Addis Coder Quiz 3

Problem 1

Which of the following describe useful criteria for comparing the efficiency of algorithms?

a) Time complexity
b) Memory complexity
c) Both of the above

Problem 2

What is the time compexity of the following code given below?

In [ ]:

```
#Given code

value = 0
for i in range(n):
    for j in range(i):
        value += 1
```

a) $O(1)$

b) $O(n)$

c) $O(n^2)$

d) $O(\log n)$

Problem 3

How is time complexity measured?

a) By counting the total number of loops in a program.

b) By counting the number of primitive operations performed by the algorithm on a given input size.

c) By counting the size of data input to the algorithm.

d) By counting the number of lines of code in a program.

Problem 4

What is the time complexity of the following code? **ConstFun(n)** has constant time compexity.

In [ ]:

```
summ = 1
for i in range(n):
    ConstFun(n)
    summ += i
print(summ)
```

a) $O(1)$

b) $O(n)$

c) $O(n^2)$

d) $O(\log n)$

Problem 5

What is the **time complexity** of the following code? **LinearFun(n)** has linear time compexity and **ConstFun(n)** has constant time complexity.

In [ ]:

```
number_string = ''
for i in range(n):
    ConstFun(n)
    number_string += str(i)
    LinearFun(n)
    number_string += ' '

print(number_string)
```

a) $O(1)$

b) $O(n)$

c) $O(n^2)$

d) $O(\log n)$

Problem 6

Which of the following functions has the **highest time** complexity for large 'n'?

a) $5n + 10$

b) $4n^2 + 6n + 14$

c) $10n$

d) $n^4 + n^2 + n$

Problem 7

Write out the **Big-O time complexity** for all the options in **Problem 6**.

a)

b)

c)

d)

Problem 8

What is the running time of the following code in terms of n?

In [ ]:

```
y = 100

for i in range(n):
    for k in range(n):
        for j in range(5):
            y //= 2
```

a) $O(\log(n))$

b) $O(n^2)$

c) $O(n^2 \log(n))$

d0 $O(n^3)$

Problem 9

Choose the best answer for the following questions?

I) Which sorting algorithm typically has the best time complexity for large datasets?

a) Bubble Sort

b) Selection Sort

c) Merge Sort

d) Insertion Sort

II) In which scenario is Bubble Sort the most efficient?

a) When the list is already sorted in correct order

b) When the list is sorted in opposite order

c) When the list contains a large number of elements

Problem 10

Suppose we want to implement a function `foo(x, L)` in which `L` is sorted. This function should return `True` if `x` is in `L` and return `False` otherwise. `x` is an `int` and `L` is a list of ints.

For instance `foo(2, [0,2,6])` should return True.

Consider the following code:

```
In [ ]: # check if x is in L[A:B]
def Search(x, L, A, B):
    if B < A+1:
        return False
    else:
        mid = (A+B)//2
        if L[mid] == x:
            return True
        elif L[mid] > x:
            return Search(x, L, A, mid)
        else:
            return Search(x, L, mid, B)

def foo(x, L):
    return Search(x, L, 0, len(L))

foo(1,[0])
```

```
KeyboardInterrupt
```

What **error** might the following code give if run? **Hint:** what happens if you run `foo(1, [0])` ?

A. maximum recursion depth exceeded

B. index out of range error

C. No error will be printed by Python, the code is correct.

D. No error will be printed by Python, but there is a mistake in the code so sometimes it gives the wrong answer.

Problem 11

What is the **time complexity** of the `Search(x, L, 0, len(L))` function in **Problem 10** if `L` is a sorted list of length `n` and `x` is in `L` ?

In [ ]:

```

```

Problem 12

Write a function that takes a list of integers and returns the position (index) of the number 34 or -1 if 34 is not part of the list.

for example: if `L=[5,67,34]`, it would return 2 and if `L=[23,33,89,-34]` it would return -1.

In [ ]:

```python
def find_34(L):
```

Problem 13

Write a function that takes an unsorted list `L` and returns `L` sorted in **descending** order.

For example, if you give the function the list `L= [34,-3,98,-100]`, **Sorting(L,n)** would return `[98,34,-3,-100]`

In [ ]:

```python
def SortingDescending(L):
```

Problem 14

As a Teff trader, you want to **buy** Teff for a **low price** and later **sell** it at a **high price**. You can only buy and sell **one time**.

For example, if you have `prices = [1200, 1300, 1100, 900, 1100, 1000, 1150]`, the best strategy is to **buy** when the price is `900` and **sell** when the price is `1150`, for a **profit** of `1150-900 = 250`.

Write a function that takes a **list of Teff prices** and returns the **highest profit** you can make.

In [ ]:

```python
def highest_profit(prices):
```

Problem 15

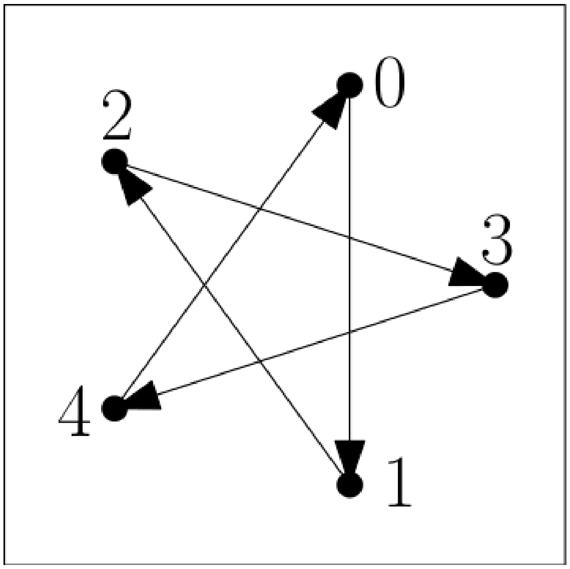
a) **Draw** the following graph: `{0: [1, 3], 1: [0, 2], 2: [0], 3: [1]}`

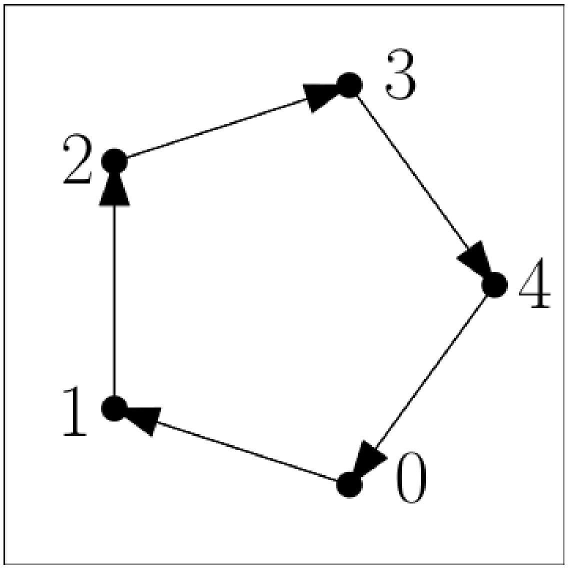
b) What is the **list of edges** of this graph?

In [ ]:

Problem 16

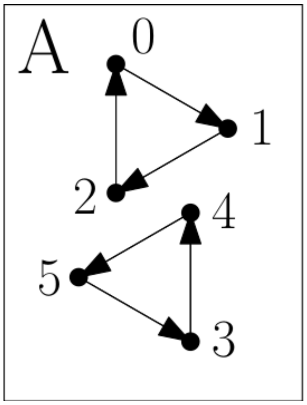
a) Are the following two graphs the same? (Hint: two graphs are the same if their **list of edges** is the same.)

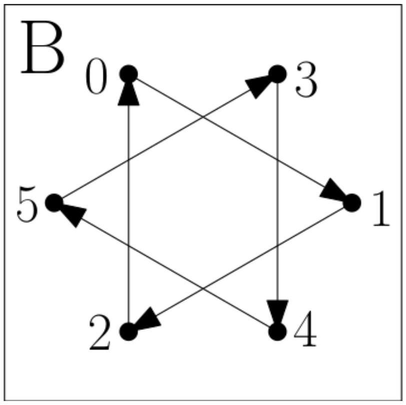


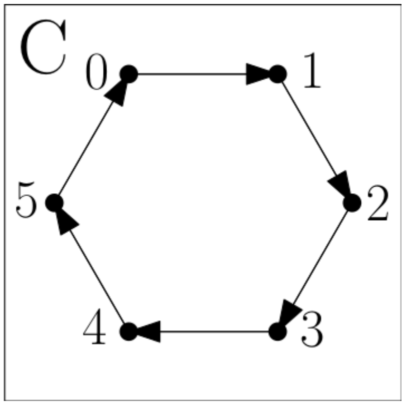


In [ ]:

b) Which of the following graphs are the same?





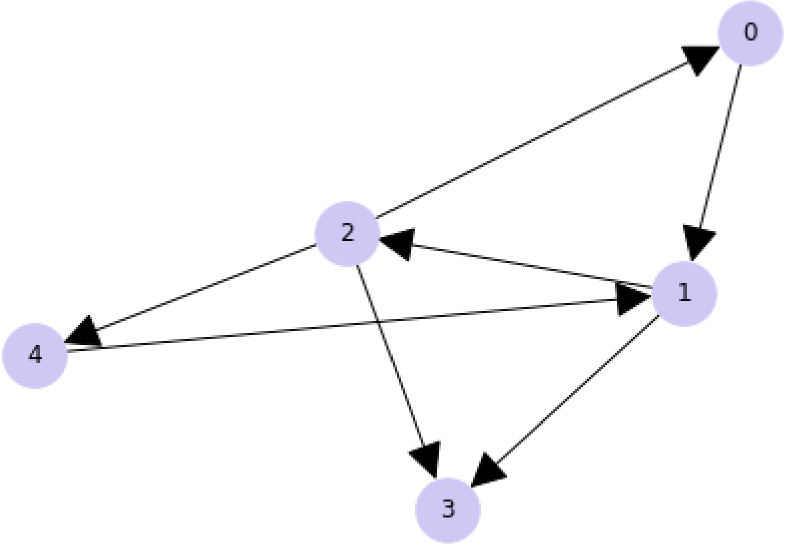


In [ ]:

Problem 17

Write a function `neighbors(G, s)` that takes a graph `G` as **list of edges** and a node `s`, find all out-neighbors (successors) of `s`.

Example: For the graph `G = [[2, 0], [0, 1], [2, 3], [1, 3], [4, 1], [2, 4], [1, 2]]`, `neighbors(G, 2)` should return `[0, 3, 4]`.



In [ ]:

Problem 18

Note: 1 bonus point for this problem if you can find an $O(n)$ solution.

In the list `[2, 1, 4, 7, 3, 5]`, the elements `2`, `4` and `7` are **bigger than all previous** elements: `4 > 2` and `4 > 1`; `7 > 2` and `7 > 1` and `7 > 4`.

a) **Write a function** `elimination_sort(lst)` that takes a list `lst` and returns a **new list** containing those elements **bigger than all previous** elements.

Examples:

- `elimination_sort([2, 1, 4, 7, 3, 5])` should return `[2, 4, 7]`
- `elimination_sort([1, 6, 5, 4, 7, 9])` should return `[1, 6, 7, 9]`

In [ ]:

b) What is the time complexity of your algorithm, if n is the length of `lst`?

In [ ]:

```
```