NumPy Sorting Searching Exercises and NumPy Advanced Indexing: Exercises

1. Sort Array Along Different Axes

Write a NumPy program to sort a given array of shape 2 along the first axis, last axis and on flattened array.

```
import numpy as np

a = np.array([[10,]])
```

1. Structured Array Sorted by Height

Write a NumPy program to create a structured array from given student name, height, class and their data types. Now sort the array on height.

```
# Importing the NumPy library
import numpy as np

# Defining the data types for the structured array
data_type = [('name', 'S15'), ('class', int), ('height', float)]

# Defining the details of students as a list of tuples
students_details = [('James', 5, 48.5), ('Nail', 6, 52.5), ('Paul', 5,
42.10), ('Pit', 5, 40.11)]

# Creating a structured array 'students' using the defined data type
and provided details
students = np.array(students_details, dtype=data_type)

# Displaying the original structured array
print("Original array:")
print(students)

# Sorting the structured array by 'height' field
print("Sort by height")
print(np.sort(students, order='height'))
```

```
Original array:
[(b'James', 5, 48.5 ) (b'Nail', 6, 52.5 ) (b'Paul', 5, 42.1 )
 (b'Pit', 5, 40.11)]
Sort by height
[(b'Pit', 5, 40.11) (b'Paul', 5, 42.1 ) (b'James', 5, 48.5 )
 (b'Nail', 6, 52.5 )]
```

1. Structured Array Sorted by Class Then Height

Write a NumPy program to create a structured array from given student name, height, class and their data types. Now sort by class, then height if class are equal.

```
# Importing the NumPy library
import numpy as np

# Defining the data types for the structured array
data_type = [('name', 'S15'), ('class', int), ('height', float)]

# Defining the details of students as a list of tuples
students_details = [('James', 5, 48.5), ('Nail', 6, 52.5), ('Paul', 5,
42.10), ('Pit', 5, 40.11)]

# Creating a structured array 'students' using the defined data type
and provided details
students = np.array(students_details, dtype=data_type)

# Displaying the original structured array
print("Original array:")
print(students)

# Sorting the structured array by 'class' and 'height' fields
print("Sort by class, then height if class are equal:")
print(np.sort(students, order=['class', 'height']))
```

```
Original array:
[(b'James', 5, 48.5 ) (b'Nail', 6, 52.5 ) (b'Paul', 5, 42.1 )
 (b'Pit', 5, 40.11)]
Sort by class, then height if class are equal:
[(b'Pit', 5, 40.11) (b'Paul', 5, 42.1 ) (b'James', 5, 48.5 )
 (b'Nail', 6, 52.5 )]
```

1. Sort Student IDs by Increasing Height

Write a NumPy program to sort the student id with increasing height of the students from given students id and height. Print the integer indices that describes the sort order by multiple columns and the sorted data.

```
# Importing the NumPy library
import numpy as np

# Creating NumPy arrays for student IDs and their heights
student_id = np.array([1023, 5202, 6230, 1671, 1682, 5241, 4532])
student_height = np.array([40., 42., 45., 41., 38., 40., 42.0])

# Sorting the indices based on 'student_id' and then 'student_height'
indices = np.lexsort((student_id, student_height))

# Displaying the sorted indices
print("Sorted indices:")
print(indices)

# Displaying the sorted data based on the obtained indices
```

```
print("Sorted data:")
for n in indices:
    print(student_id[n], student_height[n])
```

```
Sorted indices:
[4 0 5 3 6 1 2]
Sorted data:
1682 38.0
1023 40.0
5241 40.0
1671 41.0
4532 42.0
5202 42.0
6230 45.0
```

1. Indices of Sorted Elements

Write a NumPy program to get the indices of the sorted elements of a given array.

```
# Importing the NumPy library
import numpy as np

# Creating a NumPy array for student IDs
student_id = np.array([1023, 5202, 6230, 1671, 1682, 5241, 4532])

# Displaying the original array
print("Original array:")
print(student_id)

# Sorting the indices of the array's elements in ascending order
i = np.argsort(student_id)

# Displaying the indices of the sorted elements
print("Indices of the sorted elements of a given array:")
print(i)
```

```
Original array:
[1023 5202 6230 1671 1682 5241 4532]
Indices of the sorted elements of a given array:
[0 3 4 6 1 5 2]
```

1. Sort Complex Array by Real then Imaginary Part

Write a NumPy program to sort a given complex array using the real part first, then the imaginary part.

Note: "busday" default of Monday through Friday being valid days.

```
# Importing the NumPy library
import numpy as np
```

```
# Creating a list of complex numbers
complex_num = [1 + 2j, 3 - 1j, 3 - 2j, 4 - 3j, 3 + 5j]

# Displaying the original array of complex numbers
print("Original array:")
print(complex_num)

# Sorting the given complex array using the real part first, then the
imaginary part
print("\nSorted a given complex array using the real part first, then
the imaginary part.")
print(np.sort_complex(complex_num))
```

```
Original array:
[(1+2j), (3-1j), (3-2j), (4-3j), (3+5j)]

Sorted a given complex array using the real part first, then the
imaginary part.
[1.+2.j 3.-2.j 3.-1.j 3.+5.j 4.-3.j]
```

1. Partition Array by Specified Position

Write a NumPy program to partition a given array in a specified position and move all the smaller elements values to the left of the partition, and the remaining values to the right, in arbitrary order (based on random choice).

Sample output: Original array: [70 50 20 30 -11 60 50 40] After partitioning on 4 the position: [-11 30 20 40 50 50 60 70]

```
# Importing the NumPy library
import numpy as np

# Creating an array of integers
nums = np.array([70, 50, 20, 30, -11, 60, 50, 40])

# Displaying the original array
print("Original array:")
print(nums)

# Partitioning the array at the 4th position
print("\nAfter partitioning on 4 the position:")
print(np.partition(nums, 4))
```

```
Original array:
[ 70  50  20  30 -11  60  50  40]

After partitioning on 4 the position:
[-11  30  20  40  50  50  60  70]
```

1. Partial Sorting from the Beginning of an Array

Write a NumPy program to sort the specified number of elements from beginning of a given array.

Sample output: Original array: [0.39536213 0.11779404 0.32612381 0.16327394 0.98837963 0.25510787 0.01398678 0.15188239 0.12057667 0.67278699] Sorted first 5 elements: [0.01398678 0.11779404 0.12057667 0.15188239 0.16327394 0.25510787 0.39536213 0.98837963 0.32612381 0.67278699]

```
# Importing the NumPy library
import numpy as np

# Creating an array of 10 random numbers
nums = np.random.rand(10)

# Displaying the original array
print("Original array:")
print(nums)

# Sorting the first 5 elements using argpartition
print("\nSorted first 5 elements:")
print(nums[np.argpartition(nums, range(5))])
```

```
Original array:
[0.723881   0.72532594 0.96228781 0.74461304 0.58295294 0.25854051
 0.36398703 0.89243774 0.47435653 0.25449578]

Sorted first 5 elements:
[0.25449578 0.25854051 0.36398703 0.47435653 0.58295294 0.72532594
 0.96228781 0.89243774 0.74461304 0.723881  ]
```

1. Sort Array by Nth Column

Write a NumPy program to sort an given array by the nth column.

Original array: [[1 5 0] [3 2 5] [8 7 6]] Sort the said array by the nth column: [[3 2 5] [1 5 0] [8 7 6]]

```
# Importing the NumPy library
import numpy as np

# Displaying a message indicating the original array is going to be
printed
print("Original array:\n")

# Generating a 3x3 array of random integers from 0 to 10
nums = np.random.randint(0, 10, (3, 3))

# Displaying the original array
print(nums)

# Displaying a message indicating sorting the array by the nth column
print("\nSort the said array by the nth column: ")
```

```
# Sorting the array by the values in the nth column using argsort
print(nums[nums[:, 1].argsort()])
```

```
Original array:

[[6 2 5]
 [7 5 0]
 [5 1 4]]

Sort the said array by the nth column:
[[5 1 4]
 [6 2 5]
 [7 5 0]]
```

NumPy Advanced Indexing: Exercises

1. 2D Array Random Integers & Boolean Indexing

Write a NumPy program that creates a 2D NumPy array of random integers. Use boolean indexing to select all elements greater than a specified value.

```
import numpy as np

# Create a 2D NumPy array of random integers
array_2d = np.random.randint(0, 100, size=(5, 5))

# Specify the value for boolean indexing
threshold = 50

# Use boolean indexing to select elements greater than the threshold
selected_elements = array_2d[array_2d > threshold]

# Print the original array and the selected elements
print('Original 2D array:\n', array_2d)
print(f'\nElements greater than {threshold}:\n', selected_elements)
```

```
Original 2D array:
 [[62 83 49 81 83]
 [11 75 45 74 98]
 [62 56 73 71 26]
 [90 93 76 10 93]
 [67 88 75 58  8]]

Elements greater than 50:
 [62 83 81 83 75 74 98 62 56 73 71 90 93 76 93 67 88 75 58]
```

1. 1D Array & Integer Array Indexing

Write a NumPy program that creates a 1D NumPy array and uses integer array indexing to select a subset of elements based on an index array.

```python
import numpy as np

# Create a 1D NumPy array
array_1d = np.array([10, 20, 30, 40, 50, 60, 70, 80, 90, 100])

# Define an index array to select specific elements
index_array = np.array([0, 2, 4, 6, 8])

# Use integer array indexing to select elements
selected_elements = array_1d[index_array]

# Print the original array and the selected elements
print('Original 1D array:\n', array_1d)
print('Index array:\n', index_array)
print('Selected elements:\n', selected_elements)
```

```
Original 1D array:
 [ 10  20  30  40  50  60  70  80  90 100]
Index array:
 [0 2 4 6 8]
Selected elements:
 [10 30 50 70 90]
```

1. 3D Array & Fancy Indexing

Write a NumPy program that creates a 3D NumPy array and uses fancy indexing to select elements from specific rows and columns.

```python
import numpy as np

# Create a 3D NumPy array of shape (3, 4, 5)
array_3d = np.random.randint(0, 100, size=(3, 4, 5))

# Define the row and column indices to select specific elements
row_indices = np.array([0, 1, 2])
col_indices = np.array([1, 2, 3])

# Use fancy indexing to select elements from specific rows and columns
selected_elements = array_3d[row_indices[:, np.newaxis], col_indices]

# Print the original 3D array and the selected elements
print('Original 3D array:\n', array_3d)
print('Row indices:\n', row_indices)
print('Column indices:\n', col_indices)
print('Selected elements:\n', selected_elements)
```

```
Original 3D array:
 [[[37 41 42 70 84]
  [45 12 33 38 76]
  [60 82 57 52 72]
```

```
  [79 48 40 48 58]]

 [[74 23 41 45 36]
  [40 48 94 58 70]
  [14 77 87 98 47]
  [74  5 25 87 32]]

 [[13 59 63 45 12]
  [86  2 92 61 75]
  [82 16 60 39 13]
  [18 31 12 77 11]]]
Row indices:
 [0 1 2]
Column indices:
 [1 2 3]
Selected elements:
 [[[45 12 33 38 76]
  [60 82 57 52 72]
  [79 48 40 48 58]]

 [[40 48 94 58 70]
  [14 77 87 98 47]
  [74  5 25 87 32]]

 [[86  2 92 61 75]
  [82 16 60 39 13]
  [18 31 12 77 11]]]
```

1. 4D Array & Multi-dimensional Indexing

Write a NumPy program that creates a 4D NumPy array and uses multi-dimensional indexing to select a subarray.

```python
import numpy as np

# Create a 4D NumPy array of shape (3, 4, 5, 6) with random integers
array_4d = np.random.randint(0, 100, size=(3, 4, 5, 6))

# Use multi-dimensional indexing to select a subarray
# For example, select all elements from the first two blocks,
# first three rows, first four columns, and first five depth slices
subarray = array_4d[:2, :3, :4, :5]

# Print the shape of the original 4D array and the selected subarray
print('Original 4D array shape:', array_4d.shape)
print('Selected subarray shape:', subarray.shape)
print('Selected subarray:\n', subarray)
```

```
Original 4D array shape: (3, 4, 5, 6)
Selected subarray shape: (2, 3, 4, 5)
```

```
Selected subarray:
 [[[[ 3 13 53 97 92]
   [59 22 45 79 14]
   [65 52 66 49 74]
   [20 65 97 80 56]]

  [[51 14 57 64 17]
   [83 46 16 51 23]
   [75  9 21 91 13]
   [61  7 46  7 64]]

  [[27 77  2 91  3]
   [59 54 63  2 18]
   [11 36 24  9 73]
   [46 58 88 71 97]]]


 [[[49 62  7 71 83]
   [49 92 52 55 93]
   [ 8  5 76 25 81]
   [43 26 99  1 37]]

  [[86 54 32 89 80]
   [73 58 36 56 30]
   [38 55 48 18 16]
   [78 50 59 72 81]]

  [[ 4 49 80 70 55]
   [24 67 20 73 93]
   [67 41 76 84 69]
   [90 90 13  5 80]]]]
```

1. 2D Random Floats & Multiple Condition Boolean Indexing

Write a NumPy program that creates a 2D NumPy array of random floats and uses boolean indexing to select elements that satisfy multiple conditions (e.g., greater than 0.5 and less than 0.8).

```python
import numpy as np

# Create a 2D NumPy array of random floats between 0 and 1
array_2d = np.random.rand(5, 5)

# Define the conditions
condition1 = array_2d > 0.5
condition2 = array_2d < 0.8

# Use boolean indexing to select elements that satisfy both conditions
selected_elements = array_2d[condition1 & condition2]
```

```
# Print the original array and the selected elements
print('Original 2D array:\n', array_2d)
print('Elements greater than 0.5 and less than 0.8:\n',
selected_elements)
```

```
Original 2D array:
 [[0.24078157 0.74098245 0.66869924 0.64398649 0.99858288]
 [0.29739968 0.59422099 0.52273859 0.03214772 0.9380528 ]
 [0.4656626  0.88616843 0.71622137 0.04467153 0.73837477]
 [0.92998099 0.57791482 0.04601077 0.55182457 0.72159867]
 [0.28696581 0.94415966 0.47814615 0.61477509 0.07607168]]
Elements greater than 0.5 and less than 0.8:
 [0.74098245 0.66869924 0.64398649 0.59422099 0.52273859 0.71622137
 0.73837477 0.57791482 0.55182457 0.72159867 0.61477509]
```

1. 2D Array & Integer Indexing with Broadcasting

Write a NumPy program that creates a 2D NumPy array and uses integer indexing with broadcasting to select elements from specific rows and all columns.

```
import numpy as np

# Create a 2D NumPy array of shape (5, 5) with random integers
array_2d = np.random.randint(0, 100, size=(5, 5))

# Define the row indices to select specific rows
row_indices = np.array([1, 3])

# Use integer indexing with broadcasting to select elements from
specific rows and all columns
selected_elements = array_2d[row_indices[:, np.newaxis],
np.arange(array_2d.shape[1])]
# Print the original array and the selected elements
print('Original 2D array:\n', array_2d)
print('Selected rows:\n', row_indices)
print('Selected elements from specific rows and all columns:\n',
selected_elements)
```

```
Original 2D array:
 [[98 61 99 91 72]
 [76 40 28 94 21]
 [32 76 87 76 49]
 [23 97 85 23 57]
 [ 9 96 45 36 79]]
Selected rows:
 [1 3]
Selected elements from specific rows and all columns:
 [[76 40 28 94 21]
 [23 97 85 23 57]]
```

1. 3D Array & Boolean Indexing Along One Axis

Write a NumPy program that creates a 3D NumPy array and use boolean indexing to select elements along one axis based on conditions applied to another axis.

```
import numpy as np

# Create a 3D NumPy array of shape (3, 4, 5) with random integers
array_3d = np.random.randint(0, 100, size=(3, 4, 5))

# Define the condition to apply on the second axis (axis 1)
condition = array_3d[:, :, 0] > 50

# Use boolean indexing to select elements along the third axis (axis
2)
selected_elements = array_3d[condition]

# Print the original 3D array and the selected elements
print('Original 3D array:\n', array_3d)
print('Condition array (elements along second axis where first element
> 50):\n', condition)
print('Selected elements along third axis based on condition:\n',
selected_elements)
```

```
Original 3D array:
 [[[74 48 41 70 49]
  [85 47 79 26 58]
  [30 96 79  5 35]
  [78 38 24 25 74]]

 [[50 49 27 71 53]
  [ 8 70 28 59 71]
  [ 7 85 17 73 26]
  [93 55 63 24 62]]

 [[14 71 74  6 49]
  [86 92 99 12 67]
  [19 28 53 57  9]
  [89 81 76 66  3]]]
Condition array (elements along second axis where first element > 50):
 [[ True  True False  True]
 [False False False  True]
 [False  True False  True]]
Selected elements along third axis based on condition:
 [[74 48 41 70 49]
 [85 47 79 26 58]
 [78 38 24 25 74]
 [93 55 63 24 62]
 [86 92 99 12 67]
 [89 81 76 66  3]]
```

1. 2D Array & Tuple of Arrays Indexing

Write a NumPy program that creates a 2D NumPy array and uses a tuple of arrays to index and select a specific set of elements.

```
import numpy as np

# Create a 3D NumPy array of shape (3, 4, 5) with random integers
array_3d = np.random.randint(0, 100, size=(3, 4, 5))

# Define the condition to apply on the second axis (axis 1)
condition = array_3d[:, :, 0] > 50

# Use boolean indexing to select elements along the third axis (axis
2)
selected_elements = array_3d[condition]

# Print the original 3D array and the selected elements
print('Original 3D array:\n', array_3d)
print('Condition array (elements along second axis where first element
> 50):\n', condition)
print('Selected elements along third axis based on condition:\n',
selected_elements)
```

```
Original 3D array:
 [[[22 30 11 22 99]
  [68 88 25 32 70]
  [ 4  7 72  3 36]
  [49 15 30 27 44]]

 [[60 57 19 62 69]
  [14 15 20 53 20]
  [32 66 33 77 32]
  [12 42 18 37 25]]

 [[84 12 52 17 44]
  [ 8  6  1 80  9]
  [99 26 15 35 34]
  [16 13 38 62 82]]]
Condition array (elements along second axis where first element > 50):
 [[False  True False False]
 [ True False False False]
 [ True False  True False]]
Selected elements along third axis based on condition:
 [[68 88 25 32 70]
 [60 57 19 62 69]
 [84 12 52 17 44]
 [99 26 15 35 34]]
```

1. Cross-Indexing with np.ix_

Write a NumPy program that creates two 1D NumPy arrays and uses np.ix_ to perform cross-indexing on a 2D array.

```
import numpy as np

# Create a 2D NumPy array of shape (5, 5) with random integers
array_2d = np.random.randint(0, 100, size=(5, 5))

# Create two 1D NumPy arrays for cross-indexing
row_indices = np.array([1, 3])
col_indices = np.array([0, 2, 4])

# Use np.ix_ to create an open mesh from the row and column indices
index_grid = np.ix_(row_indices, col_indices)

# Use the index grid to select elements from the 2D array
selected_elements = array_2d[index_grid]

# Print the original array and the selected elements
print('Original 2D array:\n', array_2d)
print('Row indices:\n', row_indices)
print('Column indices:\n', col_indices)
print('Selected elements using np.ix_:\n', selected_elements)
```

```
Original 2D array:
 [[61 40 81 22 58]
 [ 8 51 21 33 30]
 [36 27 24  9 90]
 [36 32 20 45 27]
 [ 6 39 92 68 30]]
Row indices:
 [1 3]
Column indices:
 [0 2 4]
Selected elements using np.ix_:
 [[ 8 21 30]
 [36 20 27]]
```

1. 2D Array & Boolean Indexing Replacement

Write a NumPy program that creates a 2D NumPy array and uses boolean indexing to replace all elements that meet a certain condition with a specified value.

```
import numpy as np

# Create a 2D NumPy array of shape (5, 5) with random integers
array_2d = np.random.randint(0, 100, size=(5, 5))

# Define the condition to replace elements greater than 50
condition = array_2d > 50
```

```python
# Define the value to replace the elements that meet the condition
replacement_value = -1

# Use boolean indexing to replace elements that meet the condition
array_2d[condition] = replacement_value

# Print the modified array
print('Modified 2D array:\n', array_2d)
```

```
Modified 2D array:
 [[-1 32 18 -1 -1]
 [-1  9 -1 -1 -1]
 [-1 -1 32 26 33]
 [-1 15 44 -1 13]
 [33  6 -1 -1 -1]]
```