# Problem szeregowania

## Algorytm Johnsona dla 2 oraz 3 maszyn

In [2]:
```
import numpy as np
```

In [3]:
```
def johnson_2(matrix, with_order=False):
    matrix_copy = matrix.copy().astype("float")
    _, k = matrix.shape
    result = np.zeros(matrix.shape)

    start = 0
    end = k - 1
    if with_order:
        order = {i: 0 for i in range(k)}

    # Szereguje zadania
    while matrix_copy.min() < np.inf and start <= end:
        # Wyznaczam najmniejszy element w tablicy
        arg = np.argmin(matrix_copy)
        # Przypadek jezeli najmniejszy element jest w pierwszym rzedzie
        if arg < k:
            i = arg
            result[:,start] = matrix_copy[:,i]
            if with_order:
                order[i] = start

            start += 1
        # Przypadek jezeli najmniejszy element jest w drugim rzedzie
        else:
            i = arg % k
            result[:,end] = matrix_copy[:,i]
            if with_order:
                order[i] = end

            end -= 1

        # Blokuje kolumny
        matrix_copy[:,i] = np.inf

    # Wyznaczenie czasow
    T = np.zeros(matrix.shape)
    time = 0
    for i in range(k):
        time += result[0,i]
        T[0,i] = time

    T[1,0] = T[0,0] + result[1,0]
    for i in range(1, k):
        T[1,i] = max(T[1,i-1], T[0, i]) + result[1,i]

    # Rozwiazanie z mapowaniem starych kolumn na nowe
    if with_order:
        return result, T, order
    return result, T
```

In [4]:
```
matrix = np.array([
    [8, 9, 2, 8, 0, 3, 2, 1, 8, 4, 9, 6],
    [3, 7, 3, 6, 4, 1, 3, 1, 2, 9, 5, 3],
])
```

```
result, T = johnson_2(matrix)
print(f"Poczatkowe ulozenie zadan: \n{matrix}")
print(f"Koncowe ulozenie zadan: \n{result}")
print(f"Czasy: \n{T}")
```

```
Poczatkowe ulozenie zadan:
[[8 9 2 8 0 3 2 1 8 4 9 6]
 [3 7 3 6 4 1 3 1 2 9 5 3]]
Koncowe ulozenie zadan:
[[0. 1. 2. 2. 4. 9. 8. 9. 6. 8. 8. 3.]
 [4. 1. 3. 3. 9. 7. 6. 5. 3. 3. 2. 1.]]
Czasy:
[[ 0.  1.  3.  5.  9. 18. 26. 35. 41. 49. 57. 60.]
 [ 4.  5.  8. 11. 20. 27. 33. 40. 44. 52. 59. 61.]]
```

In [1]:
```
def johnson_3(matrix):
    _, k = matrix.shape
    mod_matrix = np.zeros((2, k))
    # Tworze tablice z dwoma rzedami
    for j in range(k):
        mod_matrix[0,j] = matrix[0,j] + matrix[1,j]
        mod_matrix[1,j] = matrix[1,j] + matrix[2,j]

    _, T, order = johnson_2(mod_matrix, with_order=True)
    result = np.zeros((3, k))

    # Szereguje kolumny wedlug mapy ktorej otrzymalem z johnsona z dwoma maszynami
    for key, value in order.items():
        result[:,value] = matrix[:,key]
    return result, T
```

In [7]:
```
matrix = np.array([
    [8, 9, 2, 8, 0, 3, 2, 1, 8, 4, 9, 6],
    [3, 4, 8, 2, 5, 2, 8, 8, 6, 1, 5, 3],
    [2, 1, 3, 6, 4, 1, 3, 2, 2, 9, 4, 0],
])
result, time = johnson_3(matrix)
print(f"Poczatkowe ulozenie zadan: \n{matrix}")
print(f"Koncowe ulozenie zadan: \n{result}")
print(f"Czasy: \n{time}")
```

```
Poczatkowe ulozenie zadan:
[[8 9 2 8 0 3 2 1 8 4 9 6]
 [3 4 8 2 5 2 8 8 6 1 5 3]
 [2 1 3 6 4 1 3 2 2 9 4 0]]
Koncowe ulozenie zadan:
[[0. 4. 1. 2. 2. 9. 8. 8. 9. 8. 6. 3.]
 [5. 1. 8. 8. 8. 5. 6. 2. 4. 3. 3. 2.]
 [4. 9. 2. 3. 3. 4. 2. 6. 1. 2. 0. 1.]]
Czasy:
[[  5.  10.  19.  29.  39.  53.  67.  77.  90. 101. 110. 115.]
 [ 14.  24.  34.  45.  56.  65.  75.  85.  95. 106. 113. 118.]]
```

# Zadanie 3

### Jaki typ problemu rozwiazujemy (klasyfikacja Grahama) ?

Rozwiazujemy problem otwarty dla dwoch oraz trzech maszyn.

### Jakie waruneki sa konieczne do spelnienia algorytmu?

Aby algorytm byl poprawny musza byc spelniony jeden z dwoch warunkow

$$\min_j(t_{1j}) \geq \max_j(t_{2j})$$

lub

$$\min_j(t_{3j}) \geq \max_j(t_{2j})$$

### Jaka jest zlozonosc obliczeniowa algorytmu ?

Zlozonosc obliczeniowa algorytmu dla trzech maszyn wynosi $O(\frac{n^2+n}{2} + n)$. Gdzie n to liczba zadan.