Algorithm-8

—— FlowShop

A. Problem Description

The scheduling problem, under consideration, is called flow-shop scheduling where given a set of parts to be processed (jobs) and a set of machines for processing. Each part has the same technological path on all machines; the order of jobs is arbitrary. The goal is to find the appropriate sequence of jobs that minimizes the sum of idle times.

B. Description of algorithm

$\min\{b_{\pi(i)}, a_{\pi(j)}\} \ge \min\{b_{\pi(j)}, a_{\pi(i)}\}$ JOHNSON'S ALGORITHM:

If any two jobs in the π schedule met the JOHNSON'S ALGORITHM, then the total time must be the minimum.

```
FlowShop()
[i]     N1 = { i |    }   & $a_i \gtrless b_j$ N2 = { i |  }
[ii]        Sort the set N1 in non-descending order
[iii]       Sort the set N2 in non-ascending order
[iiii]  Concatenate the sets N1 & N2
```

C. *Time Complexit*

Step [i]: → O(n)

Step [ii] & [iii]: → O(nlogn) if using MergeSort-Algorithm

Step [iiii]: → O(n)

Therefore, the total time

T=O(nlogn)

D. *Code[Python]*

```
#!/usr/bin/python
# Filename: FlowShop.py

class JobType:
  def judgeJob(self, x, y):
    self.job = (x <= y)
    self.key = x if x <= y else y
    return self.job
  def judgeIndex(self, index):
    self.index = index

def sort(array, n):
  for i in range(0, n - 1):
    k = i
    for j in range(i + 1, n):
      if array[k].key > array[j].key:
        k = j
    temp = array[k]
    array[k] = array[i]
    array[i] = temp


def FlowShop(n, a, b, c):
  d = []
  temp = JobType()
```

```
for i in range(0, n):
  d.append(JobType())
  d[i].judgeJob(a[i], b[i])
  d[i].index = i

sort(d, n)

'''
for i in range(0, n):
  print d[i].job, d[i].key, d[i].index
'''

j = 0
k = n - 1
for i in range(0, n):
  if d[i].job:
    c[j] = d[i].index
    j += 1
  else:
    c[k] = d[i].index
    k -= 1

j = a[c[0]]
k = j + b[c[0]]
for i in range(1, n):
  j += a[c[i]]
  k = (k + b[c[i]]) if j < k else (j + b[c[i]])

return k
```