Distributed programs send user requests to datacenters around the world, so you must make sure your code runs correctly independent of the machine it's run on.

Q10: Con-What Cur-Is Ren-My Cy-Name

Which of the following could be the value of global variable `my_name` when the green flag is clicked and after both scripts below finish executing? (select all that apply)

when clicked

wait pick random 1 to 3 secs

set my name to Yifat

wait pick random 1 to 3 secs

set my name to join my name Bear

when clicked

wait pick random 1 to 3 secs

set my name to Steven

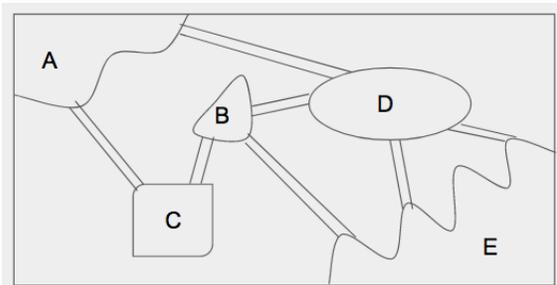
wait pick random 1 to 3 secs

set my name to join my name Oski

Yifat	Steven	Yifat Bear	Steven Oski	Yifat BearOski	Steven Yifat	Oski	Yifat OskiBear
<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Q11: Graph Abstraction

Which of the following undirected graphs correctly abstracts the following map of islands and bridges? (pick one)



-
-
-
-

Q12 a): Setting You Up For Success

A set is similar to a list, with one major difference: it cannot contain duplicate values. If you try to add a value to a set that already contains that value, no value should be added. Furthermore, sets are *unordered*, so we do not care about the positions of values in a set. **empty_set** reports an empty set.

First, we'll write the block `add_value_to_set`. Some example inputs and outputs are shown to the right (the output appears like a list, but we will treat it as a set).

Complete code using the blanks for A, B, and C.

```

+add+ value +to+ set+ set : +
if
  report
else
  report
  
```

A → if
B → report (inside if)
C → report (inside else)

```
add a to set list a b c
```

```
add z to set list a b c
```

```
item 1 of set
```

```
set contains value
```

```
set
```

```
value in front of set
```

```
item 1 of set = value
```

```
empty set
```

```
empty? set
```

A	B	C
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>
<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Next, we'll write the `set_from_list` block. It takes a list as input, and reports a set. In other words, it **removes duplicate values from the list**. Remember that sets are *unordered*, so we don't care if the order of the values change when we run operations.

Some example inputs and outputs are shown to the right. Complete the blanks for D, E, F, and G so that the block works as described.

Points for this problem do not rely on correctness of the previous block.

```

+set+ from+ list+ list : +
if
  report
else
  report add to set
  
```

D → if
E → report (inside if)
F → add
G → to set

```
set from list list
```

```
set from list list 1 2 3
```

```
set from list list 1 2 1 3
```

```
item 1 of list
```

```
length of list = length of set from list list
```

```
set from list item 1 of list
```

```
all but first of list
```

```
empty? list
```

```
empty set
```

```
set from list all but first of list
```

D	E	F	G
<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>

The *union* of two sets, A and B, is a new set that contains items found in just A, just B, or both A and B. Here are two examples of union:

list [3, 1, 2] **union** **list** [1, 2, 4] => [3, 1, 2, 4]

list [] **union** **list** [1, 2, 4] => [1, 2, 4]

b) The *intersection* of two sets, A and B, is a new set that contains only items found in *both* A and B. Please fill in the block definition to achieve this operation. You may use previous whatever Snap! blocks you'd like, including those introduced in problem 12. You may not need all of the lines.

list [1, 2, 3] **intersection** **list** [7, 2, 1]

example

+ **a** **:** **+** **intersection** **+** **b** **:** **+**

Possible Solution:

keep such that < (a) contains () > and < (b) contains () >> from ((a) union (b))

Possible Solution:

keep such that < (a) contains () > from (b)

c) The *difference* of two sets, A and B, is a new set that contains only items found in just A or just B. If an item is in both A and B, it is not included. Please fill in the block definition to achieve this operation. You may use previous whatever Snap! blocks you'd like, including those introduced in problem 12. You may not need the xor variable.

list [1, 2, 3] **difference** **list** [7, 2, 1]

example

+ **a** **:** **+** **difference** **+** **b** **:** **+**

script variables **xor**

set **xor** **to** **#1** **and** **not** **#2** **or** **not** **#1** **and** **#2**

Possible Solution: keep such that <not <((a) intersection (b)) contains ()>> from ((a) union (b))

Q13: Debugging in Python

We have attempted to write a function, `index(item, lst)`, that returns the index of `item` in the list `lst`. Here are a few examples of how the function should work:

```
>>> my_list = [2, 3, 4, 5, 6]
>>> index(2, my_list)           # 2 is in the beginning of the list
0
>>> index(5, my_list)           # 5 is in the middle of the list
3
>>> index(7, my_list)           # 7 is not in the list
-1
```

We begin by writing the function using iteration, but there seems to be a bug.

```
def index(item, lst):
    for i in range(0, len(lst)):
        if lst[i] == item:
            return i
        else:
            return -1
```

- a)** What will the buggy code above output for the following call? (pick one) -1 0 1 Error
- ```
>>> index(2, [1, 2, 3])
```

- b)** Briefly describe the bug in the code above. Behavior: The function will only check the first item of the list

Also, the "return -1" should be after the for-loop, not inside of it, with no else statement necessary inside the loop.

We decide to rewrite the function using a different form of iteration, but unfortunately there's another bug.

```
def index(item, lst):
 i = 0
 current = lst[i]
 while current != item:
 i = i + 1
 current = lst[i]
 if current == item:
 return i
 else:
 return -1
```

- c)** What will the buggy code above output for the following call? (pick one)  -1  0  1  Error
- ```
>>> index(4, [1, 2, 3])
```

- d)** Briefly describe the bug in the code above. If "item" is not in "lst", "i" will increase past the range of the indices of "lst", causing an indexing error at the line in the loop that says "current = lst[i]".

Yet again, we decide to rewrite the function. This time we use recursion. Unfortunately there's another bug.

```
def index(item, lst):
    if lst[0] == item:
        return 0
    else:
        return 1 + index(item, lst[1:])
```

e) Fill in the blanks below with an example of inputs to the function that would illustrate the bug in the buggy code above. Then explain why those inputs illustrate the bug.

index(1 , []) would illustrate the bug because

An index error occurs when "lst" is empty and we call lst[0] in the if-statement (base case). This is

fixed by adding a check for an empty "lst" or "item in lst" in the base case.

f) On the line below, implement index using only a Python list comprehension. Unlike before, the index function should return a list of all of the indices at which item appears in lst. If item is not in lst, the function should return an empty list.

```
def index(item, lst):
    return [i for i in range (len(lst)) if lst[i] == item]
```

Q14: Slice Slice Baby

In Python, we have the concept of *slicing*, where we can use indices to extract a substring. Let's make our own slicing operation in Snap!. Here are some examples of how this operation should behave:



Complete the code so that the slice function works as above. You can assume that the inputs to the block are valid.

Note that the indices are treated as inclusive, and indexing starts at 1 like usual in Snap!.

You may find the following two blocks useful. You are free to use other blocks as well.

all but first letter of

all but last letter of

i. word

ii. slice (all but first letter of (word) from ((a)-1) to ((b)-1))

iii. slice (all but last letter of (word) from (a) to (b))

Q15: Say Cheese

A mouse is in a maze, and is searching for the cheese. To find the cheese, the mouse will use a Snap! block. The strategy involves leaving crumbs so that it knows where it's been.

	Reports true if the given tile has cheese on it.
	Places a crumb on the given tile.
	Reports true if the given tile has a crumb on it.
	Reports a list of all tiles that are next to the given tile in random order. Squares colored black will not be included.

```

+ find + cheese + starting + from + tile +
if tile has cheese?
  stop all
leave crumb at tile
for each neighbor of neighboring tiles tile
  if not crumb at neighbor ?
    find cheese starting from neighbor
  
```

To the right is an potential result of the the algorithm. The mouse starts from the tile marked “S”, and there is cheese on the tile marked “C”. Tiles are marked in the order that crumbs were left on them. If a tile does not have a number, no crumb was left on it.

				C
		5	6	7
		4		
		3		
S	1	2		

Please mark two *different, unique* potential results of the algorithm in the same fashion. Blank mazes are provided below.

a) Possible solution:

				C
		8	9	10
		7		
		3	4	5
S	1	2		6

b) Possible solution:

8	7	6		C
9		5	11	12
10		4		
		3		
S	1	2		

c) How many ways are there for the mouse to find the cheese using this algorithm in this scenario?

_____ 5 _____

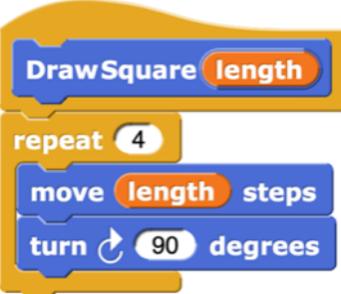
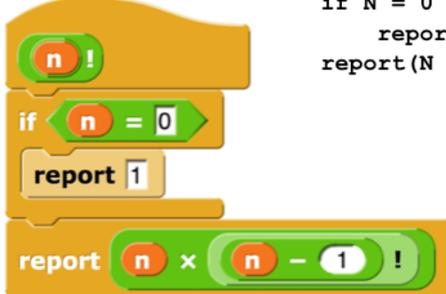
d) Which search algorithm does this code best represent?

_____ Depth-First Search _____

Writing *Snap!* code on paper (supplementary)

You will be asked to write *Snap!* code on this exam, so we've developed a technique for writing it on paper. There are a few key things to notice:

- We often write variables in **UPPERCASE**.
- We change spaces between words in block names to dashes (this makes it much easier to read).
- We use indentation just as *Snap!* does, to help us understand what is "inside" the **if**, **else**, and other Control structures. E.g., here's how you could write the **DrawSquare** and **n!** blocks:

 <pre> Draw-Square (LENGTH) repeat (4) move (LENGTH) steps turn-right (90) degrees </pre>	 <pre> (N)! if N = 0 report (1) report (N * (N - 1)!) </pre>
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------

- When you want to write a list of things, write them with an open parenthesis, then the first item, second item, etc (separated by spaces) and when you're done, put a closed parenthesis. If any of your items are a sentence, you have to put quotes around the sentence. So, for example, the following list of three things would be written as the equivalent 3-element-list:



- (life liberty "pursuit of happiness").

- Similarly, a nested list just shows up as a nested set of parenthesis. So the following would be written as
- ((Love 5) (Hate 4) (The 10)).



- If you want to pass in a function as argument, you know the function must be surrounded by a grey-border. Here are three new conventions:
 - The grey border is written as *square brackets*: []
 - Blanks are written as parenthesis with underscore `_` in the middle, but common blocks that are passed in to HOFs can be simplified by just their name (and not the parens and underscores)
 - Return values are written as `→ value`
- So the following would be written as:
 - `Combine[(_)+(_)]items-of (Map[(_) x (_)]over ((1 20 3 10)))`
- or, in a more simplified (and preferred) format, also showing return value:
 - `Combine[+]items-of (Map[x]over ((1 20 3 10)) → 51`



510

A bunch of Snap blocks are shown below as a reference. For coding problems on this exam, unless the problem says otherwise, you may use any Snap! block, not just the ones below (we've omitted lots of them, like +, -, split, etc.)

