

# **BÁO CÁO THỰC HÀNH**

## **Nhóm 14**

### **THIẾT KẾ THUẬT TOÁN SONG SONG**

*GVHD: ThS. Nguyễn Thanh Sơn*

*Lớp: CS112.N21.KHTN*

*Thành viên 1: Võ Minh Quân – 21520093*

*Thành viên 2: Nguyễn Việt Nhật – 21520378*

*Ngày thực hiện: 13/03/2023*

## 1. Giới thiệu chung:

### a) Yêu cầu bài toán:

Hãy lựa chọn CTDL và thiết kế thuật toán song song để thực hiện sắp xếp dãy tăng dần theo thuật toán merge sort.

### b) Source code:

- Lập trình tuần tự: [serial.py](#)
- Lập trình song song: [parallel.py](#)

## 2. Cấu trúc dữ liệu:

a) Array: Tập hợp các phần tử cố định được lưu trữ liên tiếp nhau trong các ô nhớ. Kiểu phần tử có thể là có các kiểu bất kỳ: ký tự, số, chuỗi ký tự => Lưu dãy cần sắp xếp và các mảng con của nó; danh sách các tiến trình.

b) Queue (multiprocessing): Cho phép chia sẻ dữ liệu giữa các tiến trình đang thực thi song song với nhau, bao gồm hàm put() và get() nhằm thêm và lấy dữ liệu ra khỏi queue. Được thiết kế dựa trên hàng đợi FIFO (first-in, first-out).

## 3. Thuật toán:

### a) Merge sort:

Merge sort là thuật toán chia để trị. Từ input là một dãy, giải thuật chia dãy thành hai nửa và mỗi nửa sau đó được áp dụng cùng chiến lược chia cho đến khi dãy chỉ còn 1 hay 2 phần tử.

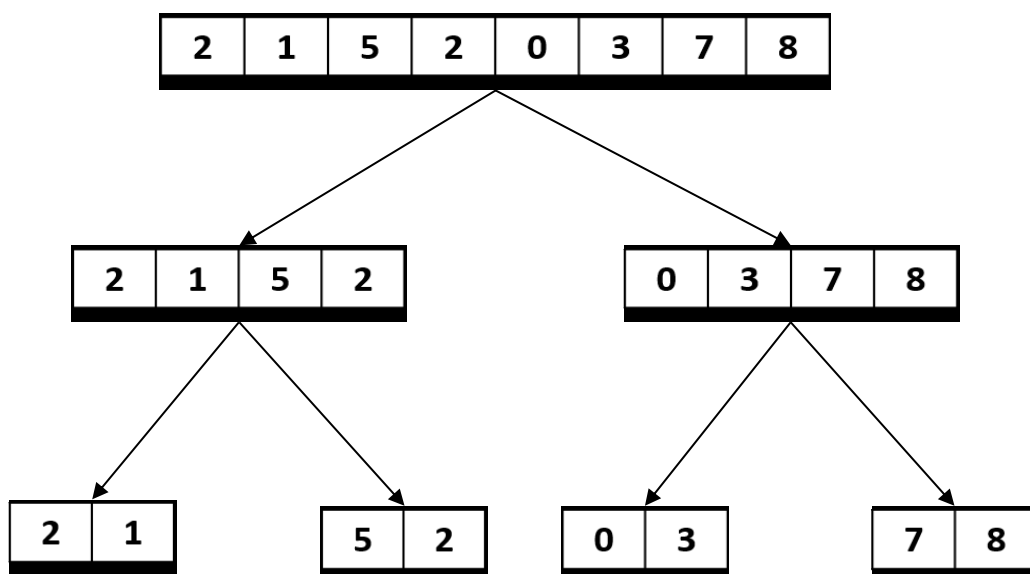


Figure 1: ví dụ về merge sort (1)

Các dãy con kề nhau sau đó được trộn với nhau thành các dãy con được sắp xếp cho đến thu được một dãy đã được sắp xếp (output bài toán).

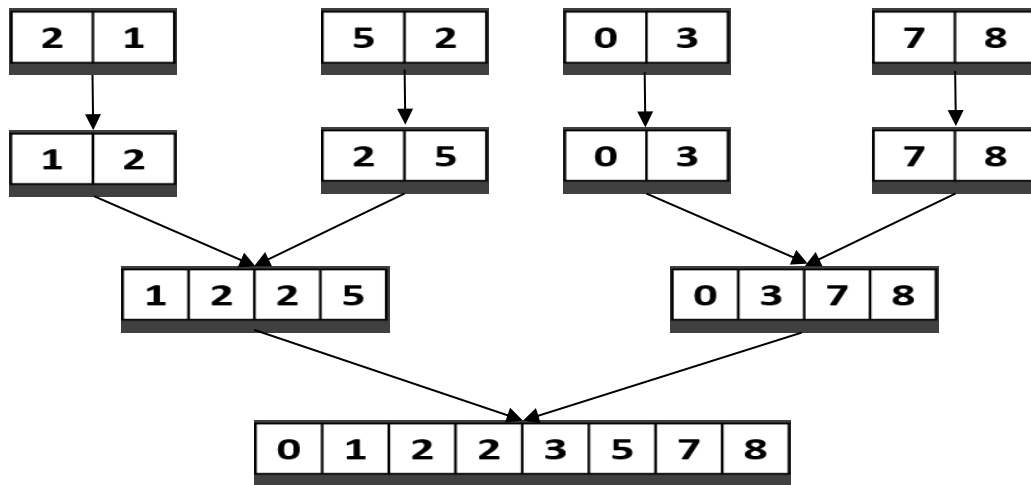


Figure 2: Ví dụ về merge sort (2)

- Pseudo code:

```

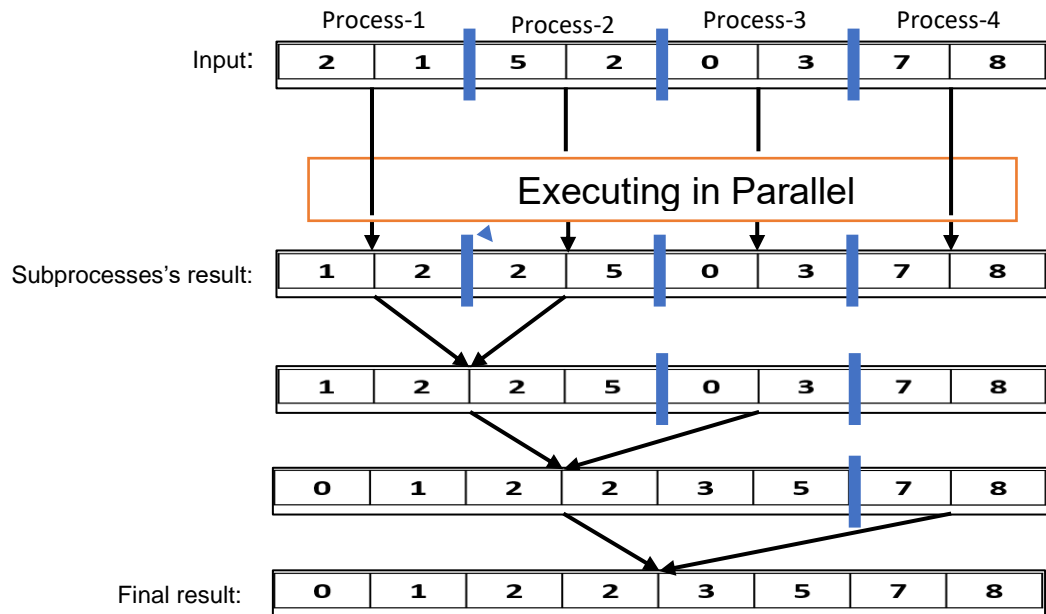
def merge_sort(arr, low, high):
    if low >= high:
        return

    mid = (low + high) // 2
    merge_sort(arr, low, mid)
    merge_sort(arr, mid+1, high)
    merge(arr, low, mid, high)
end
  
```

b) Parallel merge sort:

Vì tính chất chia để trị của merge sort, ta dễ dàng nhận thấy được mỗi dãy con được tách ra thành một bài toán con. Bên cạnh đó, các bài toán con này không yêu cầu phải thực thi theo thứ tự.

Do đó, ta chia bài toán ban đầu thành các bài toán con và thực thi chúng song song với nhau nhằm lấy được các kết quả độc lập (của từng bài toán con). Sau đó, trộn các kết quả lấy được để có được kết quả cuối cùng.



B1: Nhập dãy gồm  $n$  phần tử (input).

B2: Chia dãy ban đầu thành  $N$  dãy con bằng nhau ( $N$  là số lượng process).

B3: Khởi tạo  $N$  process, với mỗi process thực thi merge sort trên từng dãy con tương ứng.

B4: Sau khi  $N$  process hoàn thành, trộn kết quả thu được từ  $N$  process. Dãy sau khi trộn là output cần tìm.

- Pseudo-code:

```
def parallel_merge_sort(arr, low, high):
    Split array into NUM_PROCESS subarrays
    # each subarray contains about n // NUM_PROCESS elements)

    Create NUM_PROCESS processes (each handles one subarray)

    for process in processes:
        process.start()

    for process in processes:
        process.join()

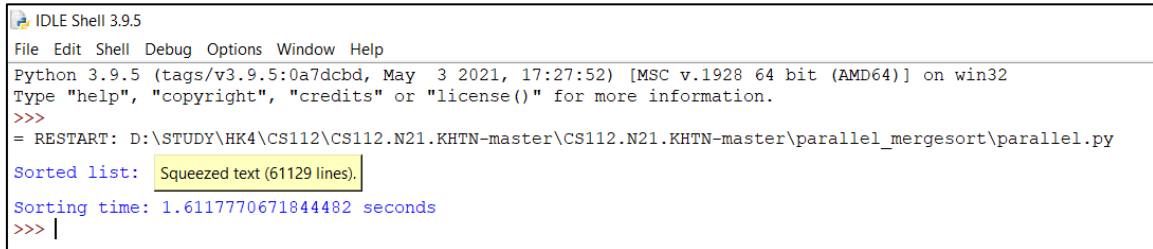
    # Q is array of multiprocessing Queue
    # Each q in Q is a return data of each processes)
    for q in Q:
        result = merge(result, q)

    return result
end
```

Độ phức tạp: Time complexity  $O(n \log n / \text{NUM\_PROCESS})$ , Space complexity  $O(n)$

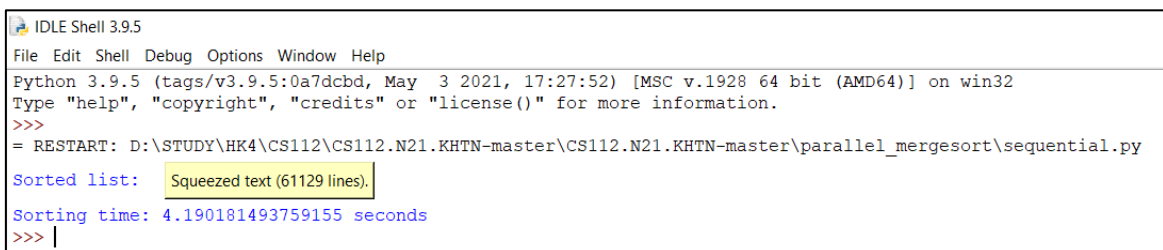
#### 4. Thực nghiệm:

##### a) Test case $N = 1e6$ :



```
IDLE Shell 3.9.5
File Edit Shell Debug Options Window Help
Python 3.9.5 (tags/v3.9.5:0a7dcdbd, May 3 2021, 17:27:52) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: D:\STUDY\HK4\CS112\CS112.N21.KHTN-master\CS112.N21.KHTN-master\parallel_mergesort\parallel.py
Sorted list: Squeezed text (61129 lines).
Sorting time: 1.6117770671844482 seconds
>>> |
```

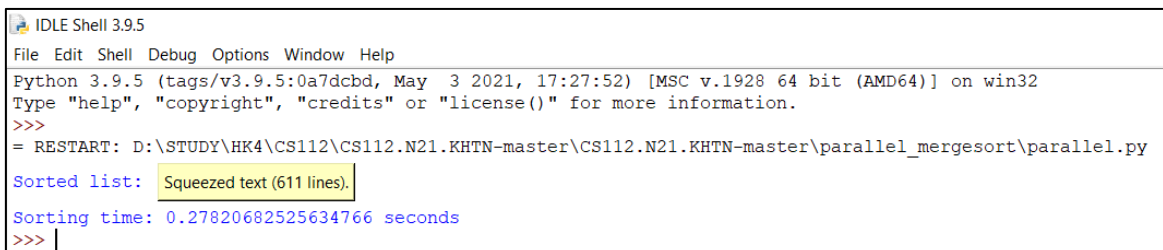
Figure 4: Thời gian chạy parallel 3.a)



```
IDLE Shell 3.9.5
File Edit Shell Debug Options Window Help
Python 3.9.5 (tags/v3.9.5:0a7dcdbd, May 3 2021, 17:27:52) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: D:\STUDY\HK4\CS112\CS112.N21.KHTN-master\CS112.N21.KHTN-master\parallel_mergesort\sequential.py
Sorted list: Squeezed text (61129 lines).
Sorting time: 4.190181493759155 seconds
>>> |
```

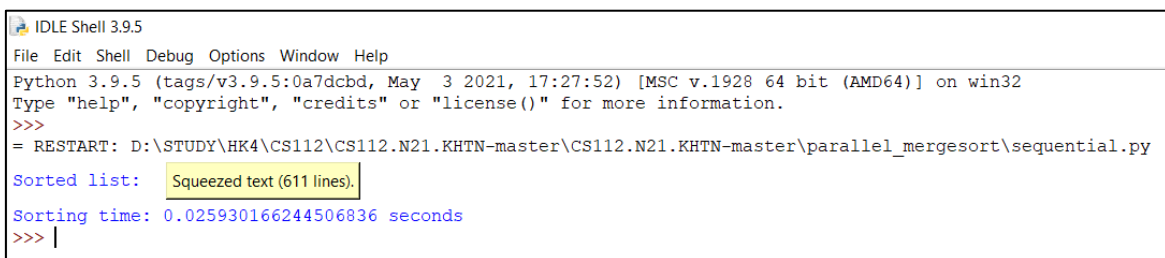
Figure 5: Thời gian chạy sequential (serial) 3.a)

##### b) Test case $N = 1e4$



```
IDLE Shell 3.9.5
File Edit Shell Debug Options Window Help
Python 3.9.5 (tags/v3.9.5:0a7dcdbd, May 3 2021, 17:27:52) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: D:\STUDY\HK4\CS112\CS112.N21.KHTN-master\CS112.N21.KHTN-master\parallel_mergesort\parallel.py
Sorted list: Squeezed text (611 lines).
Sorting time: 0.27820682525634766 seconds
>>> |
```

Figure 6: Thời gian chạy parallel 3.b)



```
IDLE Shell 3.9.5
File Edit Shell Debug Options Window Help
Python 3.9.5 (tags/v3.9.5:0a7dcdbd, May 3 2021, 17:27:52) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: D:\STUDY\HK4\CS112\CS112.N21.KHTN-master\CS112.N21.KHTN-master\parallel_mergesort\sequential.py
Sorted list: Squeezed text (611 lines).
Sorting time: 0.025930166244506836 seconds
>>> |
```

Figure 7: Thời gian chạy sequential (serial) 3.b)

#### 5. Kết luận:

Trong thuật toán trên, nhóm em quyết định thực thi thuật toán song song với multiprocessing trên python. Song, vẫn có thể sử dụng multithreads nhằm lập trình song song. Qua thực nghiệm, lập trình song song với multiprocessing có ưu điểm như sau:

1. Xử lý các bài toán nhanh hơn. Cho phép giải quyết các vấn đề lớn hơn trong một khoảng thời gian ngắn.
2. Tận dụng được tối đa tài nguyên của máy tính.

Song, lập trình song song cũng có những bất lợi cần phải chú ý:

1. Tiêu tốn tài nguyên và năng lượng bởi kiến trúc multi-core.
2. Khó triển khai, sửa lỗi. Do đó dẫn tới khó khăn trong việc chứng minh tính đúng đắn của thuật toán.
3. Thường cho ra kết quả chậm hơn so với tuần tự (với dữ liệu đầu vào vừa và nhỏ) bởi hệ điều hành phải chịu thêm overhead trong khởi tạo các process và chi phí phát sinh cho việc truyền tải dữ liệu, đồng bộ hóa hay giao tiếp giữa các tiến trình.