Exercise 1

1.1

1.1. (*5 points*) $5(n^3)^2 + \sqrt{n^7}$ is:

☐ **a)** $\Theta(n^7)$ ☐ **b)** $\Theta(n^6)$ ☒ **c)** $\Theta(n^5)$ ☐ **d)** $\Theta(n^4)$

1.2. (*5 points*) $n^2\sqrt{\lg n}$ is:

☐ **a)** $O(n^2)$ ☐ **b)** $\Theta(n \lg n)$ ☐ **c)** $O(n \lg n)$ ☒ **d)** $\Omega(n)$

1.2

2.1. (*2 points*) Is this recurrence in the format that can be solved by the master method? If yes, choose which case should be used?

☐ **a)** No, the master method cannot solve the recurrence
☐ **b)** Yes, case 1 should be used
☒ **c)** Yes, case 2 should be used
☐ **d)** Yes, case 3 should be used

2.2. (*4 points*) If you could apply the master method, choose possible a and b values.

☒ **a)** $a = 4, b = 4$ ☐ **b)** $a = 4, b = \frac{1}{4}$
☐ **c)** $a = 1, b = 4$ ☐ **d)** None of above.

2.3. (*4 points*) Choose the correct solution for $T(n)$.

☒ **a)** $\Theta(n \lg n)$ ☐ **b)** $\Theta(n)$ ☐ **c)** $\Theta(\sqrt{n})$ ☐ **d)** $\Theta(\lg n)$

1.3

3.1. (*4 points*) Which of the following statements is true?

- [ ] **a)** The worst case time complexity of merge sort is $\Theta(n^2)$.
- [x] **b)** Merge sort is an in place sorting algorithm.
- [x] **c)** The average case time complexity of merge sort is $\Theta(n \lg n)$.
- [ ] **d)** None of above is true.

3.2. (*6 points*) Given a sequence of numbers $A = \langle 4, 5, 7, 2, 1, 6, 8, 3 \rangle$, and we call MERGESORT$(A, 1, 8)$. In the following sequences, which one is the sequence A after the MERGE() procedure has been executed **three** times?

- [ ] **a)** `2, 4, 5, 7, 1, 3, 6, 8`
- [ ] **b)** `4, 5, 2, 7, 1, 6, 3, 8`
- [ ] **c)** `1, 2, 3, 4, 5, 6, 7, 8`
- [x] **d)** `2, 4, 5, 7, 1, 6, 8, 3`

1.4

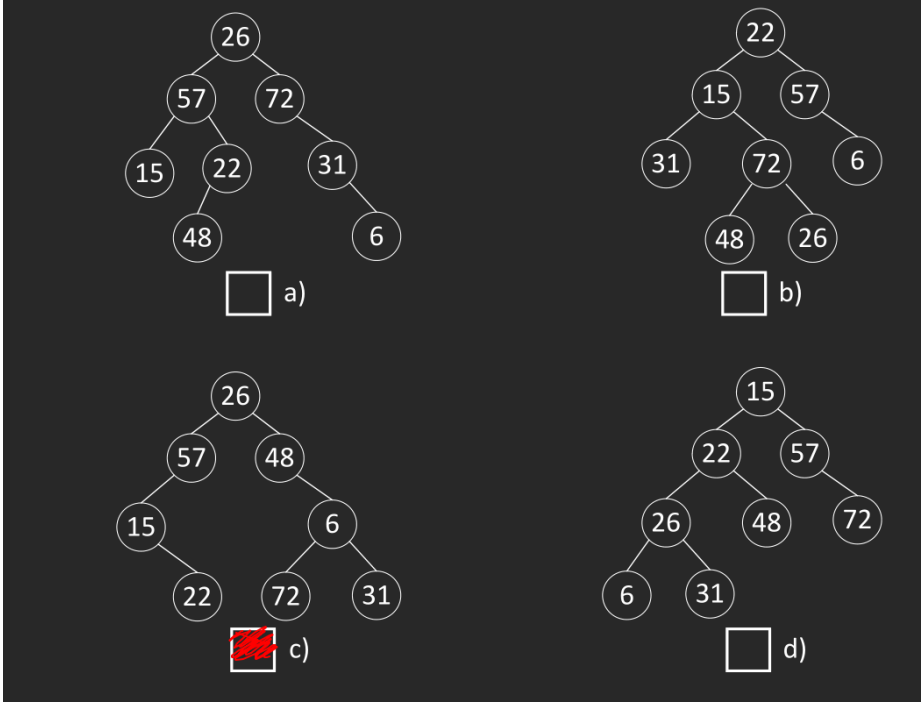
- [ ] **a)** 27, 21, 25, 30, 32, 6
- [ ] **b)** 14, 17, 16, 18, 13
- [x] **c)** 26, 19, 20, 17, 15, 14, 9, 10
- [ ] **d)** 5, 3, 2, 6

1.5

- [ ] **a)** `11, 10, 2, 3, empty, 95, empty, empty, 8`
- [ ] **b)** `empty, 10, 2, 3, empty, empty, 11, empty, 8`
- [ ] **c)** `empty, 10, 2, 3, 95, 11, empty, empty, 8`
- [x] **d)** `empty, 10, 2, 3, 11, 95, empty, empty, 8`

1.6

6.1. (*6 points*) Which of the following tree is correct for T?



6.2. (*4 points*) What does a post-order tree walk on T produce?

- ☐ **a)** `22, 15, 57, 72, 31, 6, 48, 26`
- ☐ **b)** `26, 57, 48, 15, 6, 22, 72, 31`
- ☐ **c)** `22, 15, 57, 26, 48, 6, 72, 31`
- ☐ **d)** `15, 22, 57, 26, 72, 31, 48, 6`

Exercise 2

Given the assumption that the queue always has elements in it we do not have to worry about underflow when dequeing.

Since we are using a singly-linked-list we do not have to worry about overflow either.

I will assume that the queue implementation has the properties `q.head` and `q.tail` which will point to the head and the tail respectively.

Since we are working with pointers to to nodes in the linked list both of these operations can make use of the previously mentioned properties to get the desired nodes, so long as we update them when we are done.

```
def enqueue(q, n)
    q.tail.next = n
    q.tail = n

def dequeue(q)
    n = q.head
    q.head = q.head.next
    return n
```

Exercise 3

First thing to do is to formalize the problem.

Input: An array of integers $[a_1, \ldots, a_n]$

Output: An element a such that $a \leq a_i, 1 \leq i \leq n$.

3.1

Here we can use the same structure as merge sort, except our merge of values will be a little different.

```
def min_element(xs)
    return min_element'(xs, 1, xs.length)

def min_element'(xs, p, r)
    if p < r
        q = floor((p + r) / 2)
        m1 = min_element'(xs, p, q)
        m2 = min_element'(xs, q + 1, r)
        return if m1 < m2 then m1 else m2
    else
        return xs[p]
```

The algorithm works by halving the array just like merge sort. However, when we reach a sub-array of length 1 we return the element instead of doing nothing. Now when we merge we will have to find the least of the two sub-arrays.

3.2

Again we will see that it is very similar to merge sort. But, the combine step will be $\Theta(1)$ instead of $\Theta(n)$. We will get 2 sub-problems, and each sub-problem will have size $n/2$ relative to their parent problem.

$T(1) = \Theta(1)$

$T(n) = 2T(n/2) + \Theta(1)$

3.3

Since the recurrence is in the form $T(n) = aT(n/b) = f(n)$, we can solve it using the Theorem 4.1 in CLRS (the master theorem)

$a = 2, b = 2$

$n^{log_2(2)} = n^1 = n$

Let say that $\epsilon = 1/2$

Then we can use case 1 since $\Theta(1) = O(n^{1/2})$.

Thus $T(n) = \Theta(n)$

3.4

When the problem gets bigger we will have to save more space as we will have to saves `m1`'s while calculating `m2`'s. Every time the input size doubles we will need to save another `m1`. In fact it uses $lg(n) + 1 = \Theta(lg(n))$ space.