# Algorithms Laboratory (CS29203)
# Lab Test - 1
# Department of CSE, IIT Kharagpur

**15th September 2022**

---

## Question-1

**(30 points):** In this question, you have to implement a slightly modified version of quicksort to sort an array in ascending order. The partitioning scheme discussed in the class is not very efficient when the array is already sorted or when the array has all equal elements. So we would like to modify the partitioning routine of quicksort to handle this problem. The new partitioning scheme uses two indices that start at the ends of the array being partitioned, then move towards each other until the following situation arises: one of the elements which is to the left of the pivot is greater then the pivot, and the other which is to the right of the pivot is less than the pivot. These elements are then swapped. When the indices meet, the algorithm stops. Think carefully what you will need to do with the pivot after the said swap. This type of partitioning scheme is more efficient than standard partitioning scheme discussed in the class, because it does fewer swaps on average, and it creates efficient partitions even when all values are equal.

Note that in this scheme, the next two segments that the main algorithm recurs on are [low ... pivot] and [pivot+1 ... high] (as opposed to [low ... pivot-1] and [pivot+1 ... high] in the standard scheme).

Your task is to implement the quicksort algorithm using the above mentioned partitioning strategy. Choose the *first* element as pivot. Think carefully what will be the segment [low ... pivot] when the first element is pivot.

Example:

```
Enter the number of elements: 8
Enter the elements: 6, 5, 3, 1, 8, 7, 2, 4
---------------
Sorted sequence: 1, 2, 3, 4, 5, 6, 7, 8
```

## Question-2

Near the end of the semester you realize that you have $n$ number of pending assignments. Suppose you have $M$ hours left to solve the assignments, and the assignments will take $m_1, \cdots, m_n$ hours of effort for solving (assume $M$ and each $m_i$ are integers). Also, suppose that the maximum points for these assignments are $p_1, \cdots, p_n$ (all of these are integers). There is no part marking, so you have to solve an assignment totally, or leave it.

Your primary goal is to solve the assignments such that the total points of the solved assignments is maximized. You want to preserve energy for the end-semester tests. So your secondary goal is to attempt the maximum achievable total points with the minimum possible effort.

**(a) (30 points):** Assume that all points are equal, that is, $p_1 = p_2 = \cdots = p_n$. Implement a greedy strategy to obtain maximum points with minimum effort. Print this optimal solution.

Example:

```
Enter the number of assignments (n): 10
Enter the total number of hours left (M): 100
Enter the number of hours needed to solve each assignment: 23, 23, 28, 28, 13, 5, 21, 25, 31, 5
---------------
Minimum effort needed (by greedy strategy) = 90 (23 + 23 + 13 + 5 + 21 + 5)
```

(Note that the order of the hour values may not necessarily correspond to the order in which the greedy strategy chooses the assignments)

**(b) (40 points):** Now, assume that the points array $p_1, \cdots, p_n$ contains arbitrary integer values. There need not be any correlation between the efforts and the points. Greedy strategies do not work now. Dynamic programming helps you instead. Let the total points in all the $n$ assignments be

$$P = p_1 + p_2 + \cdots + p_n$$

Your task is to solve the problem using a Dynamic Programming approach. Following is an ouline of building the DP table that might be useful to you.

Build a two-dimensional table T of size $(n+1) \times (P+1)$ such that $T[i][p]$ is intended to store the minimum possible effort to attempt **exactly** $p$ points from the assignments $1, 2, \cdots, i$. If exactly $p$ points cannot be achieved from $p_1, p_2 \cdots, p_i$, you should store $T[i][p] = \infty$.

First, notice that $T[i][0] = 0$ for all $i$ (irrespective of the number of assignments, if you do not solve any assignment, the attempted point is 0). Another boundary condition pertains to $i = 0$, which means that no assignment is considered. Therefore, $T[0][0] = 0$, whereas $T[0][p] = \infty$ whenever $p > 0$.

Now, take $1 \leq i \leq n$ and $1 \leq p \leq P$. Let $M_0$ and $M_1$ denote two options: not solving, and solving the $i$-th assignment. If $p < p_i$, you cannot attempt the $i$-th assignment, so $T[i][p] = T[i-1][p]$. If $p \geq p_i$ you have two possibilities: if you do not solve the $i$-th assignment, then take $M_0 = T[i-1][p]$, whereas you solve the $i$-th assignment, then take $M_1 = m_i + T[i-1][p-p_i]$. If $M_1 > M$, the second option is not valid, so set $T[i][p] = M_0$, otherwise set $T[i][p] = \min(M_0, M_1)$.

Write a program to build the table T using the approach just mentioned. The maximum achievable point is the largest $p$ such that $T[n][p] \neq \infty$. To achieve this $p$, the minimum effort needed is $T[n][p]$.

Example:

```
Enter the number of assignments (n): 10
Enter the total number of hours left (M): 100
Enter the points of each assignment:  8, 2, 7, 4, 5, 3, 8, 2, 2, 10
Enter the number of hours needed to solve each assignment: 15, 9, 22, 25, 25, 30, 26, 4, 29, 39
---------------
Maximum points = 30 (8 + 7 + 5 + 8 + 2)
Minimum effort = 92 (15 + 22 + 25 + 26 + 4)
```