INSTRUCTIONS: Read all the problems carefully before you start working. The number of points given for a problem are a rough indication of its difficulty or the time it takes to write them down. Start with the simple problems. Most problems can be answered in a few lines of text or equations. Don’t get stuck with the algorithm writing tasks, first try to get an idea how the algorithm works and sketch it for yourself in plain text. Leave the detailed writing of the longer algorithms until the end.

1.) Algorithms and Recursions

The logistic difference equation is defined through the following recurrence:

\[ u_{n+1} = \alpha u_n (1 - u_n) \text{ for } n \geq 1 \]

a) Give a recursive function \( \text{LDE}(n, u_0, \alpha) \) that computes \( u_n \). (1P)

b) Give a recurrence that describes the runtime of \( \text{LDE}(n) \) for \( n \geq 1 \). (1P)

c) Give an non-recursive function \( \text{NEWLDE}(n, u_0, \alpha) \) that also computes \( u_n \) by enumerating \( u_1 \ldots u_n \). Give its runtime in asymptotic notation. (3P)

a) \( \text{LDE}(n, u_0, \alpha) \)
   
   if \( n = 0 \) then
   
   return \( u_0 \)
   
   else
   
   \( v \leftarrow \text{LDE}(n - 1, u_0, \alpha) \)
   
   return \( \alpha \cdot v \cdot (1 - v) \)
end if

b) \( T(i) = T(i - 1) + O(1) \)

c) \text{LDENew}(i)

\[
v \leftarrow u_0
\]

\[
\text{for } i \leftarrow 1 \text{ to } n \text{ do}
\]

\[
v \leftarrow \alpha \cdot v \cdot (1 - v)
\]

\[
\text{end for}
\]

\[
\text{return } v
\]

LDENew runs in \( \Theta(n) \) time.

1P: Iterative Solution solving somehow; 1P: correct algorithm; 1P: run-time
2. Heaps

Given the array $A = (37, 26, 39, 14, 16, 4, 40)$

a) Is this a max heap? Give all violations of the heap property. (1P)

b) What is the content of $A$ after the execution of $\text{BUILDMAXHEAP}(A)$? On what elements $\text{MAXHEAPIFY}$ is called? Which elements are compared or exchanged? Give the sequence of compares and exchanges. (3P)

c) What is the content of $A$ after $\text{HEAPINCREASEKEY}(A, 6, 41)$? Which elements are compared or exchanged? Give the sequence of compares and exchanges. Use the output of part b, not the original content of $A$. (3P)

a) no: 40 larger than 39, 39 larger than 37

0.5P for correct; -0.5 for incorrect answers

b) $40, 26, 39, 14, 16, 4, 37$

Comparisons: $\text{MAX-HEAPIFY} (A[3])$ 39? > 40
39? > 4 - EXCHANGE 40 ↔ 39

$\text{MAX-HEAPIFY} (A[2])$ 26? > 14
26? > 16 - don’t exchange

$\text{MAX-HEAPIFY} (A[1])$ 37? > 26

$\text{MAX-HEAPIFY} (A[3])$ 37? > 4

A = [40, 26, 39, 14, 16, 4, 37]

-0.5P per error

c)

A = [41, 26, 40, 14, 16, 39, 37]


-0.5P per error
3. Sorting

a) Illustrate the operation of RADIXSORT with the following hexadecimal numbers:

9D3,634,295,194,B4,965,2C5,747,303,C

Give a list of the numbers after each step of the algorithm. (3P)

b) Assuming that there are less symbols in the set the digits are chosen from, than there are numbers in the input sequence, what is the asymptotic worst-case runtime of RADIXSORT? Prove your answer. (2P)

c) What is the lower bound for comparison-based sorting? Why is the asymptotic worst-case runtime of $\Theta(n)$ for COUNTINGSORT no contradiction to this? (1P)

a) 

<table>
<thead>
<tr>
<th>start</th>
<th>1</th>
<th>2</th>
<th>end</th>
</tr>
</thead>
<tbody>
<tr>
<td>9D3</td>
<td>9D3</td>
<td>303</td>
<td>C</td>
</tr>
<tr>
<td>634</td>
<td>303</td>
<td>C</td>
<td>B4</td>
</tr>
<tr>
<td>295</td>
<td>303</td>
<td>634</td>
<td>194</td>
</tr>
<tr>
<td>194</td>
<td>194</td>
<td>747</td>
<td>295</td>
</tr>
<tr>
<td>B4</td>
<td>B4</td>
<td>965</td>
<td>2C5</td>
</tr>
<tr>
<td>965</td>
<td>295</td>
<td>194</td>
<td>303</td>
</tr>
<tr>
<td>2C5</td>
<td>965</td>
<td>295</td>
<td>634</td>
</tr>
<tr>
<td>747</td>
<td>2C5</td>
<td>B4</td>
<td>747</td>
</tr>
<tr>
<td>303</td>
<td>747</td>
<td>2C5</td>
<td>965</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
<td>9D3</td>
<td>9D3</td>
</tr>
</tbody>
</table>

-0.5P per error

b) Let $n$ be the number of numbers in the input set and $k$ the number of symbols in the set the digits are chosen from (the alphabet).

The according to Cormen, Page 172, the runtime of RADIXSORT is $\Theta(d(n + k))$ with $d$ being the number of digits in the words. Since $k < n$, we can say that $d(n + k) < d(2n)$

1P for correct Radix runtime, 1P for correct solution

Therefore, the runtime of RADIXSORT is then $O(dn)$.

c) $\Omega(n \log n)$. It is no contradiction since COUNTINGSORT is not using comparisons to sort.
4. Medians

a) Give an algorithm that uses one of the median procedures RandomizedSelect or Select to sort an array $A[1..n]$ into an array $B[1..n]$ starting with the largest element in $B[1]$. (2P)

b) What worst-case runtime has your algorithm? Give a proof. How does this runtime compare to other sorting algorithms, would you use it for sorting? (2P)

a)

```plaintext
MEDIANSORT(A, B)
for i ← 1 to length[A] do
    B[i] ← SELECT(A, 1, length[A], i)
end for
```

1P algorithm; 1P algorithm correct;

b) The algorithm executes SELECT $n$ times with $n = length[A]$. Each call to SELECT takes $O(n)$ time, so the algorithm runs in $n \cdot O(n) = O(n^2)$ time.

The algorithm is worse than all other sorting algorithms, even INSERTIONSORT has the same worst-case runtime but at least sorts in place.

1P: runtime; 1P: comparison

5. Hashing

Given an Array of integers of length 16 i.e. with array indices 0 to 15 and a hash function $h(k, i) = ((k \mod 23) + i) \mod 16$.

Insert the following sequence of numbers into the array, show the contents of the array after each step. Use hashing with open addressing. (4P)

$$180 \ 181 \ 233 \ 406 \ 15 \ 45 \ 1171$$

$h(180, 0) = 3 \ ; A[3] \leftarrow 180$

$h(181, 0) = 4 \ ; A[4] \leftarrow 181$

$h(233, 0) = 3 \ ; h(233, 1) = 4 \ ; h(233, 2) = 5 \ ; A[5] \leftarrow 233$

$h(406, 0) = 15 \ ; A[15] \leftarrow 406$

$h(15, 0) = 3 \ ; h(15, 1) = 0 \ ; A[0] \leftarrow 15$

$h(45, 0) = 6 \ ; A[6] \leftarrow 45$
\( h(1171, 0) = 5 \); \( h(1171, 1) = 6 \); \( h(1171, 2) = 7 \); \( A[7] \leftarrow 1171 \)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>180</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>180</td>
<td>181</td>
<td>233</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>180</td>
<td>181</td>
<td>233</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>180</td>
<td>181</td>
<td>233</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>180</td>
<td>181</td>
<td>233</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>180</td>
<td>181</td>
<td>233</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>180</td>
<td>181</td>
<td>233</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

0.5P per right element; 0.5P bonus for complete solution
6. **MERGE**SORT with Queues

Given the following QM**ERGE**SORT procedure:

```plaintext
QM**ERGE**SORT(Q)
    if ((tail[Q] − head[Q]) mod length[Q] = 1) then
        return
    end if
    create queue QL
    create queue QR
    flag ← 0
    while (head[Q] ≠ tail[Q]) do
        if (flag = 0) then
            ENQUEUE(QL, DEQUEUE(Q))
            flag ← 1
        else
            ENQUEUE(QR, DEQUEUE(Q))
            flag ← 0
        end if
    end while
QM**ERGE**SORT(QL)
QM**ERGE**SORT(QR)
QM**ERGE**(Q, QL, QR)
free queue QL
free queue QR
```

For a complete mergesort algorithm, the **MERGE** procedure is still missing. Unfortunately, we can’t use the merge procedure we used in **MERGE**SORT since this worked on arrays and not on queues. Therefore, you have to write the QM**ERGE** procedure that merges two queues and thus forms a queue-based mergesort algorithm together with QM**ERGE**SORT.

**Hint:** use ENQUEUE(Q, ∞) to enqueue a sentinel symbol into a queue that is always larger than all other numbers.

This function could also be of help, but you may have to modify it.

```plaintext
Q**EMPTY**(Q)
    if (tail[Q] = head[Q]) then
        return true
    else
        return false
    end if
```
QMerge(Q, QL, QR)

\[ l \leftarrow \text{DEQUEUE}(QL) \]
\[ r \leftarrow \text{DEQUEUE}(QR) \]

while (true) do
  if \((l < r)\) then
    ENQUEUE(Q, l)
    if QUEMPTY(QL) then
      while not (QUEMPTY(QR)) do
        ENQUEUE(Q, DEQUEUE(QR))
      end while
      return
    else
      \( l \leftarrow \text{DEQUEUE}(QL) \)
    end if
  else
    ENQUEUE(Q, r)
    if QUEMPTY(QR) then
      while not (QUEMPTY(QL)) do
        ENQUEUE(Q, DEQUEUE(QL))
      end while
      return
    else
      \( r \leftarrow \text{DEQUEUE}(QR) \)
    end if
  end if
end while

explanation 1P; complete, full algorithm 4P (non-cumulative); -0.5P/-1P for errors