Week 3 – Counting

3-1. Find the number of ways in which the letters of the word ABCDEF can be arranged if A and B must not be next to one another.

a) 72
b) 240
c) 480
d) 720

3-2. Find the number of ways in which 6 people A,B,C,D,E,F can be seated at a round table such that A and B must not sit together. Left and right neighbours are considered different.

a) 72
b) 96
c) 36
d) 48

3-3. How many positive integers between 1 and 500 inclusive are divisible by 3 or 5?

a) 266
b) 233
c) 166
d) 33

3-4. Covid-19 regulations state that groups of more than 5 are not allowed to dine in, even if seated separately. Suppose there is a group of friends of 5 women and 6 men. How many ways can we select 5 diners such that there is at least 1 man?

a) 55,440
b) 55,320
c) 461
d) 6

3-5. How many bit strings (0 or 1) of length five contain at least two consecutive 0s?

a) 32
b) 24
c) 19
d) 20

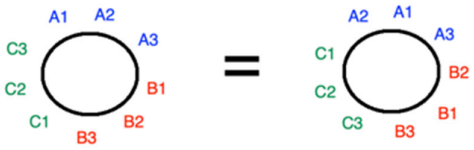
3-6. An ID number ranges from 1 to 4 digits. How many possible IDs are there if the number cannot start or end with 0.

a) 7380
b) 9000
c) 8100
d) 77,244

3-7. 3 boys and 3 girls seated together in a row of 6 chairs. Given that Amanda (one of the 3 girls) is sitting at the end, how many possible ways are there for her sitting next to another girl?

a) 8
b) 48
c) 480
d) 600

3-8. How many ways are there to seat 9 people around a circular table where they are grouped in 3s? The 3 people can sit anywhere as long as they sit with their group.



a) 2
b) 12
c) 6720
d) 84

3-9. How many unique ways can we arrange the letters in the word "APPLE"?

a) 120
b) 60
c) 118
d) 58

3-10. A card deck contains 52 cards, comprising 13 ranks (from 2 to 10, and then Jack, Queen, King, Ace) with 4 suits each (Hearts, Diamonds, Spades, Clubs). A pair contains 2 cards of the same rank & of any suit (e.g. 2 Kings - one Spades, the other Hearts). How many possible pair combinations are there?

a) 13 * 2
b) 13 * 4
c) 13 * 4c2
d) 13 * 4p2

Week 4 – Complexity

4-1. Given that g has a complexity of O(n), what is the complexity of f?

```
def f(n):
  i = 1
  while i < n+1:
    for j in range(2,22):
      for k in range(n, 1, -2):
        g(n)
    i = i * 2
```

a) O(n^2)
b) O(n^3)
c) O($n^2 log(n)$)
d) O(n^{2n})

4-2. The following expression provides the number of steps taken by an algorithm to solve a problem of size *n*. What is the simplified Big O complexity?

$$[log(n^{12}) - log(n^2)] \times \frac{3}{2}(n) + (log 100)^2 \times log(n^{20})$$

a) $O(nlog(n^{10}))$
b) $O(log(n))$
c) $O(nlog(\mathrm{n}))$
d) $O(n)$

4-3. f(n) has O(n) complexity; g(n) has O(n^2) complexity. What is the complexity of the function below?

```
def odd_even(n):
  for i in range(n):
    if i%2==0:
      f(n) # has O(n) complexity
    else:
      g(n) # has O(n^2) complexity
```

a) O(n^2)
b) O(n^3)
c) O($\mathrm{n}^2/2 + \mathrm{n}^3/2$)
d) O(n^6)

4-4. list_m contains integers and a is a positive integer. What is the number of <u>assignment operations</u> in sum_of_multiple?

```
def sum_of_multiple(list_m, a):
  sum = 0
```

```
for i in list_m:
  if i % a == 0:
    sum += i
return sum
```

a) 2 * len(list_m)
b) 2 * (no. of multiples of a) + 1
c) 1
d) no. of multiples of a + 1

4-5. Which of the following options provides a decreasing order of asymptotic complexity of functions f1, f2 & f3?

Function	no. of steps
f1(n)	$3n * 2n^3 + 7n$
f2(n)	$(3n)! + 6n^2 + n(\log_2 n)$
f3(n)	$15 \log(n^2) + 20^3 + 7(\log n)^2$

a) f2 > f3 > f1
b) f2 > f1 > f3
c) f2 > f3 > f1
d) f1 > f3 > f2

4-6. Which of the following statements is true? (Choose 1)

a) Characterising the worst case Big O is preferred over the best case and average case.
b) Big O is determined by counting the size of data input to the algorithm
c) $O(n + m^2)$ can be reduced to $O(m^2)$
d) Out of the O(n log n), O(2n), $O(n^2)$, O(log n), the slowest time complexity is $O(n^2)$.

4-7. n = length of no_list. Which is the best representation of the big-O time complexity of this algo?

```
def f(no_list):

  # Print all the numbers in no_list
  for no in no_list:
    print no

  # Print the sum of all pairs
  for no1 in no_list:
    for no2 in no_list:
      print no1 + no2
```

a) $O(n^2)$
b) $O(n \times n^2)$

c) $O(n + n^2)$
d) $O(n^3)$

4-8. What is the Big O complexity of the function below?

```
def f(n):
  total = 0
  for i in range(0, n, 2):
    for j in range(n):
      if j % 2 == 0:
        total = total + 2 + j
  return total
```

a) $O(n^2 + n)$
b) O(log n)
c) $O(n^3)$
d) $O(n^2)$

4-9. List the following time complexity functions in **decreasing order** of asymptotic complexity.

i. $\log (100n)^3$
ii. $100n^2$
iii. $2^{2n+1} + 100$
iv. n log (n)
v. $100n + \log (5n)^2$

a) iii, ii, v, iv, i
b) iii, ii, iv, v, i
c) ii, iii, iv, v, i
d) ii, iii, v, iv, i

4-10. An image is represented by a 2D array of pixels.

```
pixel = [ [1,0], [0,1], ... ]
```

"1" & "0" represent a white pixel & a black pixel respectively.

You use a nested for loop to iterate through every pixel:

```
for i in range(len(pixel)):
  for j in range(len(pixel[i])):
    f(pixel[i][j])  #  f is an O(1) function
```

What is the time complexity of the algorithm when the image is considered as the input, where n represents the number of pixels?

a) O(n)
b) $O(n^2)$
c) O(2n)
d) $O(n^4)$

Week 5 – Iteration & Decomposition

5-1. a = [3, 7, 8, 13, 17, 53, 91, 92, 70]

```
def search (a, k):
  lower = -1
  upper = len(a)
  while (lower + 1 != upper):
    mid = (lower + upper) // 2
    if k == a[mid]:
      return mid
    elif k < a[mid]:
      upper = mid
    else:
      lower = mid
  return None
```

When **search(a, 3)** is called, what is the final value of **lower**, **upper** and **mid**?

a) -1, 1, 0
b) -1, 4, 1
c) 0, 2, 1
d) -1, 3, 1

5-2. Consider the following list of partially sorted numbers (using insertion sort):

[18 30 43 56 95 || 41 28]

The double bars represent the sort marker. How many comparisons and swaps are needed to sort the next number (41)?

a) 3 comparisons, 2 swaps
b) 4 comparisons, 3 swaps
c) 4 comparisons, 4 swaps
d) 2 comparisons, 2 swaps

5-3. Given the function below:

```
def moveleft(a, i):
  while a[i] < a[i-1]:
    a[i], a[i-1] = a[i-1], a[i]
    i -= 1
```

What is the value of the **num_array** after the function **moveleft** is executed like this?

```
num_array = [70,90,80,30]
moveleft(num_array, 2)
```

a) [70, 80, 90, 30]
b) [70, 90, 80, 30]
c) [70, 90, 30, 80]
d) [30, 70, 80, 90]

5-4. Given a = ['Apple', 'Banana', 'Orange', 'Mango', 'Peach', 'Yam'], how many comparisons will iterative merge sort make?

a) 7
b) 8
c) 9
d) 10

5-5. Given a = [1, 3, 10, 35, 40, 46, 57, 67], how many comparisons will be made for linear search and binary search respectively to find the number 67?

a) Linear: 7, Binary: 3
b) Linear: 8, Binary: 3
c) Linear: 8, Binary: 4
d) Linear: 7, Binary: 4

5-6. Using insertion sort, how many comparisons are required to sort the following array in ascending order? [23, 56, 38, 10, 72, 65]

a) 0 + 1 + 3 + 2 + 2 = 8
b) 1 + 2 + 3 + 1 + 2 = 9
c) 0 + 1 + 2 + 1 + 1 = 5
d) 1 + 1 + 2 + 1 + 1 = 6

5-7. You have an array containing 78 **even** numbers. How many items in the array must be searched using a binary search to find out that the number '33' is not in the array?

a) 7
b) 8
c) 6
d) 1

5-8. How many comparisons are required to sort this array using merge sort?

[7, 11, 9, 12, 5, 2, 10]

a) 13
b) 11
c) 14
d) 17

5-9. Given an array arr = [45, 77, 89, 90, 94, 99, 100] and the key = 99, using binary search, what are the values of **mid** in the 1st and 2nd iteration ?

a) 90 and 99

b) 90 and 94
c) 89 and 99
d) 89 and 94

5-10. Given array = [10, 64, 28, 83, 97, 19, 42, 72, 37, 42, 20] and key = 42. Using linear search, what will the algorithm return?

a) 10
b) 7
c) 6
d) 9

Week 6 – Recursion

6-1. Consider this function:

```
def convert(n, base):
  s = "0123456789ABCDEF"
  if n < base:
    return s[n]
  else:
    return convert(n // base, base) + s[n % base]
```

What will this return? `convert(2835, 16)`

a) B12
b) C13
c) B13
d) A13

6-2. Consider this function. What will be printed out?

```
def practice(a,n):
  if n == 1:
    return a[0]
  else:
    x = practice(a, n-1)

  if x > a[n-1]:
    return x
  else:
    return a[n-1]

arr = [2,28,55,99]
print(practice(arr,4))
```

What will be printed out?

a) 2
b) 28
c) 55
d) 99

6-3. Function sum_series calculates the sum of the integers in this series: **n**, **n-2**, **n-4**, **n-6** ..... (series continues as long as **n** is positive). What will be the reduction step? Fill in the blank below:

```
# e.g.:
# sum_series(10) = 10 + 8 + 6 + 4 + 2
# sum_series(5) = 5 + 3 + 1
```

```
def sum_series(n):
  if n < 1:
    return 0
  else:
    return ______
```

a) sum_series(n + 2)
b) sum_series(n - 2)
c) n + sum_series(n - 2)
d) n + sum_series(n + 2)

6-4. What is the complexity of the function below?

```
def f(k, n):
  if n == 0:
    return 1
  elif n % 2 == 0:
    return f(k, n//2) * f(k, n//2)
  else:
    return k * f(k, n//2 ) * f(k, n//2)
```

a) O(n)
b) $O(n^2)$
c) O(2n)
d) O(log n)

6-5. What is the output of the following python code?

```
def tryme(n):
  if(n > 0):
    tryme(n - 1)
    print(n, end = " ")
    tryme(n - 1)

tryme(3)
```

a) 1 3 2 1 3 2 1
b) 1 2 3 3 2 1 1
c) 1 2 1 3 1 2 1
d) 1 2 3 3 1 2 1

6-6. What will the following call return? → **`recursivefunc(6, 2)`**

```
def recursivefunc(x,y):
  if x == 0:
    return (y)
  else:
    return recursivefunc(x-1, x+y)
```

a) 23
b) 22
c) 19
d) 21

6-7. What will the following return? → **`countMe(5)`**

```
def countMe(n):
  if n <= 0:
    return 1
  else:
    return n * countMe(n-1)
```

a) 120
b) 121
c) 720
d) 721

6-8. What will the following return? → **`fun(6)`**

```
def fun(n):
  if(n>10):
    return n-1
  return fun(fun(n+3))
```

a) 13
b) 11
c) 10
d) Infinite recursion

6-9. To check if **n** is a prime number, what should be the default value of **i** for it to work correctly? (assume **i** < **n**)

```
def is_prime(n, i=?):
  if n==i:
    return True
  if n%i==0:
    return False
  return is_prime(n, i+1)
```

a) 0
b) 1
c) 2
d) n

6-10. What will the following return? → **`convert(6)`**

```
def convert(v):
  if v == 0:
    return 0
  else:
    return (v%2 + 10*convert(int(v//2)))
```

What will convert(6) return?

a) 010
b) 110
c) 011
d) 101

Week 7 – Stacks/Queues

7-1. You have an empty circular queue which can hold only 4 elements. What are the values of **head** and **tail** after executing this series of queue operations?

```
enqueue("a")
enqueue("b")
enqueue("c")
enqueue("d")
dequeue()
dequeue()
enqueue("e")
enqueue("f")
```

a) Head: 1, Tail: 3
b) Head: 2, Tail: 2
c) Head: 1, Tail: 2
d) Head: 2, Tail: 0

7-2. What will be in the Stack **s** and Queue **q** after executing the following operations?

```
s.push("a")
s.push("b")
q.enqueue(s.pop())
s.push("c")
q.enqueue("f")
s.push("d")
s.push("e")
q.enqueue(s.pop())
s.pop()
q.dequeue()
```

a) s: top ['a', 'c'] , q: head ['f', 'e']
b) s: top ['e', 'd'] , q: head ['f', 'b']
c) s: top ['c', 'a'] , q: head ['f', 'e']
d) s: top ['d', 'e'] , q: head ['b', 'c']

7-3. The function **decode** takes in a string (c).

```
def decode(c):
  s = Stack()
  q = Queue()
  for i in range(0, len(c)):
    current = c[i]
    if current.isnumeric():
      s.push(current)
    else:
      q.enqueue(current)
```

What will be in s and q just before the function returns?

```
decode("1234abcd")
```

a) s: top ['4', '3', '2', '1'] , q: head ['d', 'c', 'b', 'a']
b) s: top ['1', '2', '3', '4'] , q: head ['d', 'c', 'b', 'a']
c) s: top ['4', '3', '2', '1'] , q: head ['a', 'b', 'c', 'd']
d) s: top ['1', '2', '3', '4'] , q: head ['a', 'b', 'c', 'd']

7-4. You are trying to simulate a queue using 2 stacks (**s1** & **s2**). The size of **s1** is **n** and there are **m** elements in this stack. What is the time complexity of performing dequeue operation so that the newest element is always at the top of **s1**? You may only use the stack operations **push** and **pop**; both are O(1).

a) O(m)
b) O(n)
c) O(m * n)
d) Insufficient info to determine answer

7-5. Consider mystery:

```
def mystery(n):
  s = Stack()
  for ch in n:
    if ch == "*":
      s.push(ch)
    elif ch == "&":
      if s.count() >0:
        s.pop()
      else:
        return False
  return s.count() == 0
```

Which of the following function calls will return **True**?

a) mystery ("a*&b*c&")
b) mystery ("*abc&&")
c) mystery ("*a&b&c*")
d) mystery ("&a&b*c*")

7-6. What is the state of the stack after these operations?

```
s=Stack()
s.push(33)
s.push(32)
s.peek()
s.push(23)
s.pop()
s.push(22)
s.push(38)
s.pop()
s.pop()
s.pop()
s.push(69)
s.display()
```

a) [Bottom] 69, 33 [Top]
b) [Top] 69, 22, 33 [Bottom]
c) [Bottom] 69, 22, 33 [Top]
d) [Top] 69, 33 [Bottom]

7-7. Consider the following **push** and **pop** functions of a stack:

```
def push(s, items):
  for item in items:
    s.insert(0, item)

def pop(s):
  if len(s) > 0:
    item = s[0]
    del s[0]
    return item
```

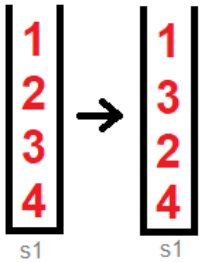
What is at the top of the stack (s) after these operations?

```
a1 = ['apple', 'pear', 'orange']
a2 = ['watermelon', 'mango']
s = []

push(s, a2)
pop(s)
push(s, a2)
push(s, a1)
pop(s)
```

a) orange
b) apple
c) pear
d) mango

7-8. You are given a stack **s1** containing 4 elements: (top)1,2,3,4, and an empty queue **q1**. What is the minimum number of operations required to rearrange the elements in s1 to 1,3,2,4 as shown? You are only allowed to use one temporary variable.



There are 4 possible operations:

- stack.pop()
- stack.push()

- queue.enqueue()
- queue.dequeue()

a) 12
b) 16
c) 20
d) 24

7-9. A series of pushes and pops are executed on an empty stack: a letter means push and asterisk (*) means pop. What will be popped out?

A B C * * D E F G O P Q R S * * T U * * * *

a) U T Q P S F B C
b) C B F S P Q T U
c) C B S R U T Q P
d) B C R S T U P Q

7-10. What is the worst-case time complexity to enqueue an element in a priority queue implemented as a "sorted list"?

a) O(n log n)
b) O(log n)
c) O(n)
d) $O(n^2)$