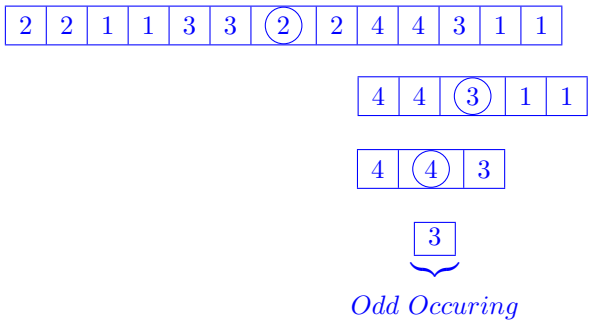
Informatics II
Exercise 4 / **Solution**

March 16, 2020

Divide and Conquer

Task 1. Given an array $A[0...n-1]$ of integers, every element has an even number of occurrences except one element with odd number of occurrences which we call odd element. An element with even occurrences appears in pairs in the array. Find an odd occuring element with an asymptotic complexity of $O(\log n)$ using divide and conquer approach.

a) Draw a tree to illustrate the process of finding the odd occuring element in the array A = [2, 2, 1, 1, 3, 3, 2, 2, 4, 4, 3, 1, 1], according to your divide and conquer algorithm.



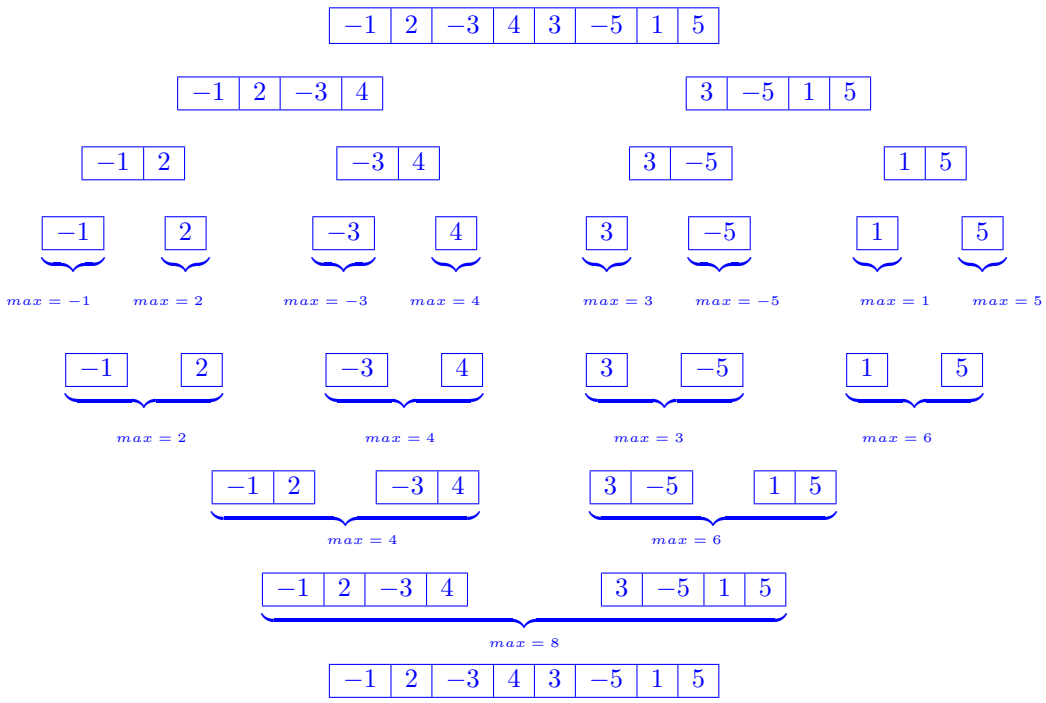
b) Provide a C code for divide and conquer floor finder algorithm.

```
1  int findOddOccuring(int arr[], int l, int r){
2      // base case
3      if (l == r) {
4          return l;
5      }
6
7      int mid = (l + r) / 2;
8      if (mid %2 == 1){ // if mid is odd
9          if (arr[mid] == arr[mid - 1])
10             return findOddOccuring(arr, mid + 1, r);
11         else
12             return findOddOccuring(arr, l, mid - 1);
13     }
```

```
14      else{ // mid is even
15          if (arr[mid] == arr[mid + 1])
16              return findOddOccuring(arr, mid + 2, r);
17          else
18              return findOddOccuring(arr, l, mid);
19      }
20  }
```

Task 2. The maximum-subarray algorithm finds the contiguous subarray that has the largest sum within an **unsorted** array `A[0...n-1]` of integers. For example, for array `A = [-2, -5, 6, -2, -3, 1, 5]`, the maximum subarray is `[6, -2, -3, 1, 5]`.
The algorithm works as follows:
Firstly, it divides the input array into two equal partitions: **I** (A[0]...A[*mid*]) and **II** (A[*mid*+1]...A[n-1]). Afterwards, it calls itself recursively on both partitions to find the maximum subarray of each partition. The combination step decides the maximum-subarray by comparing three arrays: the maximum-subarray from the left part, the maximum-subarray from the right part, and the maximum-subarray that overlaps the middle. The maximum-subarray that overlaps the middle is determined by considering all elements to the left and all elements to the right of the middle.

a) Based on the above algorithm description, draw a tree that illustrates the process of determining the maximum subarray in array `A = [-1, 2, -4, 1, 9, -6, 7, -3, 5]`.



b) Provide a C code for maximum-subarray algorithm.

```
1  int maxOverlapArraySum(int arr[], int l, int m, int r) {
2      int sum = 0, i=0;
3      int left_sum = −50000;
4      for ( i = m; i ≥l; i−−) {
5          sum = sum + arr[i];
6          if (sum > left_sum) {
7            left_sum = sum;
8        }
9      }
10     sum = 0;
11     i=0;
12     int right_sum = −50000;
13     for (i = m+1; i ≤r; i++) {
14         sum = sum + arr[i];
15         if (sum > right_sum) {
16           right_sum = sum;
17       }
18     }
19     return left_sum + right_sum;
20 }
21
22 int maxSubArraySum(int arr[], int l, int r) {
23    if (l == r)
24      return arr[l];
25    int m = (l + r)/2;
26    int leftArraySum = maxSubArraySum(arr, l, m);
27    int rightArraySum = maxSubArraySum(arr, m+1, r);
28    int overlapArraySum = maxOverlapArraySum(arr, l, m, r);
29    if (leftArraySum>rightArraySum && leftArraySum>overlapArraySum) {
30         return leftArraySum;
31    } else if (rightArraySum>leftArraySum && rightArraySum>overlapArraySum) {
32         return rightArraySum;
33    }
34    return overlapArraySum;
35 }
```

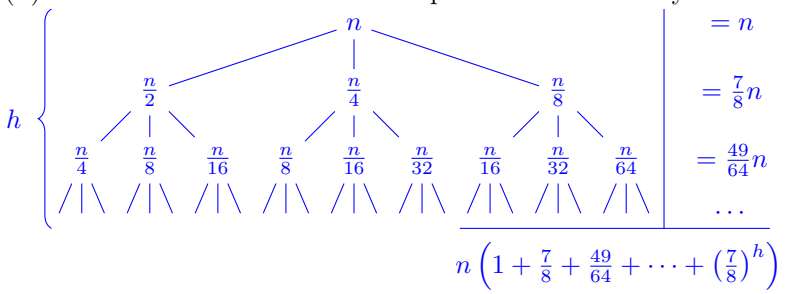
c) What is the recurrence relation of the algorithm and its asymptotic tight bound.
Recurrence: $T(n) = 2T(n/2) + \theta(n)$
Asymptotic Complexity : $\theta(n \log n)$

Recurrences

Task 3. Consider the following recurrence:

$$T(n) = \begin{cases} 1 & \text{, if } n = 1 \\ T(n/2) + T(n/4) + T(n/8) + n & \text{, if } n > 1 \end{cases}$$

a) Draw a recursion tree and use it to estimate the asymptotic upper bound of $T(n)$. Demonstrate the tree-based computations that led to your estimate.



- The tree grows until $\frac{n}{2^h} = 1$; $\implies h = \log_2 n$

Guess: $O(n)$

b) Use the substitution method to prove that your estimate in (a) is correct.

Inductive Step

- $T(n) \leq cn \implies T(n/2) + T(n/4) + T(n/8) + n \leq c\frac{n}{2} + c\frac{n}{4} + c\frac{n}{8} + n$

? $c\frac{n}{2} + c\frac{n}{4} + c\frac{n}{8} + n \leq cn$

Proof

- $c\frac{n}{2} + c\frac{n}{4} + c\frac{n}{8} + n \leq cn$
- dividing both parts by n we get $c\frac{1}{2} + c\frac{1}{4} + c\frac{1}{8} + 1 \leq c$
- $\frac{1}{8}c \geq 1 \implies c \geq 8$

Asymptotic Complexity

$T(n) = O(n)$, for $c \geq 8$

Task 4. Calculate the asymptotic tight bound of the following recurrences. If the Master Theorem can be used, write down a, b, $f(n)$ and the case (1-3).

1. $T(n) = 3T(\frac{n}{9}) + 32\sqrt{n}$
   $a = 3$, $b = 9$, $f(n) = 32\sqrt{n}$, Case 2:
   $T(n) = \Theta(\sqrt{n} \log_9 n)$

2. $T(n) = T(\sqrt[3]{n}) + \log n$

$$\begin{aligned} T(n) &= \log n + \log \sqrt[3]{n} + \log \sqrt[3]{\sqrt[3]{n}} + \dots \\ &= \log n + \frac{1}{3}\log n + \frac{1}{9}\log n + \dots \\ &= \left(\sum_{k=0}^{\infty} (\frac{1}{3})^k\right) \log n \\ &= \frac{3}{2}\log n \end{aligned}$$

$\Rightarrow T(n) \in \Theta(\log n)$

3. $T(n) = 15T(n/4) + n^2$
$a = 15$, $b = 4$, $f(n) = n^2$, Case 3:
$T(n) \in \Theta(n^2)$

4. $T(n) = 2T(n-2) + n$

$$\begin{aligned} T(n) &= 2T(n-2) + n \\ &= 2(2T(n-4) + (n-2)) + n \\ &= 4T(n-4) + 3n - 4 \\ &= 4(2T(n-6) + (n-4)) + 3n - 4 = 8T(n-6) + 7n - 20 \\ &= 8(2T(n-8) + (n-6)) + 7n - 20 = 16T(n-8) + 15n - 68 \\ &= \dots \end{aligned}$$

$$\Rightarrow T(n) = 2^k T(n-2k) + (2^k - 1)n - \sum_{i=0}^{k-1} 2^i \cdot 2i$$

k grows until it reaches $k = \lfloor \frac{n}{2} \rfloor$, thus we have:

$$T(n) = (2^{\lfloor \frac{n}{2} \rfloor} - 1)n - \sum_{i=0}^{\lfloor \frac{n}{2} \rfloor - 1} 2^i \cdot 2i \leq (2^{\lfloor \frac{n}{2} \rfloor} - 1)n - 2^{\lfloor \frac{n}{2} \rfloor - 1} 2 \cdot (\lfloor \frac{n}{2} \rfloor - 1) \in \Theta(n\sqrt{2}^n)$$

5. $T(n) = \log n + T(\sqrt{n})$

$$\begin{aligned} T(n) &= \log n + \log \sqrt{n} + \log \sqrt{\sqrt{n}} + \dots \\ &= \log n + \frac{1}{2}\log n + \frac{1}{4}\log n + \dots \\ &= \left(\sum_{k=0}^{\infty} (\frac{1}{2})^k\right) \log n \\ &= 2\log n \end{aligned}$$

$\Rightarrow T(n) \in \Theta(\log n)$