

Exercise 1

1.1

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

- ☐ a) $\Theta(n \lg n)$ ☐ b) $\Theta(n)$ ☐ c) $\Theta(\sqrt{n})$ ☒ d) $\Theta(n^2)$ ☐ e) $\Theta(n^{1.5})$

1.2. (3 points) $700 \cdot n^2 + 999 \cdot n^2 \lg n + 0.1 \cdot n^2 \lg^2 n$ is:

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

1.2

- ☐ a) $f(n) = n^2, g(n) = n^2$
☒ b) $f(n) = 2^n, g(n) = 2^n$
☒ c) $f(n) = 2^n, g(n) = 4^n$
☐ d) $f(n) = n \lg n, g(n) = 1$

1.3

- ☐ a) $\Theta(n^4 \lg n)$ ☐ b) $\Theta(n^2)$ ☐ c) $\Theta(n^3)$ ☒ d) $\Theta(n^4)$

1.4

- ☐ a) Insertion sort
- ☐ b) Selection sort
- ☐ c) Merge Sort
- ☒ d) Quick Sort

1.5

- ☒ a) Max-Heapify(A, 4), Max-Heapify(A, 2), Max-Heapify(A, 3), Max-Heapify(A, 1)
- ☒ b) Max-Heapify(A, 3), Max-Heapify(A, 4), Max-Heapify(A, 2), Max-Heapify(A, 1)
- ☒ c) Max-Heapify(A, 1), Max-Heapify(A, 2), Max-Heapify(A, 4), Max-Heapify(A, 3)
- ☒ d) Max-Heapify(A, 3), Max-Heapify(A, 4), Max-Heapify(A, 1), Max-Heapify(A, 2)

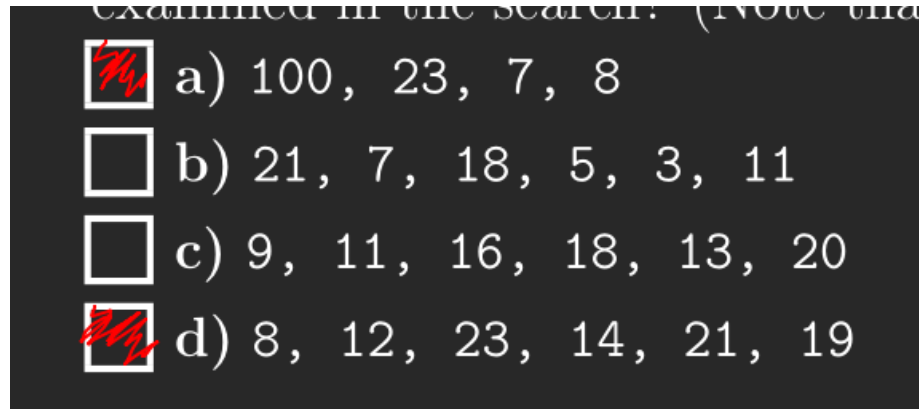
1.6

- ☐ a) $\Theta(1)$
- ☐ b) $\Theta(\lg n)$
- ☐ c) $\Theta(n)$
- ☐ d) $\Theta(n \lg n)$
- ☒ e) $\Theta(n^2)$
- ☐ f) $\Theta(n^3)$

1.7

The correct should be [_, 1, 10, 17, 13, 5, _, _, 9]

1.8



Exercise 2

2.1

We can represent each combination as a binary number. Given that there are five questions it would be 5-bit. Doing the questions Q_1, Q_3, Q_5 would be represented as 10101. Since we are evaluating each combination and each combination has a unique binary representation we would need to evaluate $2^5 = 16$ cases, or 2^n given n questions.

2.2

This can easily be restated as an instance of the knapsack problem. The main idea in the algorithm is to save results in an array $V[0..n][0..X]$. Saving the results saves us from recomputing a lot of values as per usual in dynamic programming. In the end we will have a value for $V[n][X]$ which we can return.

The recurrence is as follows

$$V(0,0) = 0$$

$$V(i,j) = V(i-1,j), \text{ if } t_i > j$$

$$V(i,j) = \max(v_i + V(i-1, j - t_i), V(i-1, j)), \text{ otherwise}$$

```
def student_questions(v, t, X)
    n = v.length
    let V[0..n][0..X] be filled with 0's
    for i = 1 to n
        for j = 1 to X
            if t[i] > j
                V[i][j] = V[i-1][j]
            else
```

```

        if V[i - 1][j] >= v[i] = v[i - 1][j - t[i]]
            V[i][j] = v[i - 1][j]
        else
            V[i][j] = v[i] + V[i - 1][i - t[i]]
    return V[n][X]

```

Due to the initialization of V and the double nested for loops we get a run-time of $\Theta(n \cdot X)$.

Exercise 3

3.1