### Exam 3 Study

1. Which of the following sorting algorithms in its typical implementation gives best performance when applied on an array which is sorted or almost sorted (maximum 1 or two elements are misplaced).
   1. Quick Sort
   2. Merge Sort
   3. Insertion sort
   4. Radix Sort
2. **Review each** pro and con from the sheet Matt provided for sorting algorithms and explain each pro and con. Discuss why each helps or hurts the users of these sorting methods.

**Heaps**:

1. What are the two properties of a max heap?
2. What are the three different implementations for a priority queue? Describe the differences in runtimes of these methods: Add, remove\_max.
3. Draw a heap with 10 elements
4. Write the array representation of the heap
5. Where will we find the max element in the heap? Second Max?
6. Perform these operations on the heap:

Add(90)

Add(80)

Add(40)

Remove\_Max()

Remove\_Max()

1. Draw the resulting heap and its array representation.
2. Describe any problem that a heap could help improve.

**Huffmans Trees**

1. Describe the process of constructing a Huffman tree and explain how it is used to encode text efficiently.

2. Que – 1. Which of the following is true about Huffman Coding?

(A) Huffman coding may become lossy in some cases

(B) Huffman Codes may not be optimal lossless codes in some cases

(C) In Huffman coding, no code is prefix of any other code.

(D) All of the above

3. Que – 2. How many bits may be required for encoding the message ‘mississippi’?

4. Encodes the following line using the shortest possible bit string from huffmans tree algo:

"mississippi”

5. Draw the frequency array and Huffman tree for the following string: "dogs do not spot hot pots or cats".

**Merge sort**

1. Is merge sort inplace? Explain why

2. Is merge sort stable? Explain why

3. How can we make merge sort more efficient?

4. What types of datasets is merge sort best with?

5. Explain how merge sort is a divide and conquer algorithm

6. When does the worst case of merge sort occur?

7. Visualize merge sort algorithm on a set of numbers: [2, 5, 3, 7, 5, 1, 8]

8. Explain the best and worst case time complexity with inputs also

9. Look over the code to confirm techniques used in your visualization

**Quick sort**

1. Describe the goal of one iteration of the quick sort algorithm

2. Discuss the importance of choosing a pivot element in Quick Sort and how it affects the algorithm's efficiency.

3. Visualize quick sort on a set of numbers: [4, 7, 3, 9, 1]

4. On which input might a quick sort not be the best choice?

5. Can you explain the concept of partitioning in Quick Sort and its significance in the algorithm's performance?

6. What are the limitations of the Quick Sort algorithm, and how do these limitations affect its practical use in certain situations?

7. Suppose we are sorting an array of eight integers using quicksort, and we have just finished the first partitioning with the array looking like this:

2 5 1 7 9 12 11 10

Which statement is correct?

a. The pivot could be either the 7 or the 9

b. The pivot could be the 7, but it is not the 9

c. The pivot is not the 7, but it could be the 9

d. Neither the 7 nor the 9 is the pivot.

8. Explain best and worst case time complexity with inputs

9. Encourage students to look over the code to confirm their methodologies

**Radix/Bucket sort**

1. Compare and contrast Radix Sort and Bucket Sort in terms of their underlying principles, efficiency, and applicability to different types of data.

2. Discuss the concept of buckets and their role in the Bucket Sort algorithm. How does the choice of bucket size impact the sorting process?

3. Why would you use radix sort over any other sorting algorithm?

4. Sort these key-value pairs with bucket sort

S = {(1, 1), (1, 5), (2, 4), (0, 9), (0, 1), (4, 6), (8, 10), (1, 2)}

(Key, value)

5. Have students sort names with radix sort in lexographical order

6. Explain best/worst case time complexity with inputs

7. Implement Bucket Sort from scratch and analyze performance on various datasets. (DO this if you finish quickly)