Question 1

Identifying asymptotic notation. (Note: lg means logarithm in base 2)

(1.1) [3 Pts] Mark **ALL** the correct answers. $\lg n^{100} + \sqrt{n} + \lg 8^n$ is

☐ **a)** $\Theta(\lg n)$ ☑ **b)** $\Theta(n)$ ☐ **c)** $\Theta(\sqrt{n})$ ☐ **d)** $\Theta(n^2)$ ☐ **e)** $\Theta(2^n)$

(1.2) [3 Pts] Mark **ALL** the correct answers. $n \lg n^2 + \lg 32^n + 100n^2$ is

☐ **a)** $O(\lg n)$ ☐ **b)** $O(n)$ ☐ **c)** $O(n \lg n)$ ☑ **d)** $O(n^2)$ ☑ **e)** $O(2^n)$

(1.3) [3 Pts] Mark **ALL** the correct answers. $n \lg n^2 + \lg 5^n + 100n^4$ is

☑ **a)** $\Omega(\lg n)$ ☑ **b)** $\Omega(n)$ ☑ **c)** $\Omega(n \lg n)$ ☑ **d)** $\Omega(n^2)$ ☐ **e)** $\Omega(2^n)$

(1.4) [10 Pts] Mark **ALL** the correct answers. Consider the following recurrence

$$T(n) = \begin{cases} 60 & \text{if } n = 1 \\ T(3n/4) + \sqrt{n} & \text{if } n > 1 \end{cases}$$

☐ **a)** Using the second case of the master method one can prove $T(n) = \Theta(n^2)$

☐ **b)** The master method cannot be used to solve the above recurrence

☐ **c)** Using the first case of the master method one can prove $T(n) = \Theta(\sqrt{n})$

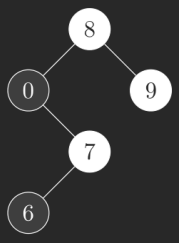
☐ **d)** Using the master method one can prove that $T(n) = O(n^2)$.

☑ **e)** Using third case of the master method one can prove $T(n) = \Theta(\sqrt{n})$

Question 2

2.1

(2.1) [3 Pts] Mark **ALL** the correct statements. Consider the tree T depicted below



- ☐ **a)** The height of T is 4
- ☑ **b)** T satisfies the binary-search tree property
- ☑ **c)** T satisfies the red-black tree property (NIL nodes are omitted)
- ☐ **d)** T satisfies the max-heap property
- ☐ **e)** T corresponds to the array $[8, 0, 9, 7, 6]$ interpreted as a binary tree.

2.2

(2.2) [5 Pts] Mark **ALL** the correct statements. Consider a modification to MERGE-SORT(A, p, r), called MI-SORT, that when $p < r$ and $r - p \leq 5$ calls INSERTION-SORT on the subarray $A[p\,..\,r]$, otherwise it works as MERGE-SORT.

- ☐ **a)** MI-SORT worst-case running time is $\Theta(n^2)$
- ☑ **b)** MI-SORT worst-case running time is $\Theta(n \lg n)$
- ☑ **c)** MI-SORT best-case running time is $\Theta(n \lg n)$
- ☐ **d)** MI-SORT best-case running time is $\Theta(n)$
- ☐ **e)** MI-SORT works in-place

2.3

(a):

The algorithm is very similar to merge-sort, therefore the recurrence is going to look a lot like the one for merge-sort. Except we can combine solution in constant time.

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

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

(b):

As the recurrence is in the form $T(n) = aT(n/b) + f(n)$ we can use theorem 4.1 from CLRS (the master theorem). We have the flowwing values: $a = 2, b = 2, f(n) = \Theta(1)$. Using case 1 we get that $\Theta(1) = O(n^{log_2(2)-\epsilon}) = O(n^{1-\epsilon})$, which have has to be true for $\epsilon > 0$, thus $\epsilon = 1$ works and we have that, $T(n) = \Theta(n)$.

Question 3

3.1

- ☐ **a)** The tree below to the left shows the result of RB-INSERT$(T, 25)$
- ☑ **b)** The tree below in the centre shows the result of RB-INSERT$(T, 25)$
- ☐ **c)** The tree below to the right shows the result of RB-INSERT$(T, 25)$
- ☐ **d)** None of the trees below represents the result of RB-INSERT$(T, 25)$
- ☑ **e)** Only one tree among the trees below satisfies the red-black tree property

3.2

(3.2) [5 Pts] Consider the hash table $H = \text{NIL}, 12, 101, 24, \text{NIL}, 5, \text{NIL}, \text{NIL}, 8, 20, \text{NIL}$. Insert the keys $18, 110, 49$ in H using *open addressing* with the auxiliary function $h'(k) = k$.

Mark the hash table resulting by the insertion of these keys using linear probing.

- ☐ **a)** $110, 12, 101, 24, 49, 5, \text{NIL}, 18, 8, 20, \text{NIL}$
- ☐ **b)** $110, 12, 101, 24, \text{NIL}, 5, \text{NIL}, 18, 8, 20, 49$
- ☑ **c)** $110, 12, 101, 24, \text{NIL}, 5, 49, 18, 8, 20, \text{NIL}$
- ☐ **d)** none of the above

Mark the hash table resulting by the insertion of these keys using quadratic probing with $c_1 = 1$ and $c_2 = 4$.

- ☐ **a)** $110, 12, 101, 24, 49, 5, \text{NIL}, 18, 8, 20, \text{NIL}$
- ☑ **b)** $110, 12, 101, 24, \text{NIL}, 5, \text{NIL}, 18, 8, 20, 49$
- ☐ **c)** $110, 12, 101, 24, \text{NIL}, 5, 49, 18, 8, 20, \text{NIL}$
- ☐ **d)** none of the above

Mark the hash table resulting by the insertion of these keys using double hashing with $h_1(k) = k$ and $h_2(k) = 1 + (k \mod (m-1))$.

- ☑ **a)** $110, 12, 101, 24, 49, 5, \text{NIL}, 18, 8, 20, \text{NIL}$
- ☐ **b)** $110, 12, 101, 24, \text{NIL}, 5, \text{NIL}, 18, 8, 20, 49$
- ☐ **c)** $110, 12, 101, 24, \text{NIL}, 5, 49, 18, 8, 20, \text{NIL}$
- ☐ **d)** none of the above

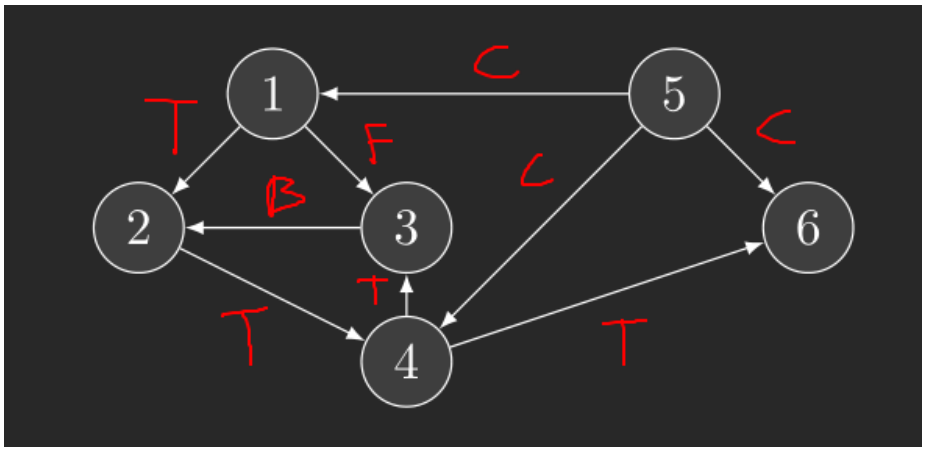
3.3

(a):

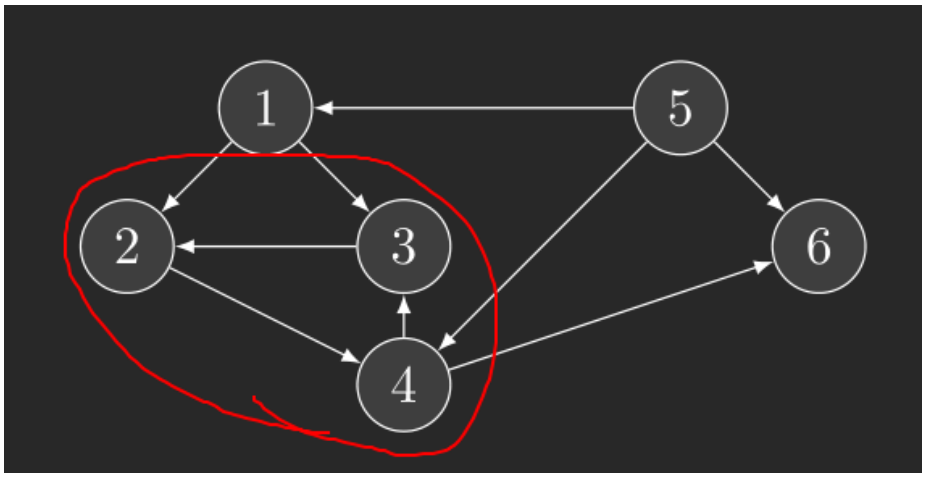
- 1: 1/10
- 2: 2/9
- 3: 4/5
- 4: 3/8
- 5: 11/12
- 6: 6/7

(b): (1 (2 (4 (3 3) (6 6) 4) 2) 1) (5 5)

(c):



(d):



Question 4

In a bottom-up dynamic programming algorithm we want to start with a base case, that is problem that does not depend on any other sub-problems, and then save the result go to a bigger problem that depends on the saller problem that was just solved. In the end we should be able to solve the problem for size L.

```
def min_blocks(len, sizes)
    let l[0..len] be a new array filled with infinity
    l[0] = 0

    for i = 1 to len
        for s in sizes
            if i - s >= 0
```

```
                l[i] = min(l[i], 1 + l[i - s])

    return l[len]
```

The worst case running-time is $\Theta(L \cdot S)$ as they are the upper bounds in the nested for loops. Everything else is $\Theta(1)$, besides line 1 which is $\Theta(L)$.

Question 5

5.1

As w represents the distances between to vertices we can safly assume that no weighting will be negative. For this algorithm we will give each edge a property `h` which will be a tuple containing the hospital and the distance to that hospital. By calculating the single-source distances for each hospital we can find the shortest distance to the hospitals.

```
def nearest_hospital(G, w)
    for every vertex v in G
        v.h = (nulll, infinity)

    for h in G.H
        dijkstra(G, w, h)
        for every vertex v in G
            if v.d < v.h.snd    // get distance from the tuple
                v.h + (h, v.d)  // set the new hospital and distance
```

Assuming Dijkstra's is implemented using an array we get a worst case running time of $O(H \cdot V^2)$.

5.2

Since it a directed graph we can not just assume that we can pick a location for the firestation and then calculate the distance from the vertex containing the firestation to every other vertex, as we cannot be sure that we can go backwards. We can however, do this if we find G^T. Then we can calculate the all-pairs-shortest-path and find the placement which gives shortent mean distance.

For this we will be using an adjencency-matrix. With this we can compute the transpose in $\Theta(V^2)$ time, as we were taught in lecture 10. As we are using an adjencency-matrix the weights are already within the graph G. The entry being the weight from vertex i to j if the is a path and infinity otherwise.

```
def firestation(G)
    GT = transpose(G)
    D = floyd_warshall(G)
    firestation = (null, infinity)
```

```
for i in G
    let current = (i, 0)
    for j in i
        current.snd += j
    current.snd /= G.length
    if current.snd < firestation.snd
        firestation = current
return firestation
```

Seeing as transpose is $\Theta(V^2)$, `floyd_warshall` is $\Theta(V^3)$ and my own added code is $\Theta(V^2)$ the running time will be $\Theta(V^3)$.