Lab-V

Hasan Amin
(374866)

CS-250
Data Structures and Algorithms

School of Natural Sciences

Task 1(a,b) and 2:

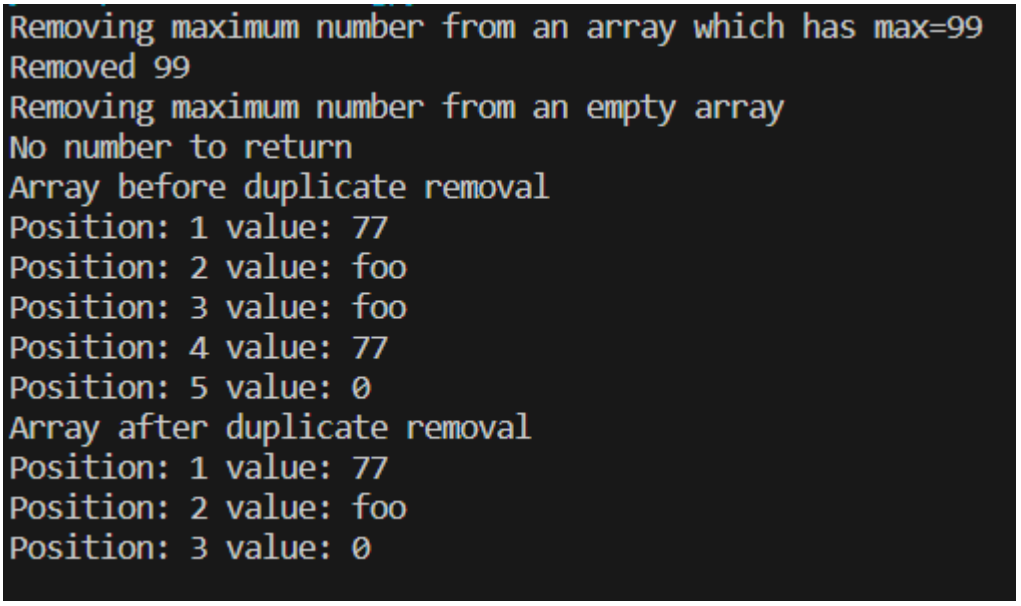
Code

```
integerArray = [1, 1, 2, 3, 5] # A list of 5 integers
charArray = ['a' for j in range(1000)] # 1,000 letter 'a' characters
booleArray = [False] * 32768 # 32,768 binary False values

maxSize = 10000
myArray = [None] * maxSize
myArraySize = 0

# Implement an Array data structure as a simplified type of list.

class Array:
   def __init__(self, initialSize):     # Constructor
      self.__a = [None] * initialSize  # The array stored as a list
      self.__nItems = 0                # No items in array initially
   def __len__(self):                  # Special def for len() func
      return self.__nItems             # Return number of items

   def isEmpty(self):
      return int(self.__nItems) == 0  #checks if length == 0

   def isFull(self):
      return int(self.__nItems) == len(self.__a)   #Length == maxSize

   def get(self, n):                   # Return the value at index n
      if 0 <= n and n < self.__nItems: # Check if n is in bounds, and
         return self.__a[n]            # only return item if in
bounds

   def insertAtEnd(self, item):              # Insert item at end
      if not self.isFull():
        self.__a[self.__nItems] = item   # Item goes at current end
        self.__nItems += 1               # Increment number of items
      else:
         print("List is Full. Insertion not possible")

   def insertAtPosition(self, position, value):           # Set the
```

```
value at index n
    if not self.isFull():
         if position >=1 and position <= self.__nItems: # Check if n
is in bounds, and
            if position == self.__nItems+1:
               self.insertAtEnd(value)
            else:
                self.MakeRoom(position)
                self.__a[position-1] = value           # only set
item if in bounds
                self.__nItems += 1
    else:
       print("List is Full. Insertion not possible")


   def MakeRoom(self, position):
       i=self.__nItems
       while i >= position:
           self.__a[i] =self.__a[i-1]
           i -=1

   def find(self, item):               # Find index for item
      for j in range(self.__nItems):   # Among current items
         if self.__a[j] == item:       # If found,
            return j                   # then return index to item
      return -1                        # Not found -> return -1

   def search(self, item):             # Search for item
      return self.get(self.find(item)) # and return item if found

   def delete(self, item):             # Delete first occurrence
      for j in range(self.__nItems):   # of an item
         if self.__a[j] == item:       # Found item
            self.__nItems -= 1         # One fewer at end
            for k in range(j, self.__nItems):  # Move items from
               self.__a[k] = self.__a[k+1]     # right over 1
            return True                # Return success flag

      return False     # Made it here, so couldn't find the item

   def traverse(self, function=print): # Traverse all items
      if not self.isEmpty():
```

```
        for j in range(self.__nItems):    # and apply a function
            function(f"Position: {j+1} value: {self.__a[j]}")
    # Task 1a & 1b
    def deleteMaxNum(self):
        curr_max=None
        for i in self.__a:
            if isinstance(i,(int,float)):
                if curr_max==None:
                    curr_max=i
                elif i>curr_max:
                    curr_max=i
        if curr_max is not None:
            self.delete(curr_max)
            return f"Removed {curr_max}"
        else:
            return f"No number to return"
    def removeDupes(self):
        new=[]
        for i in range(len(self.__a)):

            if self.__a[i] in new:
                pass
            else:
                new.append(self.__a[i])
        self.__a=new
```

Output


Removing maximum number from an array which has max=99
Removed 99
Removing maximum number from an empty array
No number to return
Array before duplicate removal
Position: 1 value: 77
Position: 2 value: foo
Position: 3 value: foo
Position: 4 value: 77
Position: 5 value: 0
Array after duplicate removal
Position: 1 value: 77
Position: 2 value: foo
Position: 3 value: 0

Task 3:

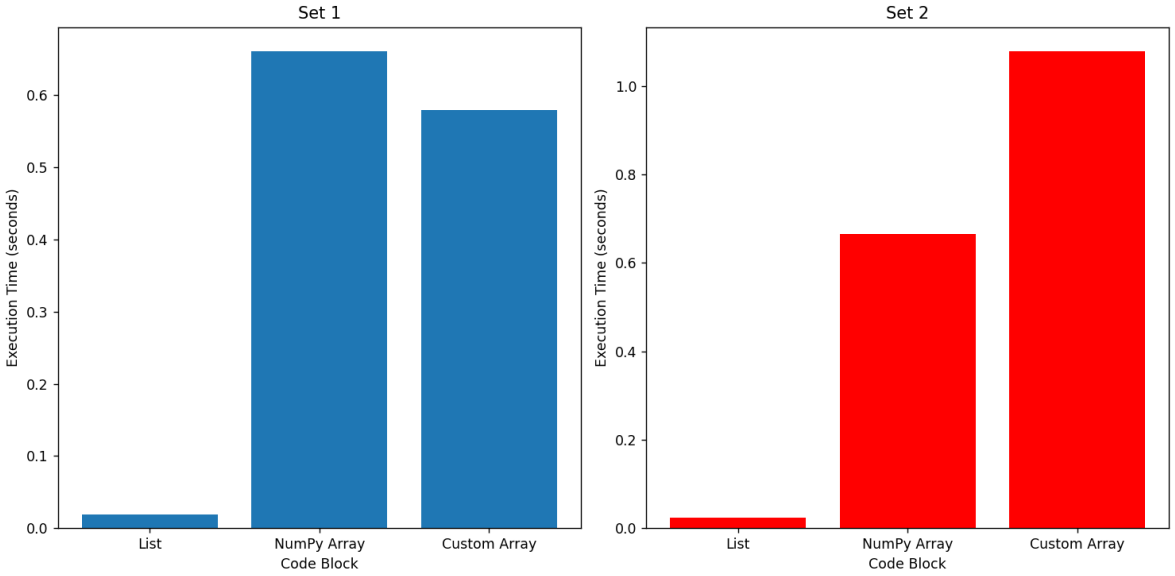
Code

```
import timeit
import numpy as np
from Array import Array
import matplotlib.pyplot as plt

# Define the code blocks to be timed
code_list = """
mylist=[2,2,4,3,5,6,4,5,8]
mylist.remove(max(mylist))
mylist=list(set(mylist))
"""

code_numpy = """
numpyArr=np.array([2,2,4,3,5,6,4,5,8])
indices=np.where(numpyArr==numpyArr.max())
newarr=np.delete(numpyArr,indices)
newarr=np.unique(newarr)
"""

code_custom_array = """
mylist=[2,2,4,3,5,6,4,5,8]
myarr=Array(len(mylist))
for i in mylist:
    myarr.insertAtEnd(i)
myarr.deleteMaxNum()
myarr.removeDupes()
"""

code_list2 = """
mylist=[2,2,4,3,5,6,4,5,8,20,30,76,45,66,66,21,34]
mylist.remove(max(mylist))
mylist=list(set(mylist))
"""

code_numpy2 = """
numpyArr=np.array([2,2,4,3,5,6,4,5,8,20,30,76,45,66,66,21,34])
indices=np.where(numpyArr==numpyArr.max())
```

```
newarr=np.delete(numpyArr,indices)
newarr=np.unique(newarr)
"""

code_custom_array2 = """
mylist=[2,2,4,3,5,6,4,5,8,20,30,76,45,66,66,21,34]
myarr=Array(len(mylist))
for i in mylist:
    myarr.insertAtEnd(i)
myarr.deleteMaxNum()
myarr.removeDupes()
"""

# Set up the number of repetitions
repeats = 10000

# Time each block for the first set of data
time_list = timeit.timeit(stmt=code_list, number=repeats,
globals=globals())
time_numpy = timeit.timeit(stmt=code_numpy, number=repeats,
globals=globals())
time_custom_array = timeit.timeit(stmt=code_custom_array,
number=repeats, globals=globals())

# Time each block for the second set of data
time_list2 = timeit.timeit(stmt=code_list2, number=repeats,
globals=globals())
time_numpy2 = timeit.timeit(stmt=code_numpy2, number=repeats,
globals=globals())
time_custom_array2 = timeit.timeit(stmt=code_custom_array2,
number=repeats, globals=globals())

# Create subplots to visualize the timing results
fig, axes = plt.subplots(1, 2, figsize=(12, 6))

blocks = ['List', 'NumPy Array', 'Custom Array']
times = [time_list, time_numpy, time_custom_array]

axes[0].bar(blocks, times)
axes[0].set_xlabel('Code Block')
axes[0].set_ylabel('Execution Time (seconds)')
axes[0].set_title('Set 1')
```

```
blocks2 = ['List', 'NumPy Array', 'Custom Array']
times2 = [time_list2, time_numpy2, time_custom_array2]

axes[1].bar(blocks2, times2,color=['red'])
axes[1].set_xlabel('Code Block')
axes[1].set_ylabel('Execution Time (seconds)')
axes[1].set_title('Set 2')

plt.tight_layout()
plt.show()
```

Output



Task 4 and 5:

Code

```
#Task4
import timeit
print(timeit.timeit('sum([x for x in range(10000000)])',number=1))
print(timeit.timeit(stmt='np.arange(100000000).sum()',setup='import
numpy as np',number=1))

#Task 5
import numpy as np

def median(list):
    array = np.sort(np.array(list))
    med = {}
    for i in range(array.shape[0]):
        if len(array[i]) % 2 == 0:
            med[f"row {i}"] = (array[i][len(array[i]) // 2] +
array[i][len(array[i]) // 2 - 1]) / 2
        else:
            med[f"row {i}"] = array[i][len(array[i]) // 2]
    return med

def mode(arr):
    arr_flat = arr.flatten()  # Flatten the array to a 1D array
    unique_values = np.unique(arr_flat)
    modes = []
    max_count = 0

    for value in unique_values:
        count = np.count_nonzero(arr_flat == value)
        if count > max_count:
            max_count = count
            modes = [value]
        elif count == max_count:
            modes.append(value)

    if len(modes) == len(arr_flat):
        return None
```

```
    else:
        return modes


print(median(np.array([[20,23,24],[1,7,4],[455,9,86]])))
print(mode(np.array([[20,23,24],[1,7,4],[455,24,86]])))
```

Output