**B0521229 林威廷**

**平行程式設計-Training3**

**Part1: Analysis(In Theory)**

where we assume

n:numbers of elements

m:numbers of processors

1. Sequential:

(1).Distribute numbers to bucket: O(*n*)

(2).Sequential sort each bucket: (n/m)log(n/m) x m

Overall: O(n log(n/m))

2.Parallelize sorting: one process per bucket:

(1).Distribute numbers to bucket: O(*n*)

(2).Sequential sort each bucket: (n/m)log(n/m)

Overall: O(n + n/m log(n/m))

Overall:A single process must scan through all numbers in step1

3. Further Parallelized Bucket Sort

(1). Partition numbers to ***m*** parts/processes

(2). Each process divides its numbers to small buckets

(3). Merge small buckets to large bucket

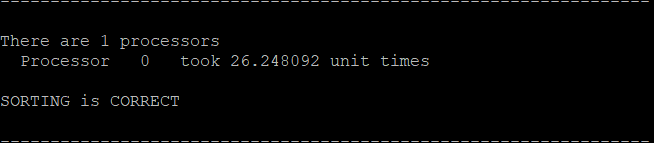
(4). Sequential sort each bucket

**Overall: O(n/m + n/m log(n/m))**

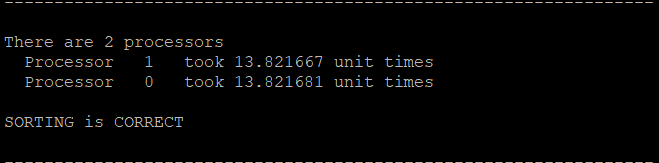
I do the No.3 method

**Part2: Implementation of Results**

1 processor(Basic)



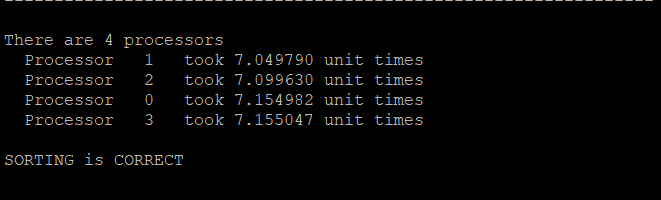
2 processors:



speed up:

26.248/13.821=1.89

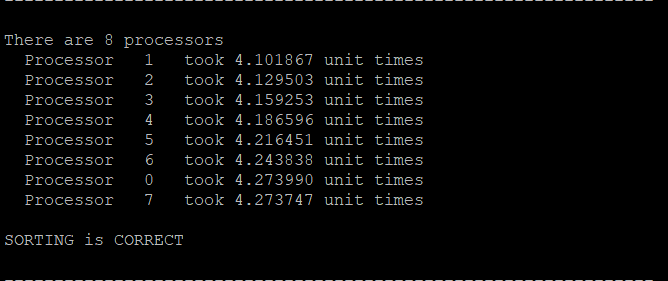
4 processors:



speed up:

26.248/7.155=3.67

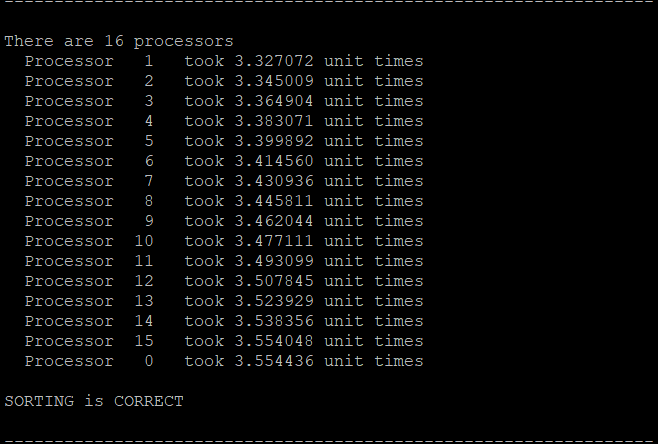
8 processors:



speed up:

26.248/4.273=6.14

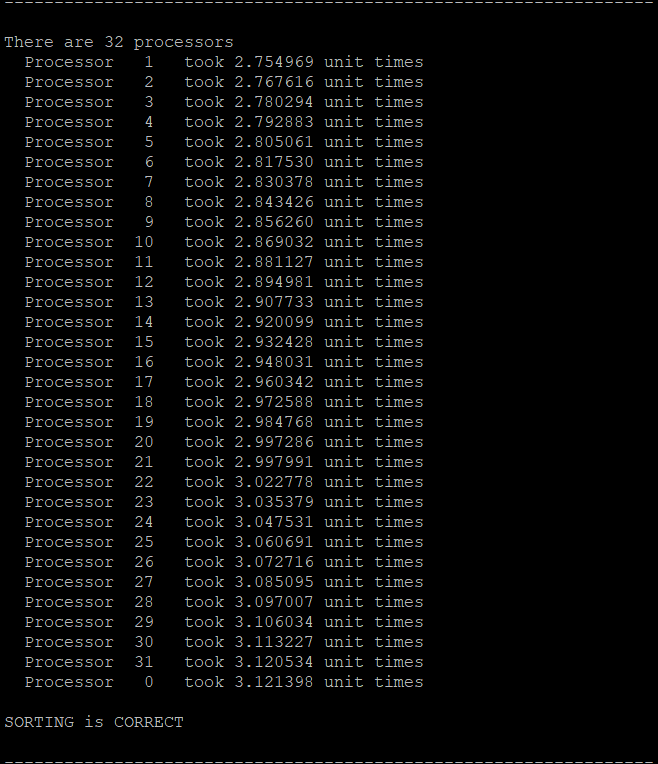
16 processors:



speed up:

26.248/3.554=7.38

32 processors:(extra test)



speed up:

26.248/3.121=8.41

**心得**

本次作業的實作，結果顯現2顆processors在此size(1億)和數字隨機分布範圍(0~99999)下之bucket sort 平行版本之speed up是最接近optimal的(2倍)，結果顯現越多顆processors的加速效果反而被抑制了，我自己的觀點為：受限於呼叫Collective Communication Call,太多顆processors反而拉長了Communication time，導致加速效果大幅下跌，而偏離optimal越來越遠。