DESIGN AND ANALYSIS OF ALGORITHMS
LAB 03: DIVIDE-AND-CONQUER

I. Divide-and-Conquer Paradigm

- Divide the problem into parts.
- Solve each part.
- Combine the Solutions.
- Complexity is usually of form $T(n) = aT(n/b) + f(n)$
- Chapter 5 of the book

II. Sample

Pseudocode of MergeSort algorithm is given as follows:

Input: $A[i], \ldots, A[j]$, where $i \leq j$.
Output: Sorted $A[i], \ldots, A[j]$.

MergeSort(A, i, j)
(* Assumption: $i \leq j$. *)
1. If $i = j$, then return Endif
2. Let $k = \lfloor \frac{i+j-1}{2} \rfloor$.
3. MergeSort(A, i, k)
4. MergeSort$(A, k+1, j)$
5. Merge(A, i, k, j)

End

Merge(A, i, k, j)
(* Assumption: $i \leq k < j$; $A[i:k]$ and $A[k+1:j]$ are sorted. *)
1. $p1 = i$; $p2 = k+1$; $p3 = i$
2. While $p1 \leq k$ and $p2 \leq j$ {
   2.1 If $A[p1] \leq A[p2]$, then $B[p3] = A[p1]$, $p1 = p1 + 1$
   Else $B[p3] = A[p2]$, $p2 = p2 + 1$; Endif
   2.2 $p3 = p3 + 1$;
   }
3. (* Copy the remaining elements into B *)
   While $p1 \leq k$ { $B[p3] = A[p1]$; $p1 = p1 + 1$; $p3 = p3 + 1$; }
   While $p2 \leq j$ { $B[p3] = A[p2]$; $p2 = p2 + 1$; $p3 = p3 + 1$; }
4. For $r = i$ to j { $A[r] = B[r]$ }

End

The MergeSort algorithm is analyzed as follows

Complexity of Merge: $O(j - i + 1)$, that is the size of the two parts to be merged.

Complexity of MergeSort:
$T(n) \leq T(\lceil \frac{n}{2} \rceil) + T(\lfloor \frac{n}{2} \rfloor) + cn.$
Gives $T(n) \in O(n \log n)$.
– For n being power of 2:
$T(n) \leq 2T(n/2) + cn.$
$T(n) \leq 4T(n/4) + 2(cn/2) + cn = 4T(n/4) + 2cn.$
$T(n) \leq 8T(n/8) + 4(cn/4) + 2cn = 8T(n/8) + 3cn.$
...
$T(n) \in O(n \log n)$

Can also be done using Master Theorem, for n being a power of 2:
$T(n) \leq 2T(n/2) + cn.$
Thus, by Master Theorem
$k = 1$, $b = 2$, $a = 2$, and $2 = 2^1$
Thus, $T(n) \in O(n^1 \log n)$

Implementation of MergeSort in Python is presented as follows

```
import timeit
from random import shuffle

def mergesort(A, i, j):
    if i == j: return None
    k = (i + j)//2
    mergesort(A,i,k)
    mergesort(A,k+1,j)
    merge(A, i, k, j)

def merge(A,i,k,j):
    B = [0]*len(A)
    p1, p2, p3 = i, k+1, i
    while p1 <= k and p2 <= j:
        if A[p1] < A[p2]:
            B[p3] = A[p1]
            p1 += 1
        else:
            B[p3] = A[p2]
            p2 += 1
        p3 += 1
    while p1 <= k:
        B[p3] = A[p1]
        p3 += 1
        p1 += 1
    while p2 <= j:
```

```
        B[p3] = A[p2]
        p3 += 1
        p2 += 1
    for r in range(i, j+1):
        A[r] = B[r]

x = list(range(400))
shuffle(x)
print(x)
start = timeit.default_timer()
mergesort(x,0, len(x) - 1)
stop = timeit.default_timer()
print(x)
print('Time: ', stop - start)
```

III. Exercises

1. Design, implement and analyze a recursive algorithm with divide-and-conquer approach to calculate the number $a^n (a, n \geq 0)$

$$a^n = \begin{cases} 1 \; if \;\; n = 0 \\ a * a * \ldots * a \; if \; n > 0 \end{cases}$$

A(n) = A(n-1) * a

2. Design, implement and analyze a recursive algorithm with divide-and-conquer approach to calculate the number x $(a, n \geq 0)$

$$x = \begin{cases} a \; if \;\; n = 0 \\ a * (a+1) * \ldots * (a+n) \; if \; n > 0 \end{cases}$$

X(n) = X(n-1) * (a + n)????

3. Design, implement and analyze a recursive algorithm with divide-and-conquer approach for:

   a. Calculating Height of a binary tree

   b. Traversing a binary tree in Pre-order, Post-order, In-order.

Node = (value, Left, Right)

Node = (1, None, None)

Node0 = (1, Node, None)

4. Implement and analyze the worst case of **QuickSort** algorithm using Python programming language. Then compare the program with the presented above MergeSort program on random datasets as large as possible.

5. Design, analyze and implement an algorithm in the divide and conquer approach to find an element in an ascending sorted array.