Exam 1

Yaoken Wu

1.

a) $\Theta(n^2 \log n)$

b). $\Theta(n)$

c). $\Theta(n^{\log_3 5})$

d). $\Theta(n^3)$

2. a.

$$SCS(i,j) = \begin{cases} i & \text{if } j = 0 \\ j & \text{if } i = 0 \\ \min\begin{cases} SCS(i,j-1)+1 \\ SCS(i-1,j)+1 \end{cases} & \text{if } A[i] \neq B[j] \\ SCS(i-1,j-1)+1 & \text{otherwise} \end{cases}$$

b.

```
sort(L[1..n]):
    k ← ⌊n/3⌋
    X[1..k] ← sort(L[1..k])
    Y[1..k] ← sort(L[k+1..2k])
    Z[1..m] ← sort(L[2k+1, n])
    W[1..n] ← concatenate X, Y, Z
    return MERGE3(W[1..n])
```

Time complexity:

Assume the MERGE3 has $O(n)$ time complexity.

The total running time would be $O(n \log n)$.

3. Assume there are at least 3 elements in the array A.

```
FindLocalMinimum(A[1..n]):
    mid ← ⌈n/2⌉
    if A[mid-1] ≥ A[mid] and A[mid] ≤ A[mid+1]:
        return A[mid].
    else:
        if A[mid-1] < A[mid+1]:
            return FindLocalMinimum(A[1..mid])
        else:
            return FindLocalMinimum(A[mid..n])
```

Proof:

Since $A[1] \geq A[2]$ and $A[n-1] \leq A[n]$, there must be at least one local minimum.

We first find the middle point.

Case 1: if it satisfy the condition, return directly.

Case 2: if it's left less than its right,
We search the left half because
A[mid-1] ≤ A[mid], which means
that the boundary condition still
holds and there must be at least
one local minmum in the left
subarray

Case 3: if it's right less than its left,
We search the right half because
A[mid] ≥ A[mid+1], which means
that the boundary condition still
holds and there must be at least
one local minmum in the right
subarray.

When the subarray size reduced to 3,
we will definitely return the middle
point.

Time complexity:
O(logn) because each time we
reduce the search space by half.

4.

a).

Basecase : $MaxProfit(i) = 0$ when $i = n$

$MaxProfit(i) = \max_{i<j\le n} (P[j-i]-10 + MaxProfit[j])$

proof:

The maxprofit(i) is given by the maximum profit selled on day j, where $P[j-i]-10$ is the profit of selling the current plant.

Since at the same day we plant a new one, maxprofit[j] is also included in our profit.

b).

```
Maxprofit(P[1...n]):
    DP[n] ← 0
    for i = n-1 to 0:
        for j = i+1 to n:
            DP[i] ← max(DP[i], P[j-i] -10 + DP[j])
Return DP[0].
```

Proof:

Since the maxprofit(i) depends on maxprofit(j) where $j > i$

we should evaluate the i from n to 0. and the solution is in maxprofit(0).

Time complexity:

$O(n^2)$ because i goes from n to 0 and for each i, j will go from i to n.