1.1. The optimal storage tape:

In this problem, the number of programs is \( N \), the tape length is \( L \), the length of the program \( L P_j = P_j \), the programs will be stored in the sequence \( P_1, P_2, ..., P_N \), to retrieve the program, initially putting the tape at its start.

The formulation: if the programs stored in the order \( P_1, P_2, ..., P_j, ..., P_N \), the time \( T_j \) to retrieve the program \( P_j \) will be as follow:

\[
T_j \propto \sum_{k=1}^{j} L P_k
\]

If the retrieving probability is equal for all programs, the Mean of Retrieval Time will be

\[
MRT = \frac{1}{N} \sum_{j=1}^{N} T_j
\]

\[
MRT = \frac{1}{N} \sum_{j=1}^{N} \sum_{k=1}^{j} L P_k
\]

We need a suitable order to store the programs, where it will minimize the MRT.

The minimization of MRT is equivalent to minimization \( \sum_{j=1}^{N} \sum_{k=1}^{j} L P_k \)

Minimize \( \sum_{j=1}^{N} \sum_{k=1}^{j} L P_k \)

Subjected to \( \sum_{k=1}^{N} L P_k \leq L \)

By using the greedy method, we need to store the programs in ascending order according to its length.

Example: find the optimal storage for 3 programs on one magnetic tape where the length of the programs are 5, 10, and 3 respectively.

Solution:

\( N= 3 \), \( (L P_1, ..., L P_3) = (5, 10, 3) \)

\( N! = 3! = 6 \)

We can calculate the retrieval time of the all as follow

\( D = N * MRT \)
The order  

<table>
<thead>
<tr>
<th>Order</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1, 2, 3</td>
<td>5 + (5 + 10) + (5 + 10 + 3) = 38</td>
</tr>
<tr>
<td>1, 3, 2</td>
<td>5 + (5 + 3) + (5 + 3 + 10) = 31</td>
</tr>
<tr>
<td>2, 1, 3</td>
<td>10 + (10 + 5) + (10 + 5 + 3) = 43</td>
</tr>
<tr>
<td>2, 3, 1</td>
<td>10 + (10 + 3) + (10 + 3 + 5) = 41</td>
</tr>
<tr>
<td>3, 1, 2</td>
<td>3 + (3 + 5) + (3 + 5 + 10) = 29</td>
</tr>
<tr>
<td>3, 2, 1</td>
<td>3 + (3 + 10) + (3 + 10 + 5) = 34</td>
</tr>
</tbody>
</table>

The optimal solution when we sort the lengths in ascending order

**Note:** we can generalize the optimal storage for many programs on many magnetic tapes, and the storing will be after sorting the programs according to its lengths.

The storing the programs on the tapes by depending on

\[ \text{Tape}_k = P_j \mod M, \quad 1 \leq j \leq N \]

\[ LP_1 \leq LP_2 \leq \ldots \leq LP_N \]

Where \( \text{Tape}_k \): the index of the tape that will be in the range 0..M-1.

\( P_j \): the program location.

\( M \): the number of the tapes.

**Example:** find the optimal storage for 6 programs on two magnetic tapes where the lengths of the programs are 3, 5, 7, 9, 11, 13 respectively.

Solution:

<table>
<thead>
<tr>
<th>T₀</th>
<th>T₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>13</td>
<td>11</td>
</tr>
</tbody>
</table>

**Homework:** find the optimal storage for 13 programs on three magnetic tapes where the lengths of the programs are 12, 5, 8, 32, 7, 5, 18, 26, 4, 3, 11, 10, 6 respectively.
1.2. Optimal merge patterns:

By comparing the element i with the element j and the smaller will put in the output array and shift the index k and shift either i or j depending on the smaller.

The merge two sorted files contains on N and M records respectively, to get one sorted file, it needs $O(N+M)$ because it needs $M+N$ of moves. To merge more than two files, we can repeat the process on each two files.

**Greedy Rule:** to minimize moving of the records we merge the two files with less size at each step.

**Example:** find the optimal merge patterns for 5 files that have the following lengths: 20, 30, 10, 5, 30, and then find the overall number of record moves for the binary merge tree.
$D_i$: the distance between the external point of file $F_i$ and the root.

$Q_i$: the length of the file $F_i$.

the overall number of record moves for the binary merge tree $= \sum_{i=1}^{N} D_i Q_i$

$= 3 \times (5 + 10) + 2 \times (20 + 30 + 30) = 205$ moves.

Homework: Find the optimal merge patterns for 10 files that have the following lengths: 28, 32, 12, 5, 84, 53, 91, 35, 3, 11, and then find the overall number of record moves for the binary merge tree.

1.3. Single – source shortest paths (Dijkstra Algorithm):

A motorist wishes to find the shortest possible route from Babylon to Basra. Given a road map of the Iraq on which the distance between each pair of adjacent cities is marked, how can we determine this shortest route?

One possible way is to enumerate all the routes from Babylon to Basra, add up the distances on each route, and select the shortest. We will use the greedy approach to develop a $O(n^2)$ algorithm for this problem (called the Single Source Shortest Paths problem).

we show how to solve such problems efficiently. In a shortest-paths problem, we are given a weighted, directed graph $G = (V, E)$, with weight function $w : E \rightarrow \mathbb{R}$ mapping edges to real-valued-weights. The weight of path $p = v_1, v_2, \ldots, v_k$ is the sum of the weights of its constituent edges:

$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$

We define the shortest-path weight from $u$ to $v$ by

$$\delta(u, v) = \begin{cases} 
\min \{w(p) : u \xrightarrow[p]{p} v\} & \text{if there is a path from } u \text{ to } v, \\
\infty & \text{otherwise}. 
\end{cases}$$
A **shortest path** from vertex \( u \) to vertex \( v \) is then defined as any path \( p \) with weight \( w(p) = \delta(u, v) \).

In the Babylon-to-Basra example, we can model the road map as a graph: vertices represent cities, edges represent road segments between cities, and edge weights represent road distances. Our goal is to find a shortest path from a given city in Babylon to a given city in Basra.

Edge weights can be interpreted as metrics other than distances. They are often used to represent time, cost, penalties, loss, or any other quantity that accumulates linearly along a path and that one wishes to minimize.

we shall focus on the **single-source shortest-paths problem**: given a graph \( G = (V, E) \), where \( V \): the set of vertices, and \( E \):the set of edges. we want to find a shortest path from a given **source** vertex \( s \in V \) to each other vertex \( v \in V \ where \ v \neq s \).

The following algorithm explains the greedy approach for generating shortest paths (Dijkstra Algorithm).

---

**Algorithm**

```
Procedure ShortestPath(v: integer; var cost: Matrix; var Dist:ElemList; n: integer);
var u, I, w, num:integer;
S: ElemListBoolean;
Begin
For i:= 1 to n do
begin
S[i] := false;
dist[i] := Cost [v, i];
end;
S[v] := true; dist[v] := 0;
For num := 2 to n do
begin
u:= Select ( dist, Cost, S, n);
S[u] := true;
For w:= 1 to n do
If  (( S[w] = false)  and  (dist[w] > dist[u] + Cost[u, w])) then
dist[w] := dist[u] + Cost[u, w];
end;
```

---

The following algorithm explains the greedy approach for generating shortest paths (Dijkstra Algorithm).

```
Procedure ShortestPath(v: integer; var cost: Matrix; var Dist:ElemList; n: integer);
var u, I, w, num:integer;
S: ElemListBoolean;
Begin
For i:= 1 to n do
begin
S[i] := false;
dist[i] := Cost [v, i];
end;
S[v] := true; dist[v] := 0;
For num := 2 to n do
begin
u:= Select ( dist, Cost, S, n);
S[u] := true;
For w:= 1 to n do
If  (( S[w] = false)  and  (dist[w] > dist[u] + Cost[u, w])) then
dist[w] := dist[u] + Cost[u, w];
end;
```
End;

Function Select(var dist: ElemList; Cost: Matrix; S: ElemListBoolean; n: integer):integer;
var min, j: integer;
beging
    min := 1;
    while ( S[min] <> false) min := min + 1;
    for j := min +1 to n do
        if (( S[j] = false) and (dist[j] < dist[min] )) then
            min := j;
    Select := min;
End;

Where n: the number of nodes in the graph. Cost: the cost array that contains the cost between each two nodes in the graph (its size is n * n). If there is a link between the two nodes it will take a cost, if there is no link it will take $\infty$ ($\infty = \text{no. of edges} \times n$). The cost from the node to itself equal 0.

Algorithm Analysis: It needs $O(n^2)$ of time complexity.

Example: Find the shortest paths with its lengths from node 1 to other nodes in the graph below using Dijkstra algorithm.
Homeworks:

1- Expand the Dijkstra algorithm to generate the shortest paths with its lengths.

2- Find the shortest paths with its lengths from node 1 to other nodes in the graph below using Dijkstra algorithm.