

HOMWORK 5-6

DATA STRUCTURES *

10-607 COMPUTATIONAL FOUNDATIONS FOR MACHINE LEARNING

START HERE: Instructions

- **Collaboration Policy:** Please read the collaboration policy in the syllabus.
- **Late Submission Policy:** See the late submission policy in the syllabus.
- **Submitting your work:** You will use Gradescope to submit answers to all questions.
 - **Written:** For written problems such as short answer, multiple choice, derivations, proofs, or plots, please use the provided template. Submissions can be handwritten onto the template, but should be labeled and clearly legible. If your writing is not legible, you will not be awarded marks. Alternatively, submissions can be written in \LaTeX . Each derivation/proof should be completed in the boxes provided. To receive full credit, you are responsible for ensuring that your submission contains exactly the same number of pages and the same alignment as our PDF template.
 - **Latex Template:** <https://www.overleaf.com/read/hfgqsqhztcbj>

Question	Points
Vector Programming	18
Tree Programming	8
Graphs	11
Divide and Conquer	14
Total:	51

Instructions for Specific Problem Types

For “Select One” questions, please fill in the appropriate bubble completely:

Select One: Who taught this course?

- ☒ Matt Gormley
- ☐ Marie Curie
- ☐ Noam Chomsky

If you need to change your answer, you may cross out the previous answer and bubble in the new answer:

Select One: Who taught this course?

- ☒ Henry Chai
- ☐ Marie Curie
- ☒ Noam Chomsky

For “Select all that apply” questions, please fill in all appropriate squares completely:

Select all that apply: Which are scientists?

- ☒ Stephen Hawking
- ☒ Albert Einstein
- ☒ Isaac Newton
- ☐ I don't know

Again, if you need to change your answer, you may cross out the previous answer(s) and bubble in the new answer(s):

Select all that apply: Which are scientists?

- ☒ Stephen Hawking
- ☒ Albert Einstein
- ☒ Isaac Newton
- ☐ I don't know

For questions where you must fill in a blank, please make sure your final answer is fully included in the given space. You may cross out answers or parts of answers, but the final answer must still be within the given space.

Fill in the blank: What is the course number?

10-606

10-60~~6~~7

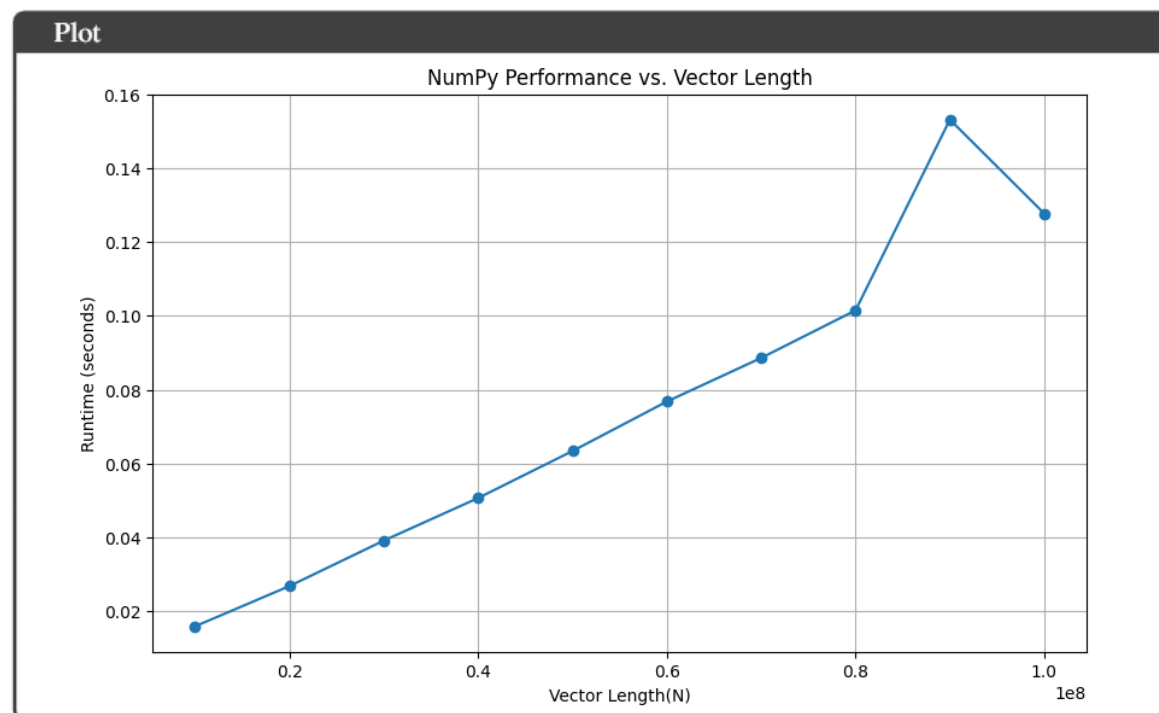
1 Vector Programming (18 points)

In this question we'll be using the vector dot product to analyze the difference in performance between implementations using for loops and those using NumPy. We'll also look at sparse representations of vectors and their effect on performance. The following Jupyter notebook contains the instructions for implementing the three dot products functions and running three performance experiments:

https://drive.google.com/file/d/1Bxi6cL0aqpcLUKud_o2LYfWgbCiOV-9q

Note: To make it more likely that we are all running our code in similar environments, if possible, please use Google Colab to run the performance experiments in the above file. After running the performance experiments, provide your plots and answer the questions below.

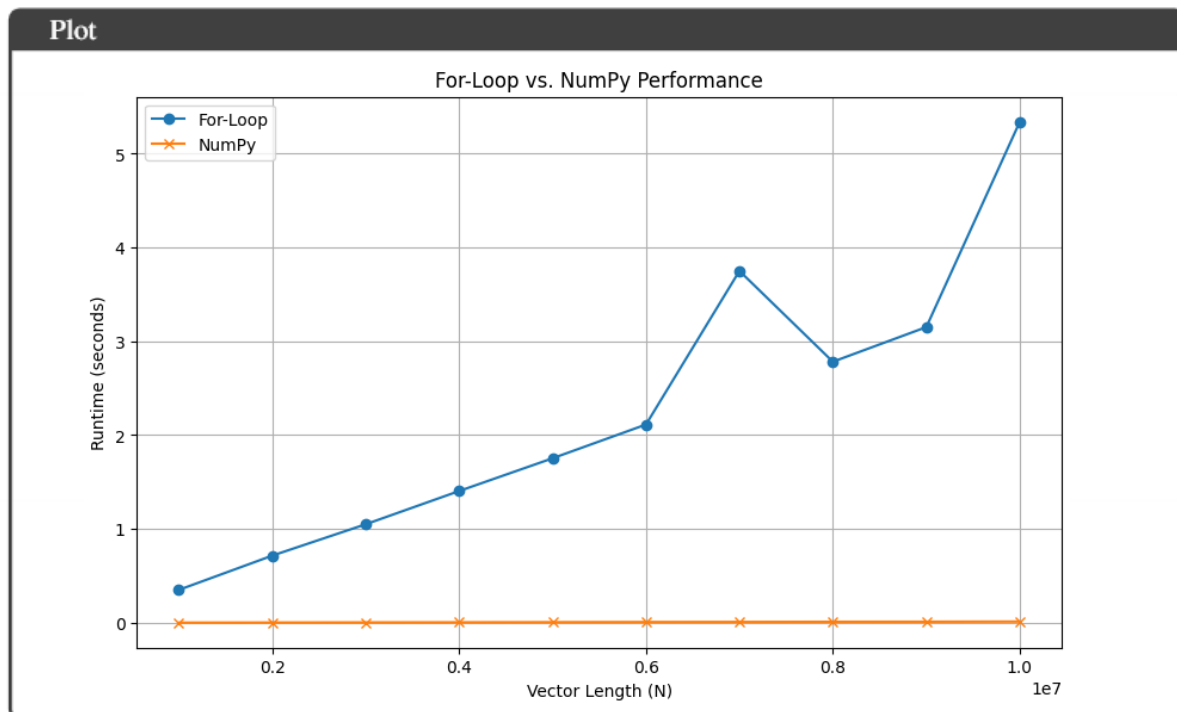
- (3 points) Experiment 1: Make a plot that compares the empirical runtime of your NumPy implementation to the length of the vectors. Follow the detailed instructions for Experiment 1 in the provided code and submit your plot below. Use the plot to provide an *empirical* estimate of the runtime complexity.



- (1 point) What is the (simplest and tightest) time complexity that you would expect for a dot product implementation with respect to the vector length N ?
☐ $O(1)$ ☐ $O(\log N)$ ☒ $O(N)$ ☐ $O(N \log N)$ ☐ $O(N^2)$
- (2 points) Do the empirical results in your plot seem to support this complexity? Explain your reasoning in one sentence. (Note that we, of course, can't measure the actual complexity from just these empirical results with a finite N).

Yes, the result supports $O(N)$. The runtime increases linearly as N grows, shown by the almost linear graph.

4. (3 points) Experiment 2: Plot the runtime of the for-loop and NumPy implementations as the length of the vectors changes. Follow the detailed instructions for Experiment 2 in the provided code and submit your plot below.

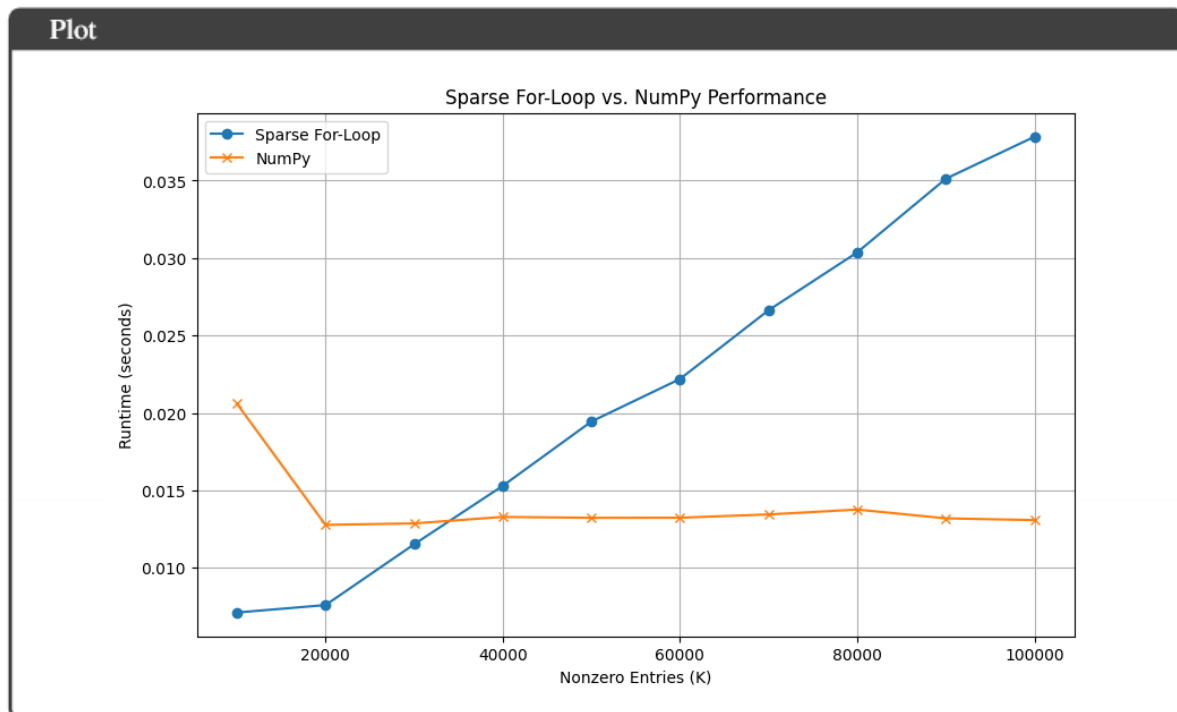


5. (2 points) Do the empirical results for Experiments 1 and 2 support the claim that for-loop and NumPy implementations are in the same complexity class? Explain your reasoning in one or two sentences.

(Again, we, of course, can't measure the actual complexity from just these empirical results with a finite N).

Yes, both implementations are in the same complexity class, $O(N)$ as they scale linearly with vector length N . NumPy is signif. faster due to optimized, low level parallelization, as seen by the smaller plot runtime.

6. (3 points) Experiment 3: Plot the runtime of the NumPy and sparse-for-loop implementations as the proportion of nonzero entries changes. Follow the detailed instructions for Experiment 3 in the provided code and submit your plot below.



7. (2 points) What is the time complexity for the sparse-for-loop dot product implementation with respect to the vector length N and the number of nonzeros K ? Give your answer in big-O notation with the simplest and tightest bound. Note: this is not an empirical estimate, you should analyze your code.

$O(K)$

8. (2 points) Based on your Experiment 3 plot, give an estimate for the density of nonzeros, K/N , below which it may be more efficient to use the sparse-for-loop implementation. Give your answer as a decimal value between 0.0 and 1.0 (not as a fraction).

Intersection $\sim 33,333$ $K =$
 $K/N \approx 0.0033$ N fixed at $10,000,000 (10^7)$
 < 0.0033

Programming Submission

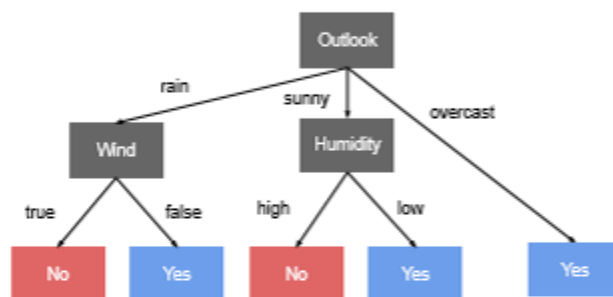
Please submit the file `vector.ipynb` (in addition to `tree.py`) directly to Gradescope under the HW5 (programming) assignment. Note, there are no autograder tests associated with this submitted file on Gradescope. We will manually grade the code for your dot product implementations and your experiments.

2 Tree Programming (8 points)

All your code for this question should be in a Python file named `tree.py`.

First implement the following three data structures.

- In Python, implement a `Stack` data structure with two functions (`push` and `pop`).
- Implement a `Queue` data structure with two functions (`enqueue` and `dequeue`).
- Implement a `DecisionTree` data structure that says whether or not you should go golfing. One way to do this is to use a Python dictionary where the condition at a node can be used as a string key to traverse the children of the node. A more object-oriented way is to create a `TreeNode` class that stores the label of the node and a list of children, which themselves are also `TreeNode`s. Using your tree implementation, create the following decision tree in Python code:



1. (4 points) Use depth first search of the tree to read out each possible rule from the root of the tree to each leaf in the tree. **Make sure to use either the stack or queue defined in part 1 of this question as part of your implementation.** Put all of your code in the function `def golf_tree_dfs()`. The output of your code should resemble the following:

```

If Outlook=overcast, Golf=yes
If Outlook=rain and Wind=false, Golf=yes
If Outlook=rain and Wind=true, Golf=no
If Outlook=sunny and Humidity=high, Golf=no
If Outlook=sunny and Humidity=low, Golf=yes
  
```

Include a legible screenshot of your console output here.

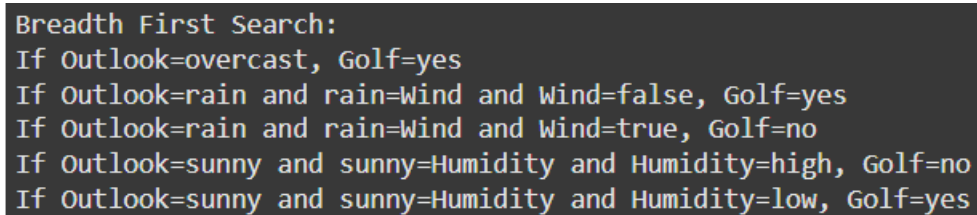
```

Depth First Search:
If Outlook=overcast, Golf=yes
If Outlook=rain and rain=Wind and Wind=false, Golf=yes
If Outlook=rain and rain=Wind and Wind=true, Golf=no
If Outlook=sunny and sunny=Humidity and Humidity=high, Golf=no
If Outlook=sunny and sunny=Humidity and Humidity=low, Golf=yes
  
```

2. (4 points) Use breadth first search of the tree to read out each possible rule from the root of the tree to each leaf in the tree. **Make sure to use either the stack or queue defined in part 1 of this question as part of your implementation.** Put all of your code in the function `def golf_tree_bfs()`. The output of your code should resemble the following:

```
If Outlook=overcast, Golf=yes
If Outlook=rain and Wind=false, Golf=yes
If Outlook=rain and Wind=true, Golf=no
If Outlook=sunny and Humidity=high, Golf=no
If Outlook=sunny and Humidity=low, Golf=yes
```

Include a legible screenshot of your console output here.



```
Breadth First Search:
If Outlook=overcast, Golf=yes
If Outlook=rain and rain=Wind and Wind=false, Golf=yes
If Outlook=rain and rain=Wind and Wind=true, Golf=no
If Outlook=sunny and sunny=Humidity and Humidity=high, Golf=no
If Outlook=sunny and sunny=Humidity and Humidity=low, Golf=yes
```

Instructions for submitting code

Please submit the file `tree.py` (in addition to `vector.ipynb`) directly to Gradescope under the HW5 (programming) assignment. Note, there are no autograder tests associated with this submitted file on Gradescope. We will manually grade the code for your tree and search implementations.

3 Graphs (11 points)

Consider the following pseudocode for a `Puzzle` class. The name of the class is chosen to indicate that in the following questions you will need to *puzzle* through what this class actually does. The `__init__(G, s)` method takes as arguments a directed graph G and a vertex s . If v is a vertex in G , then `G.get_adjacent_nodes(v)` returns an iterable collection of all vertices that are adjacent to v in G .

```
class Puzzle:
    def __init__(self, G, s):
        self.tagged = {}
        puzzle(G, s)

    def puzzle(G, v):
        self.tagged[v] = True
        for w in G.get_adjacent_nodes(v):
            if w not in self.tagged:
                puzzle(G, w)

    def tagged(v):
        return self.tagged[v]
```

- (5 points) Suppose we create a `Puzzle` object with a directed graph G and vertex s . Give a short description of what the method `tagged` returns for an input vertex v if the object is initialized with a directed graph G and a vertex s ?

Solution

The `tagged` method returns whether the vertex (v) has been visited during the recursive depth first traversal (`puzzle`) that starts from the source vertex (s).
It returns `True` if vertex (v) is reachable from the starting vertex (s) in the directed graph (G).
Otherwise it raises a `KeyError` if v is not reachable from s .

- (2 points) Suppose that we represent a directed graph using the adjacency-lists representation. When we create a new `Puzzle` object, what is the runtime complexity of this action *in the worst case*?
☐ $O(1)$ ☐ $O(E)$ ☐ $O(V)$ ☒ $O(E + V)$ ☐ $O(V^2)$ ☐ $O(E^2)$
- (2 points) Suppose that we represent a directed graph using the adjacency-lists representation. When we create a new `Puzzle` object, what is the runtime complexity of this action *in the best case*?
☒ $O(1)$ ☐ $O(E)$ ☐ $O(V)$ ☐ $O(E + V)$ ☐ $O(V^2)$ ☐ $O(E^2)$
- (2 points) To store sparse graphs, which form would we prefer to use?
☐ Adjacency matrix
☒ Adjacency list

4 Divide and Conquer (14 points)

Consider the three following divide and conquer algorithms: (1) Mergesort, (2) Quicksort (if partitioning always divides the array in half), (3) Binary search in a sorted array.

- (3 points) For each algorithm, write down a recurrence relation showing how $T(N)$, the running time on an instance of size N , depends on the running time of a smaller instance. Use c to denote costs incurred from constant time operations.

Solution

Merge sort $T(N) = 2T(N/2) + cN$

Quicksort $T(N) = 2T(N/2) + cN$

Binary search $T(N) = T(N/2) + c$

- (3 points) For each algorithm, write down the corresponding runtime for each $T(N)$ (e.g., $T(N) \sim cN^2$)

Solution

Merge sort $T(N) = O(N \log N)$

Quicksort $T(N) = O(N \log N)$

Binary search $T(N) = O(\log N)$

work

MS $T(N) = 2T(N/2) + cN$ $a=2$ $b=2$ $f(N)=cN$
 QS $f(N) = O(N \log_b a) = O(N)$
 $T(N) = O(N \log N)$

BS $T(N) = T(N/2) + c$ $a=1$ $b=2$ $f(N)=O(1)$
 $f(N) = O(N \log_b a) = O(1)$
 $T(N) = O(\log N)$

3. (4 points) Given the following list [7, 3, 10, 2, 1, 5, 2], draw out how the Mergesort algorithm would process the list (i.e., use boxes to split up the list and denote subroutines that are called in the algorithm)

Solution

```

SPLIT [7, 3, 10, 2, 1, 5, 2] → [7, 3, 10] & [2, 1, 5, 2]
  SPLIT [7, 3, 10] → [7] & [3, 10]
    SPLIT [3, 10] → [3] & [10]
  SPLIT [2, 1, 5, 2] → [2, 1] & [5, 2]
    SPLIT [2, 1] → [2] & [1]
    SPLIT [5, 2] → [5] & [2]
  MERGE [7] & [3, 10] → [3, 7, 10]
  MERGE [2] & [1] → [1, 2]
  MERGE [5] & [2] → [2, 5]
  MERGE [1, 2] & [2, 5] → [1, 2, 2, 5]
  MERGE [3, 7, 10] & [1, 2, 2, 5] → [1, 2, 2, 3, 5, 7, 10]

```

4. (4 points) Given the following list [7, 3, 10, 2, 1, 5, 2], draw out how the Quicksort algorithm would process the list (i.e., use boxes to split up the list and denote subroutines that are called in the algorithm).

Solution

```

PIVOT (7) & partition [7, 3, 10, 2, 1, 5, 2] → [3, 2, 1, 5, 2]
  (less than 7) and [10] (greater than 7)
  RECUR on [3, 2, 1, 5, 2] with 3 as the pivot
    [3, 2, 1, 5, 2] → [2, 1, 2] (less than 3) and [5] (greater than 3)
    RECUR on [2, 1, 2] with 2 as the pivot
      [2, 1, 2] → [1] (less than 2) & [2] (equal to 2)
    Combine:
      [1], [2], and [2] → [1, 2, 2]
      [1, 2, 2], [3], and [5] → [1, 2, 2, 3, 5]
      [1, 2, 2, 3, 5], [7] and [10] → [1, 2, 2, 3, 5, 7, 10]
  FINAL: [1, 2, 2, 3, 5, 7, 10]

```

5 Collaboration Questions

After you have completed all other components of this assignment, report your answers to these questions regarding the collaboration policy. Details of the policy can be found in the syllabus.

1. Did you receive any help whatsoever from anyone in solving this assignment? If so, include full details.
2. Did you give any help whatsoever to anyone in solving this assignment? If so, include full details.
3. Did you find or come across code that implements any part of this assignment? If so, include full details.

Your Answer

1.) NO

2.) NO

3.) NO