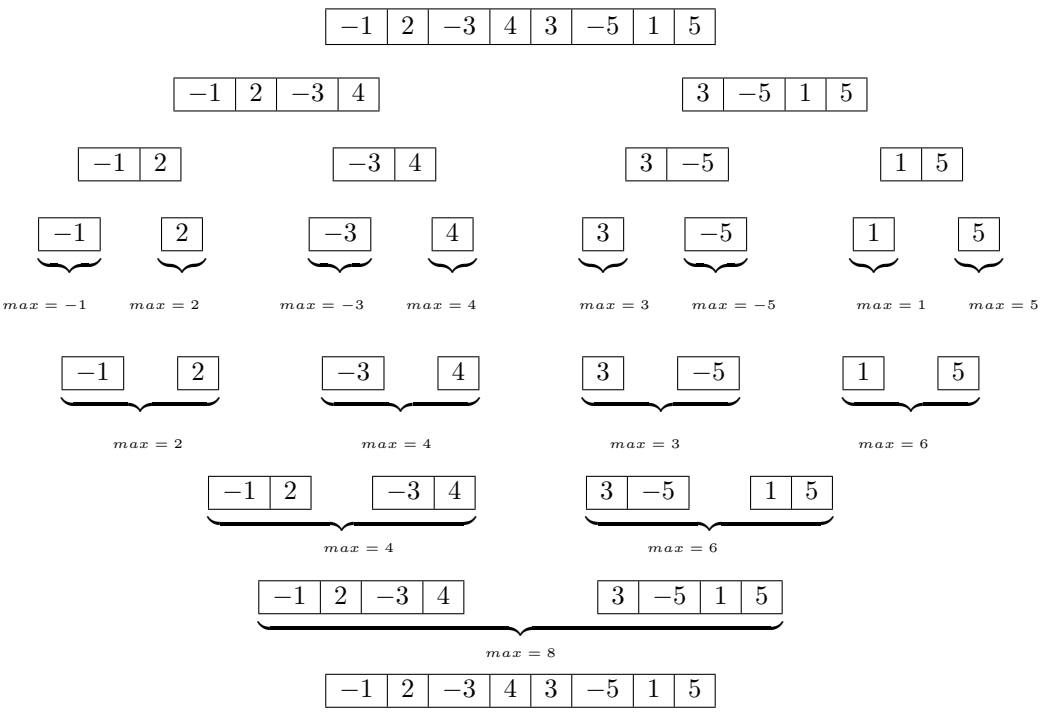
Informatics II, Spring 2023, Solution 4

Publication of exercise: March 12, 2023
Publication of solution: March 19, 2023
Exercise classes: March 20 - March 24, 2023

Task 1

1.1



1.2

```
int maxOverlapArraySum(int arr[], int l, int m, int r) {
    int sum = 0, i=0;
    int left_sum = -50000;
    for ( i = m; i >= l; i--) {
        sum = sum + arr[i];
        if (sum > left_sum) {
          left_sum = sum;
      }
    }
```

```
    sum = 0;
    i=0;
    int right_sum = -50000;
    for (i = m+1; i <= r; i++) {
        sum = sum + arr[i];
        if (sum > right_sum) {
          right_sum = sum;
      }
    }
    return left_sum + right_sum;
}

int maxSubArraySum(int arr[], int l, int r) {
   if (l == r)
     return arr[l];
   int m = (l + r)/2;
   int leftArraySum = maxSubArraySum(arr, l, m);
   int rightArraySum = maxSubArraySum(arr, m+1, r);
   int overlapArraySum = maxOverlapArraySum(arr, l, m, r);
   if (leftArraySum>rightArraySum && leftArraySum>overlapArraySum) {
        return leftArraySum;
   } else if (rightArraySum>leftArraySum && rightArraySum>overlapArraySum) {
        return rightArraySum;
   }
   return overlapArraySum;
}
   int max_sum = maxSubArraySum(arr, 0, n-1);
```

1.3

Recurrence: T(n) = 2T(n/2) + θ(n)
Asymptotic Complexity : $\theta(n \log n)$

Task 2

2.1 A good guess is $O(2^n)$.

```
int T(int n, int c1) {

    if (n <= 1) {
        return 1;
    }
    int t = 2*T(n-1, c1) + c1;
    printf("%d\n", t);
    return t;
}
```

2.2

Guess

$T_a = O(2^n)$
We need to show that $T_a(n) \leq k2^n - b$, for some constant a and b.

Base case

$n = 1$
$T_a(1) = 1 \leq k2^1 - b = 2k - b$. This is true for $k \geq (b+1)/2$.

Inductive step

$$\begin{aligned} T(n) &= 2T(n-1) + c_1 \\ &\leq 2(k2^{n-1} - b) + c_1 \\ &= k2^n - 2b + c_1 \\ &\leq k2^n - b \end{aligned}$$

This is true for $b \geq c_1$.

We have two constraints that need to be satisfied for our proof to work. The definition of the O notation allows us to choose any constant. Thus, by choosing the constants $b = c_1$ and $k = (b+1)/2$ we satisfy the constraints and show that the property $T_a(n) \in O(2^n)$ is true.

Task 3

Starting with n we compute the recurrence step by step moving backwards.

First, we need to compute $T(n/2) = 2T(\frac{n}{4}) + \frac{n}{2}$.

Next, we back substitute the value of $T(n/2)$ in $T(n)$:

$$\begin{aligned} T(n) &= 2T(n/2) + n \\ &= 2(2T(\frac{n}{4}) + \frac{n}{2}) + n \quad \text{back substitute} \\ &= 2^2T(\frac{n}{2^2}) + 2n \quad \text{simplify expression} \end{aligned} \tag{1}$$

If we continue to proceed with the backward substitution, we will observe a pattern that forms.

$$\begin{aligned} T(n) &= 2^2T(\frac{n}{2^2}) + 2n \\ &= 2^3T(\frac{n}{2^3}) + 3n \\ &= 2^4T(\frac{n}{2^4}) + 4n \\ &\dots \\ &= 2^kT(\frac{n}{2^k}) + kn \end{aligned} \tag{2}$$

Continue until we reach the base case $T(\frac{n}{n}) = T(1) = 1$. Note, we reach the base case after exactly $\log_2(n)$ steps ($2^{\log_2(n)} = n$). Thus, $k = \log_2(n)$

Now, replace k in the equation above.

$$\begin{aligned} T(n) &= 2^{\log_2 n} T(1) + n\log_2(n) \\ &= cn + n\log_2(n) \\ &= O(n\log_2(n)) \end{aligned} \tag{3}$$

We used $2^{\log_2(n)} = n$ and replaced $T(1)$ with a constant c. The term $n\log_2(n)$ dominates, thus, we can drop cn in the big O notation.

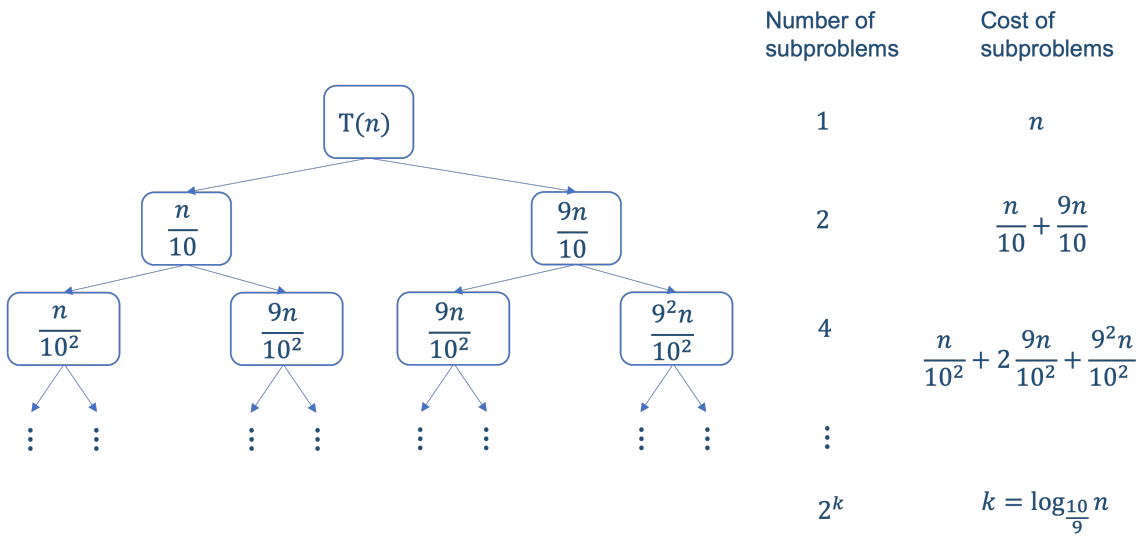
Task 4

4.1

$$T_a(n) = \begin{cases} 1, & \text{if } n = 1 \\ T(n) = T(\frac{n}{10}) + T(\frac{9n}{10}) + n, & \text{otherwise} \end{cases} \tag{4}$$

At each step, the algorithm performs two recursive calls and does work equal to n. The base case is $n \leq 1$.

4.2



In order to find the tree height, we need to find the longest path from root to a leaf node. At every branch, n is multiplied by either $\frac{9}{10}$ or $\frac{1}{10}$. Thus, the branches in which n is multiplied by $\frac{9}{10}$ will form the longest path.

We observe that the problem size shrinks as follows: $(\frac{9}{10})^0 n$, $(\frac{9}{10})^1 n$, $(\frac{9}{10})^2 n$, ...,

$$(\frac{9}{10})^k n$$

.

The size of the problem at the last level is $(\frac{9}{10})^k n$. Given the base case, we can now solve for k.

$$\begin{aligned} (\frac{9}{10})^k n &= 1 \\ (\frac{9}{10})^k &= \frac{1}{n} \\ k &= log_{\frac{9}{10}}(n) \end{aligned} \tag{5}$$

Accounting for the root node, the tree height is $\log_{\frac{10}{9}}(n) + 1$.

Now, we need to calculate the incurred cost.

At the first level of the tree we have one node (root), at the second level, we have two nodes. Number of nodes at level $k = \log_{\frac{10}{9}}(n) = 2^{\log_{\frac{10}{9}}(n)} = n^{\log_{\frac{10}{9}}(2)}$.

The cost of the sub problems at the last level is $T(1)$, thus, $n^{\log_{\frac{10}{9}}(2)} T(1) = n^{\log_{\frac{10}{9}}(2)}$.

Now sum up the costs at all levels of the tree. Given the recursive tree above, we find that

$T(n) = n + n + n + \ldots + \log_{\frac{10}{9}}(n) + O(n^{\log_{\frac{10}{9}}(2)}) = n \log_{\frac{10}{9}}(n) + O(n^{\log_{\frac{10}{9}}(2)})$

The term above is dominated by $n \log_{\frac{10}{9}}(n)$. Thus, we can state that $T(n) \in O(n \log_{\frac{10}{9}}(n))$

Task 5

5.1
- $a = 2, b = 4, f(n) = \sqrt{n} = n^{(\frac{1}{2})}$, $\frac{1}{2} = \log_4(2)$, thus, case 2 applies.
- solution: $T(n) \in O(\sqrt{n} \log_4(n))$

5.2
- $a = 12, b = 8, f(n) = n^3$, case 3 applies because the work at the root dominates.
- solution: $T(n) \in O(n^3)$