# Midterm Exam (Modules 1-7)

Comp 582 MCS@Rice Fall 2022

## Instructions

For the exam, you may refer to videos, video slide sets, problem sets, live session slides, and any of your personal notes or materials. You may NOT, however, collaborate with any other people. You may also use the Internet in a limited way. You may search for something on the Internet, but you may NOT ask a question on any question answering site (like Stack Overflow). If you use something you found on the Internet, please cite whatever you found.

For quick answer questions, you may just write the answer.

For ***all other*** questions, you must justify *all* aspects of the problem — even if the problem does not explicitly ask for justification or proof.

When justifying your answer, please show all relevant work, and *only* relevant work. If you turn in spurious work, or elements of approaches you have discarded, I will assume that you think whatever you wrote should contribute to the solution. Anything you turn in will be assumed to be part of your answer.

When you have finished your exam, please upload your work to Canvas (just like the problem sets).

You may, if you wish, email your instructor a "just-in-case..." copy of your exam.

## Format

There are 20 quick answer problems, and 7 “not-so-quick” problems on the exam.

Your exam submission should consist of **exactly 1 PDF file**. The PDF can contain scans of handwritten material.

## When Giving an Algorithm

The code standard for this exam is the same as the standard for problem sets. Namely, you ***must*** use Restricted Python. The restrictions are:

- No Python dictionaries (or classes derived from dictionaries)
- No Python sets (or classes derived from sets)

You may use any other builtin Python data structures.

You may use the python `heapq` module if you want.

## Time Limit

You have a time limit of 4 hours.

## Conventions and Nomenclature

Throughout this exam, if the question asks for "complexity", the correct answer will be the worst-case time complexity of the best algorithm you know to solve the problem.

Unless otherwise stated, any answer involving complexity, the Big Oh notation is sufficient.

In any problem text that refers to "sorting", the sort order is presumed to be ascending order unless otherwise stated.

## Pledge

Do not forget to put the Honor Code Pledge on your last age of your exam (you may use a separate page if necessary). Please be sure to include your start time and end time as part of the Pledge.

## Quick Answer [each question worth 1pt]

For the next 3 problems, the phrase "union/find problem of size *N*" means that the starting number of disjoint sets is *N*

1. For the union/find problem of size *N*, what is the worst-case complexity for any single union operation when using weighted union but **no** path compression?
   Note: the cost of initialization does not enter in to the calculation

2. For the union/find problem of size *N*, what is the worst-case complexity for any single union operation when using both weighted union and path compression?
   Note: the cost of initialization does not enter in to the calculation

3. For the union/find problem of size *N*, what is the worst-case complexity for a series of $O(N)$ union/find operations when using weighted union and path compression?

4. What is the worst-case complexity for selection sort n items?

5. What is the best-case complexity for selection sort of n items?

6. What is the worst-case complexity for insertion sort n items?

7. What is the best-case complexity for insertion sort of n items?

8. What is the worst-case complexity of quicksort of n items?

9. What is the average-case for quicksort of n items?

10. What is the worst-case for merge sort of n items?

11. You are comparing 2 algorithms A, and B. You have exact running times
    $T_A(n) = 22\log(n!) + 5n + 2205$, and
    $T_B(n) = 75n\log(n) + 70000n + 200\log(n) + 100000$.

    You are interested in a Big Oh comparison. Is $T_A(n) = O(T_B(n))$, or is $T_B(n) = O(T_A(n))$ or are these 2 complexities essentially the same in the Big Oh sense?

12. What is the worst-case complexity for quickselect to find the 1st quartile (the n/4-smallest) item in a set of n items?

13. What is the average-case complexity for quickselect to find the 1st quartile (the n/4-smallest) item in a set of n items?

14. Suppose you are looking for an item with key k in a set of n unsorted items. What is the worst-case complexity to find the item (assuming it is in the set)?

15. In a 0-based heap array of size $N$, where $N > 2$, what is the highest index of a **non**-leaf element?

16. For Shellsort, let the $h$-gap sequence be $\{1, 4, 9\}$.
    Let the input sequence be

    -27, 9, 0, -4, 12, 17, -100, -8, 42, 6404, 4

    Show the sequence after the $h = 9$ gap is completed

17. Suppose you are looking for an item with key k in a set of n sorted items. What is the worst-case complexity to determine the item is not in the set?

18. What is the worst-case complexity to delete the minimum element from a min-heap of size n?

19. Given a number $f$ and a nonnegative integer $L$, what is the worst-case complexity to compute $f^L$?

20. You are comparing the worst-case complexities of 2 algorithms A and B.
    The worst-case time for A is $T_A(n) = 4000000n + 3504n \log(n) + 100000$.
    The worst-case time for B is $T_B(n) = n^2 + 2n + 4$
    Is $O(T_A) \leq O(T_B)$, or is $O(T_B) \leq O(T_A)$, or both?

## Not-so-Quick Answers

1. [ 5 pts ] Is the following code for `fn` an algorithm? Justify your answer.

```
def fn(x):
    i,j = 0,2
    while i< j:
        x[i] = 0
        i,j = i+1,j+1
```

2. [5 pts] Using any method you like, find the Big Oh of the function that solves the recurrence

$$T(n) = 3T(\tfrac{n}{2}) + n^2 + 5000n^2 \log^2(n) + 2n^3 \log^3(n)$$

3. [15 pts] For all parts of the next question, suppose there is a function d($S$) that computes the best 1st quintile of a set $S$ of numbers. That is, the $d$ function returns the value of the $\frac{|S|}{5}$-th smallest item in the set sorted($S$). In a 0-based array, the $d$ function returns the element at index $\frac{|S|}{5} - 1$ in the sorted array. As an example, let

$$S_0 = \{1, 3, -5, -2, 0, 1000, 4, 7, 10, 15, 11\}$$

Then the sorted array is

$$\{-5, -2, 0, 1, 3, 4, 7, 10, 11, 15, 1000\}$$

So, the $d(S_0)$ value is the value that appears at index $\frac{|S_0|}{5} - 1$ (0-based indexing) in the sorted array. That means that

$$d(S_0) = -2$$

3.1. [5 pts] Write the Restricted Python code for a modified quicksort algorithm that takes an array $S$ as input, and then subsequently uses $d(S)$ as the pivot element. Your modified quicksort algorithm **must** use the COMP 582 quicksort partitioning algorithm that always pivots on the 1st element in an array.

3.2. [5 pts] Suppose the complexity of $d(S)$ is $O(1)$. Give the worst-case complexity of your modified quicksort algorithm in problem 3.1. Show how you derived the worst-case complexity (Big Oh is sufficient). In your derivation, you may assume that the subproblems always divide into $\frac{n}{5}$ and $\frac{4n}{5}$ where $n = |S|$.

3.3. [5 pts] Suppose the complexity of $d(S)$ is $O(\log(n))$. Give the worst-case complexity of your modified quicksort algorithm in problem 3.1. Show how you derived the worst-case complexity (Big Oh is sufficient). In your derivation, you may assume that the subproblems always divide into $\frac{n}{5}$ and $\frac{4n}{5}$ where $n = |S|$.

4. [10 pts] Find the Big Oh solution to the recurrence

$$T(n) = T\left(\frac{3n}{4}\right) + T\left(\frac{n}{4}\right) + \theta\left(n^2\right)$$

5. [10 pts] Suppose we want to maintain an array of the largest $M$ numbers of a stream of numbers of length $L$. By "maintain", we mean that at any point in the stream of numbers, the array contains the $M$ largest elements seen at that point in the stream. Find an efficient algorithm to maintain this array. Also, give the complexity of your algorithm, and show how you derived the complexity of your algorithm.
   NOTE: The array of the $M$ largest elements does not have to be ordered.

6. [20 pts] Suppose we have a k sets of various sizes. All k of the sets are sorted. The total number of elements is N. You have available a `bmerge` routine that performs a binary merge of 2 sets $S_1, S_2$ in exactly $|S_1| + |S_2|$ operations. The total cost of the `bmerge` sequence is the sum of the costs of all of the `bmerge` operations. To describe a `bmerge` sequence, name the result of a `bmerge` a new name (/index). Example:
$S_0, S_1, S_2$ are initial sets. Then a bmerge sequence to merge all of these sets could be:

$S_3 = \text{bmerge}(S_1, S_2)$
$S_4 = \text{bmerge}(S_0, S_3)$

6.1. [2 pts] Suppose there are 4 sets $S_0, S_1, S_2, S_3$ of sizes 3,3, 4, and 7.
Give a `bmerge` sequence that results in the largest number of total operations to merge all sets

6.2. [2 pts] For the same 4 sets (sizes 3, 3, 4, and 7),
Give the `bmerge` sequence that results in the smallest number of total operations to merge all sets.

[**HINT:** 8.1 and 8.2 have different answers!]

6.3. [16 pts]. For this problem, you want to create an efficient algorithm that produces the smallest cost `bmerge` sequence. For example, when your algorithm gets `[7,3,4,3]` as input, it returns the sequence you gave for 6.2.

Write your algorithm in Restricted Python code. Remember to give a new index for intermediate merges. For example, if your first `bmerge` is of original sets 0 and 2 (out of 3 original sets), then the index/name of your 1st bmerge will be 3. Future `bmerge`s can now refer to this intermediate stage as set 3.The output of your algorithm is a list of `bmerges` For example, given 3 original sets 0,1,2, your output could be:

3 = `bmerge`(0,2)
4 = `bmerge`(1,3)

In addition to your code, give the complexity of your algorithm.

7. [15 pts] You are given a list of numbers $L$ as input. You must process $L$ in a specific fashion:

    1. Find the maximum of $L$
    2. Find the 2nd largest element of $L$

    The complexity measure for this problem is an ordered pair:
    $O(\text{step1}), O(\text{step2})$

    Note: This is **not** the same as summing the 2 complexities. Comparison is done in lexicographic order. For example, a naive algorithm would have paired complexity:

    $$O(n), O(n)$$

    Similarly, sorting first, then picking the right elements would have paired complexity

    $$O(n \log(n)), O(1)$$

    The paired complexity of the sort-first method is worse than the naive method, since lexicographic order has $O(n \log(n)) > O(n)$.

    Your mission is to find a processing algorithm that is better than the naive algorithm above. Write the Restricted Python code for your algorithm and give the paired complexity for your algorithm.