1. Asymptotic Notations: (25 P)

(a) Use the formal definitions of $O$, $\Theta$, and $\Omega$ to determine whether the following assertions are true or false.

i. $\frac{n(n+1)}{2} \in O(n^3)$

Solution:

$$\lim_{n \to \infty} \frac{n(n+1)}{2n^3} = \lim_{n \to \infty} \frac{\frac{1}{2} \cdot \frac{1}{n} + \frac{1}{n^2}}{2n} = \lim_{n \to \infty} \frac{\frac{1}{2} + \frac{1}{n^2}}{2} = 0$$

Therefore the statement is true.

ii. $\sqrt{20n^2 + 7n + 3} \in \Theta(n)$

Solution:

$$\lim_{n \to \infty} \frac{\sqrt{20n^2 + 7n + 3}}{n} = \lim_{n \to \infty} \sqrt{\frac{20n^2 + 7n + 3}{n^2}} = \lim_{n \to \infty} \sqrt{20 + \frac{7}{n} + \frac{3}{n^2}} = \sqrt{20} = \text{const}$$

Therefore the statement is true.

iii. $\sqrt{n} \in O(n)$

Solution:

$$\lim_{n \to \infty} \frac{\sqrt{n}}{n} = \lim_{n \to \infty} n^{-2/3} = 0$$

Therefore the statement is true.

(b) Consider the following Algorithm:

Algorithm 1 $\text{Enigma}(A[0..n-1,0..n-1])$

1: {Input: A $n \times n$ matrix $A[0..n-1,0..n-1]$ of real numbers}
2: for $i \leftarrow 0$ to $n-2$
3:   for $j \leftarrow i+1$ to $n-1$
4:     if $A[i,j] \neq A[j,i]$ then
5:       return $\text{false}$
6:   end if
7: end for
8: return $\text{true}$

(a) What does this algorithm compute? (2 P)

Solution:

This program checks if the matrix is symmetric or not.
(b) What is its basic operation? (3 P)

**Solution:**
The basic operation is the comparison of the matrix elements.

(c) How many times is the basic operation executed and what is the efficiency class of this algorithm? (5 P)

**Solution:**
Depending on if the matrix is symmetric or not the number of comparisons differ. In the best case the first comparison shows that the matrix is not symmetric so in the best case we have \( \Theta(1) \) efficiency class. In the worst case we have to perform \( n^2/2 \) comparisons, so in the worst case we have \( \Theta(n^2) \) efficiency class. In average we have to calculate the following:

\[
\frac{1}{n} \sum_{i=0}^{n-1} i = \frac{(n-1)n}{2n} = \frac{n-1}{2} \in \Theta(n)
\]

2. Recurrence Relations: (10 P)

(a) Solve the recurrence relation:

\[
x(n) = x(n-1) + n, \quad \text{for } n > 0, x(0) = 0
\]

**Solution:**

\[
x(n) = x(n-1) + n
\]

\[
x(n) = x(n-2) + (n-1) + n
\]

\[
x(n) = x(n-3) + (n-3) + (n-1) + n
\]

\[
\vdots = \vdots
\]

\[
x(n) = x(0) + 1 + 2 + \cdots + n
\]

\[
x(n) = \frac{1}{2} n(n+1)
\]

(b) Solve the recurrence relation:

\[
x(n) = x(n/2) + n, \quad \text{for } n > 1, x(1) = 1 \quad \text{(Solve for } n = 2^k)\]

**Solution:**

Substitute \( n = 2^k \)

\[
x(n) = x(2^k) + 2^k
\]

\[
x(n) = x(2^{k-2}) + 2^{k-1} + 2^k
\]

\[
x(n) = x(2^{k-3}) + 2^{k-2} + 2^{k-1} + 2^k
\]

\[
\vdots = \vdots
\]

\[
x(n) = x(0) + 2^1 + 2^2 + \cdots + 2^{k-2} + 2^{k-1} + 2^k
\]

\[
x(n) = 1 + 2^1 + 2^2 + \cdots + 2^{k-2} + 2^{k-1} + 2^k
\]

\[
x(n) = \sum_{i=0}^{k} 2^i = 2^{k+1} - 1
\]

back substitution \( 2^k = n \)

\[x(n) = 2n - 1\]
3. Brute Force Algorithms: (12 P)

(a) For the Selection Sort Algorithm given below

Algorithm 2 SelectionSort(A[0..n - 1])

1: {Sorts a given array by selection sort}
2: {Input: An unsorted Array A[0..n - 1] of elements.}
3: {Output: A sorted Array A[0..n - 1] in ascending order.}
4: for i ← 0 to n - 2 do
5:     min ← i
6:     for j ← i + 1 to n - 1 do
8:             min ← j
9:     end if
10:    swap A[i] and A[min]
11: end for
12: end for

(a) Please sort the list A, L, G, O, R, I, T, H, M in alphabetical order using Selection Sort Algorithm given below. (5 P)

Solution:

\[
\begin{align*}
&\text{A L G O R I T H M} \\
&\text{A | L G O R I T H M} \\
&\text{A G | L O R I T H M} \\
&\text{A G H | O R I T L M} \\
&\text{A G H I | R O T L M} \\
&\text{A G H I L | O T R M} \\
&\text{A G H I L M | T R O} \\
&\text{A G H I L M O | R T} \\
&\text{A G H I L M O R | T} \\
\end{align*}
\]

(b) What is the basic operation of the selection sort algorithm? (2 P)

Solution:
The basic operation is comparison of elements.

(c) What is the efficiency class of the selection sort algorithm? (5 P)

The efficiency class is \(\Theta(n^2)\)

4. Divide and Conquer: (35 P)

(a) Using the Master Theorem, find the order of growth of the following recurrences: Recalling the Master Theorem:

Theorem 1 If \(f(n) \in \Theta(n^d)\) with \(d \geq 0\) in the recurrence relation \(T(n) = aT(n/b) + f(n)\), then

\[
T(n) = \begin{cases} 
\Theta(n^d) & \text{if } a < b^d \\
\Theta(n^d \log n) & \text{if } a = b^d \\
\Theta(n^{\log_b a}) & \text{if } a > b^d.
\end{cases}
\]

i. \(T(n) = 4T(n/2) + n, T(1) = 1\). (5 P)

Solution:
According to the master theorem 1 we have \(a = 4, b = 2, d = 1\), so we have the case of \(4 > 2^1\), i.e. \(T(n) = \Theta(n^{\log_2 4}) = \Theta(n^2)\)
ii. \( T(n) = 4T(n/2) + n^3, T(1) = 1. \)  

Solution:
According to the master theorem 1 we have \( a = 4, b = 2, d = 3 \), so we have the case of \( 4 < 2^3 \), i.e. \( T(n) = \Theta(n^3) \)

(b) Apply the mergesort algorithm to sort the list \( A, L, G, O, R, I, T, H, M \) in alphabetical order.  

Solution:

(c) Given is the Quicksort Algorithm as following:
The partition function partitions the subarray using a pivot key, where all elements less than the pivot key are put into the array \( A[l..s-1] \) and all elements greater than the pivot key are put into the array \( A[s+1..r] \). The pivot key will be put as first element of the second array, i.e. \( A[s] \).

i. State the recurrence relation for number of key comparisons for the best case and solve this recurrence relation to get the exact number of key comparisons.
**Algorithm 3 Quicksort(A[l..r])**

1. {Sorts a subarray by quicksort}
2. {Input: An unsorted subarray $A[l..r]$ of $A[0..n-1]$, defined by its left and right indices $l$ and $r$}
3. {Output: Subarray $A[l..r]$ in nondecreasing order.}
4. if $l < r$ then
5. $s \leftarrow Partition(A[l..r])$ \{s is the split position\}
7. Quicksort($A[s..r]$)
8. end if

**Solution:**
In the best case the Quicksort algorithm works like the Mergesort algorithm so its recursion relation is given as:

$$C(n) = 2C(n/2) + n$$

Using the Master Theorem Theorem 1 we can easily verify that $C(n) \in \Theta(n \log n)$.

ii. Describe the worst case scenario and state the number of comparisons in the worst case scenario using the $\Theta$ notation. (5 P)

**Solution:**
In the worst case the pivot key is either the smallest or the biggest element of the list, so that the other elements are either all greater or all smaller. So the we can not really say that we have a partition. In this case the Algorithm works like the selection sort algorithm and its efficiency class is $\Theta(n^2)$.

iii. Sort the list $A, L, G, O, R, I, T, H, M$ using the quicksort algorithm for the partition function using the first element of the subarray as pivot key. (5 P)

A L G O R I T H M
A L G O R I T H M
A G I H L O R T M
A G I H L M O R T
A G H I L M O R T

5. **Binary Trees:**

(a) Draw the binary search tree by inserting the elements of the list $L, O, G, A, R, I, T, H, M$ one by one to the binary search tree. (8 P)

**Solution:**

(b) Perform pre-, in-, and postorder traversal for the binary search tree in question 5a. (7 P)

**pre-order:** L G A I H O M R T
**in-order:** A G H I L M O R T
**post-order:** A H I G M T R O L
6. **Decrease and Conquer:**

Given is the following Insertion Sort Algorithm:

**Algorithm 4 InsertionSort(A[l..r])**

1: {Sorts a array by insertionsort}
2: {Input: An unsorted array A[0..n – 1] of elements.}
3: {Output: A array A[0..n – 1] sorted in nondecreasing order.}
4: for i ← 1 to n – 1 do
5:   v ← A[i]
6:   j ← i – 1
7:   while j ≥ 0 and A[j] > v do
9:     j ← j – 1
10: end while
11: A[j + 1] ← v
12: end for

(a) Use the insertion sort algorithm to sort the array A, L, G, O, R, I, T, H, M.

**Solution:**

```
A L G O R I T H M
A L G O R I T H M
A G L O R I T H M
A G L O R I T H M
A G I L O R I T H M
A G I L O R I T H M
A G I L O R I T H M
A G I L O R I T H M
```

Note: The bold face letters denote the sorted array. We start at the second position as one element in an array is by definition already sorted. Then we go one by one through the array by comparing starting from the biggest element to the smallest element in the sorted array. Once we have found the correct position we insert the element into the correct position.

(b) Describe the worst case scenario for the insertion sort algorithm and state the number of key comparisons using the $\Theta$ notation.

**Solution:**

In the worst case the array has to be sorted in reverse order so we have to carry out $1 + 2 + 3 \cdots + n - 1$ comparisons, i.e. $\Theta(n^2)$ comparisons.

(c) Describe the best case scenario for the insertion sort algorithm and state the number of key comparisons using the $\Theta$ notation.

**Solution:**

In the best case the array is already sorted. Therefore for each step we have to perform exactly one comparison, i.e. in total we have to perform $\Theta(n)$ comparisons for an array with $n$ elements.