Name: Kokil Dhakal
Source: https://www.educative.io/blog/object-oriented-programming
Extension: None
Collaborator: None

1.
```
def binary_search(arr, low, high, x):  # Check base case
   if low > high:  ---------- O (1)
      return None
   else:
      mid = (high + low) // 2 ---------- O (1)
      element = arr[mid]
      if element == x:
         return mid
      elif element > x:
         return binary_search(arr, low, mid - 1, x)  ------- O(LogN)

      else:
         return binary_search(arr, mid + 1, high, x)  -----O(LogN)
```

overall Big O notation for this function in worst case i.e big input will be-
O (1)+ O(1)+ O(Log(n))+ O(Log(n))= O(logN)
In Binary Search, since we devide given list to two half each time until we find the intended item, for very large input value, the Big O notation will be O(logN)

2.
```
def selection_sort(arr):
   for i in range(len(arr)): ----------O(N)
      smallest_index = i
      smallest_value = arr[i]
      # Find smallest element
      for j in range (i, len(arr)): --------O(N)
         candidate = arr[j]
         if candidate < smallest_value:---O(1)
            smallest_index = j
            smallest_value = candidate
         # Swap front with smallest value
         temp = arr[i]
         arr[i] = smallest_value
         arr[smallest_index] = temp
   return arr
```

overall O(N)* O(N)+ O(1)= O(N^2)

in Selection sort, we assume first element as smallest element and compare with next element to it, it is less than the next element position will not be changed otherwise we swap their position. We continue for whole array until we get sorted assay. In this sort we have used nested loop which BIG O notation for large input case is O(N^2) which will be the dominant case and we ignore other that has lesser effect in performance than this.

3.

```
def insertion_sort(arr):
   for i in range(len(arr)): ----------O(N)
      current_element = arr[i]
      # Find correct place in list
      j = 0
      while j < i and arr[j] <= current_element: ----------O(N)
         j += 1
         # Move all larger elements over 1
      for x in range (i, j - 1, -1): ----------O(N)
         arr[x] = arr[x - 1]
      # Insert current element
         arr[j] = current_element
```

over all O(N)* (O(N)+ O(N))= O(N)*2 O(N)= 2O(N^2)= O(N^2)

1. $O(2*N + Log(N) + N^3 + 10)$ dominant term is N^3

   For high value of input, the order of dominant term will be:

   N^3>C*N>Log(N)>C where C= constant

   During simplification we take a most dominant term which is N^3

   That is why overall Big O notation will be $O(N^3)$

2.

In Big O notation, input of Constant value will have constant growth which is O(1)

O (1000) =O(C) = O(1)

3. $O(10+20*N+10*N*Log(N)+10*N^2)$

O(C+C*N+C*N*Log(N)+C*N^2)

In case of large input, we discard comparatively small constant value

Order of dominant term =N^2>N LogN>N>C

That is why simplification of this expression for Big O will be N^2

4. $O(N*Log(N)+20*N+N*Log(N)^2)$

For large value of input, we will discard constant

= O (N ∗Log(N)+N+N*Log(N)^2)

Order of dominant term is

N ∗Log(N)^2> N ∗Log(N))>C*N>N

So, simplify expression is N ∗Log(N)^2

5. $O(2 * N^2 * Log(N))$

For this, as we disregard constant for large value of input. Big O notation will be

O(N^2Log(N))

Problem 3

Four tenants of OOP are:
1.abstraction
2.Encapsulation
3.Inheritance
4.Polymorphism

Abstraction in programming is a concept to deal with the complex part of the program. This is basically hides unnecessary information and shows only essential attributes.

Encapsulation is concept of bundling methods and attributes within a single unit. Creating class is encapsulation which bundles all method, instance variables into a single unit.

Inheritance is concept of inheriting behavior and attributes of one class to another from parent to child class. This concept helps to prevent using repeating attributes for similar classes.

Polymorphism is the concept of having many forms. A method name for an instances of a class can be used for another instance for that class.