Exercise Session n. 6

Algorithms and Data Structures

Let’s practice with some more exercises in understanding, describing, and improving algorithms.

Understanding, Describing, and Improving Algorithms

Solve Exercise 206 from the collection of exam exercises (midterm, 2018) using Python. That is, use the Python code below as reference algorithms, and write your solution to Question 2 in Python.

```
def algo_x(A,x):
    i = len(A) - 1
    j = 0
    while i >= 0:
        if j == i:
            j = 0
            i = i - 1
        elif A[i] - A[j] > x or A[j] - A[i] > x:
            return True
        else:
            j = j + 1
    return False
```

Understanding, Describing, and Improving Algorithms

Solve Exercise 253 from the collection of exam exercises (midterm, 2021) using Python. That is, use the Python code below as reference algorithms, and write your solution to Question 3 in Python.

```
def algo_x(A,k):
    B = algo_y(A,0,len(A))
    c = 0
    for i in range(len(B)):
        if i < k:
            c = c + B[i]
```

```
        else:
            return c
    return c

def algo_y(A,i,j):
    D = []
    if j - i == 1:
        D.append(A[i])
    elif j - i > 1:
        k = (i + j)//2
        B = algo_y(A,i,k)
        C = algo_y(A,k,j)
        b = 0
        c = 0
        while b < k - i or c < j - k:
            if c >= j - k or (b < k - i and B[b] < C[c]):
                D.append(B[b])
                b = b + 1
            else:
                D.append(C[c])
                c = c + 1
    return D
```

Understanding, Describing, and Improving Algorithms

Solve Exercise 254 from the collection of exam exercises (midterm, 2021)

Understanding, Describing, and Improving Algorithms

Solve Exercise 288 from the collection of exam exercises (midterm, 2023)

Name (print):

Signature:

Instructions: This is a closed-book exam. Communicate your ideas *clearly* and *succinctly*. Write your solutions directly *and only* on this booklet. You may use other sheets of paper as scratch, but *do not submit anything other than this booklet*, as nothing else will be considered for grading. You may use either a pen or a pencil.

On problem	you got	out of
1		30
2		20
3		20
4		20
5		30
Total		120

►**Exercise 1.** Write an algorithm MOUNTAIN-SORT(A) that, given an array A of n numbers, sorts A in-place such that the left half of A is increasing and the right half is decreasing. More specifically, the values from $A[1]$ to $A[\lfloor n/2 \rfloor]$ are increasing and the values from $A[\lfloor n/2 \rfloor]$ to $A[n]$ are decreasing. Notice that the left and right subsequences share the element in the middle position $A[\lfloor n/2 \rfloor]$. Notice also that the resulting order is not unique. You must detail every algorithm you write. Also, analyze the complexity of your solution. *(30)*

For example, for $A = [8, 2, 5, -12, 2, 11, -15, -8, -1, 12]$, MOUNTAIN-SORT($A$) might result in $A = [-12, -8, -1, 1, 12, 11, 8, 5, 2, -15]$.

►**Exercise 2.** Consider the following algorithm that takes an array A of n numbers.

```
ALGO-X(A)
n = A.length
x = 0
for i = 1 to n
    j = 1
    while j ≤ n and (i == j or A[i] ≠ A[j])
        j = j + 1
    if j > n
        x = x + 1
return x
```

Question 1: Explain what ALGO-X does. Do not simply paraphrase the code. Instead, explain the high-level semantics of the algorithm independent of the code. *(5)*

Question 2: Analyze the complexity of ALGO-X. Is there a difference between the best and worst-case complexity? If so, describe a best and a worst-case input of size n, as well as the behavior of the algorithm in each case. *(5)*

Question 3: Write an algorithm called BETTER-ALGO-X that does exactly the same thing as ALGO-X with with a strictly better time complexity. *(10)*

►**Exercise 3.** An accounting system models a revenue transaction t as an object with two attributes, $t.date$ and $t.amount$, representing the date and amount of the transaction, respectively. Dates are represented as numbers of days since a reference initial date, such that $t_2.date - t_1.date$ is the number of days between transactions t_1 and t_2. Amounts are positive numbers. With that, consider the following ALGO-Y(T) that takes an array T of transactions:

```
ALGO-Y(T)
x = 0
for i = 1 to T.length
    l = T[i].amount
    r = T[i].amount
    for j = 1 to T.length
        if i ≠ j
            if T[j].date ≤ T[i].date and T[i].date − T[j].date ≤ 10
                l = l + T[j].amount
            if T[j].date ≥ T[i].date and T[j].date − T[i].date ≤ 10
                r = r + T[j].amount
    if x < r
        x = r
    if x < l
        x = l
return x
```

Question 1: Explain what ALGO-Y does. Do not simply paraphrase the code. Instead, explain the high-level semantics of the algorithm independent of the code. *(5)*

Question 2: Analyze the complexity of ALGO-Y. Is there a difference between the best and worst-case complexity? If so, describe a best and a worst-case input of size n, as well as the behavior of the algorithm in each case. *(5)*

Question 3: Write an algorithm called Better-Algo-Y that does exactly the same thing as Algo-Y, but with a strictly better complexity in the worst case. Analyze the complexity of Better-Algo-Y. *(20)*

►**Exercise 4.** Consider the following array

$$H = [3, 5, 8, 6, 10, 9, 5, 6, 7, 20, 11, 17, 6, 9, 10]$$

Question 1: Does H contain a valid *min heap?* If so, extract the minimum value, rearranging H again as a minheap, and then write the resulting content of the array. If not, turn H into a min heap by applying a minimal number of swap operations, and write the resulting content of the array. Justify your answer. *(5)*

Question 2: Write an algorithm MIN-HEAP-ADD(H, x) that adds a new value a min heap H. *(10)*

Question 3: Execute MIN-HEAP-ADD$(H, 4)$ using the algorithm you wrote as a solution to Question 2. In this case, the input H contains the min-heap resulting from your solution to Question 1. Illustrate the execution of MIN-HEAP-ADD$(H, 4)$ by writing the full content of the array H at the beginning of each iteration of the algorithm, as well as at the end of the algorithm. *(5)*

▸**Exercise 5.** Write an algorithm SQUARE-ROOT(n) that given a non-negative integer n returns $\lfloor \sqrt{n} \rfloor$. SQUARE-ROOT(n) may only use the basic arithmetic operations of addition, subtraction, multiplication and division (integer), and must run in $O(\log n)$ time. *(20)*