Name: NGUYỄN ANH TÀI - ID: 20520924

Name: LÝ KIỀU CHÍ - ID: 20521131

Class: IT007.M22.1

OPERATING SYSTEM
LAB 04'S REPORT

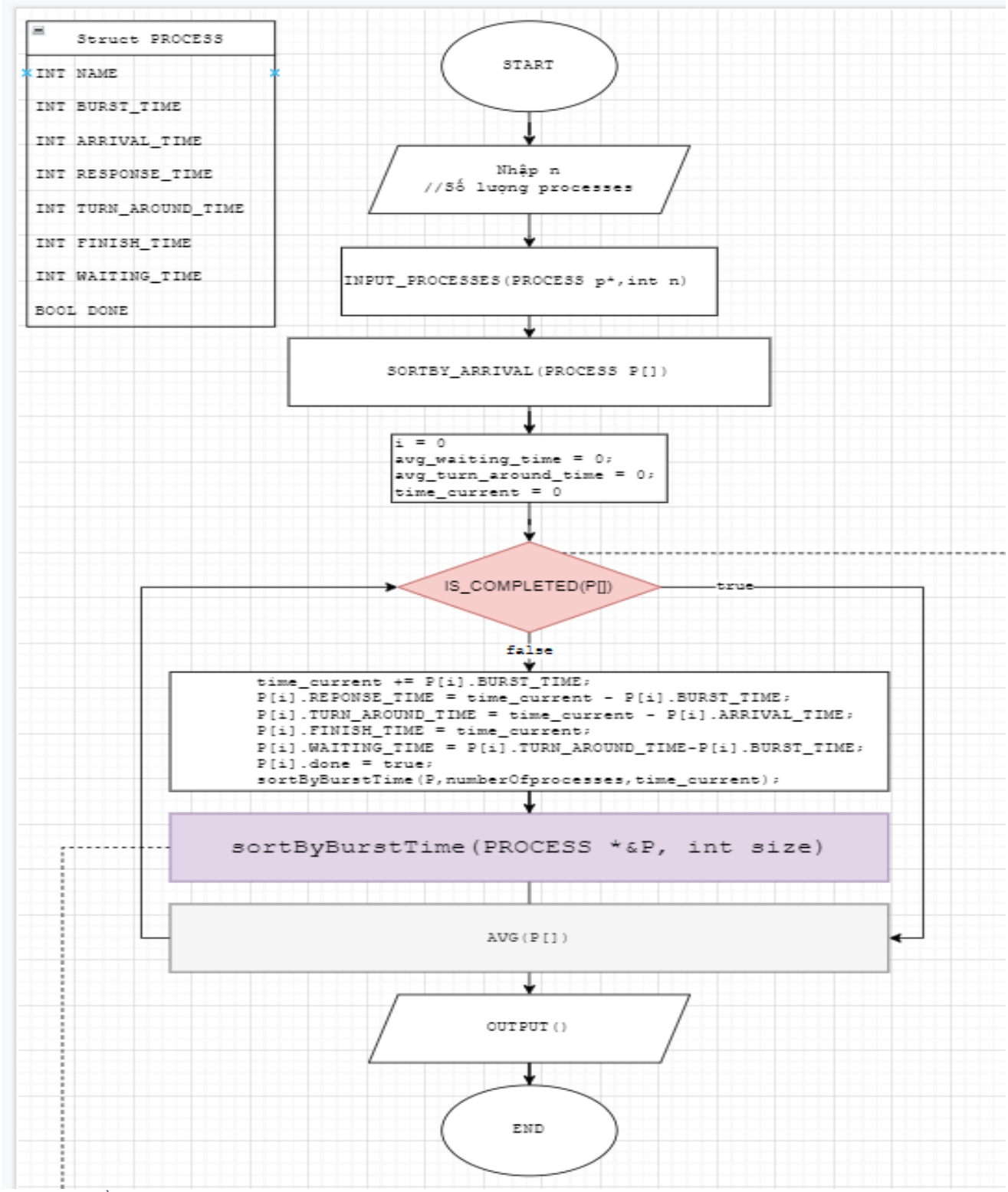
SUMMARY

Task	Status	Page
1. Giải thuật SJF: Vẽ lưu đồ giải thuật -Trình bày tính đúng đắn của lưu đồ bằng cách chạy tay ít nhất 01 test case tối thiểu 05 tiến trình -Thực hiện code cho giải thuật -Trình bày tính đúng đắn của code bằng cách chạy ít nhất 03 test case, mỗi test case 5 tiến trình	DONE	
2. giải thuật SRTF: -Vẽ lưu đồ giải thuật -Trình bày tính đúng đắn của lưu đồ bằng cách chạy tay ít nhất 01 test case tối thiểu 05 tiến trình -Thực hiện code cho giải thuật -Trình bày tính đúng đắn của code bằng cách chạy ít nhất 03 test case, mỗi test case 5 tiến trình	DONE	
3. (Bonus) Thực hiện các yêu cầu trên với giải thuật còn lại.	UNDONE	

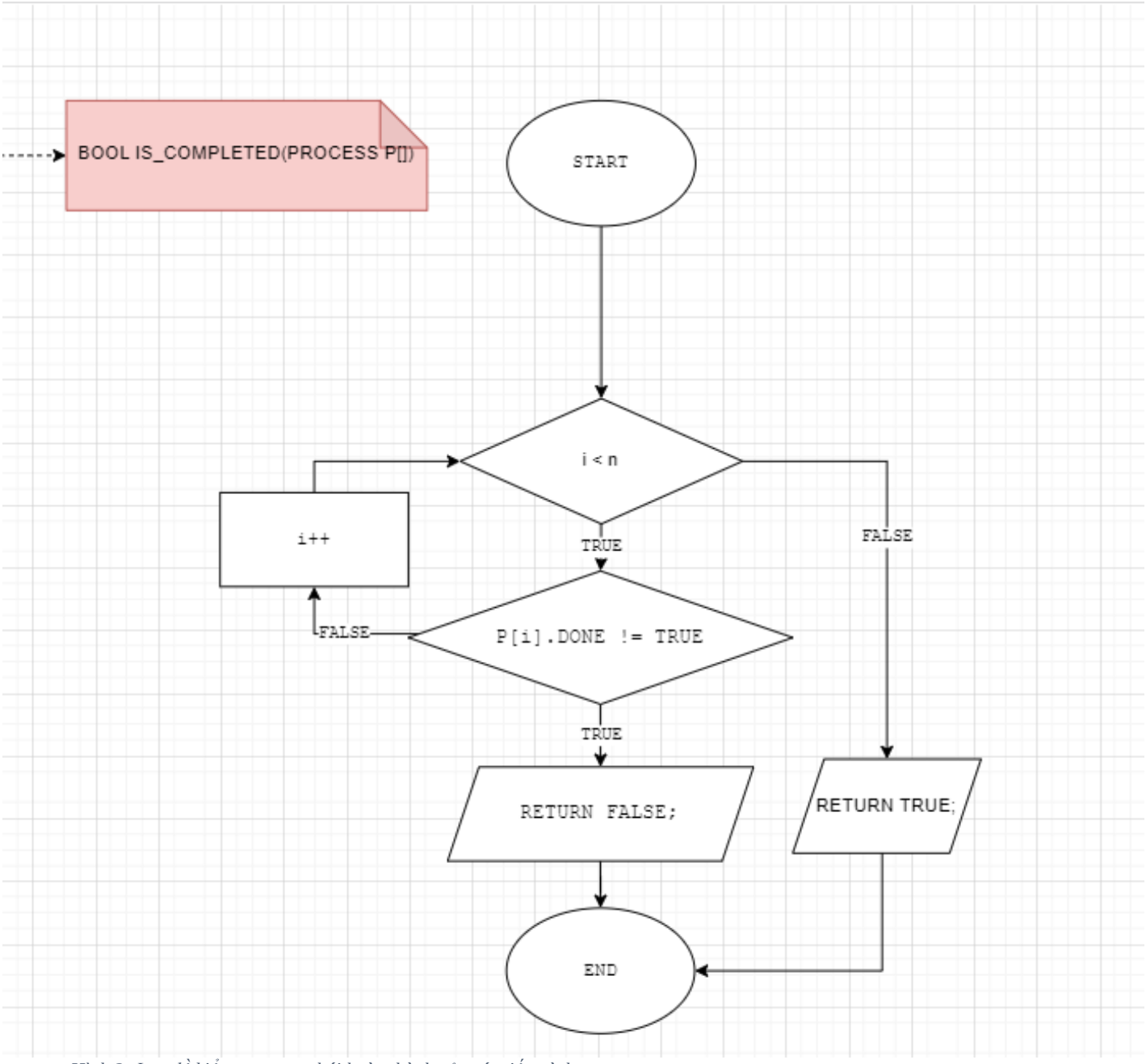
Self-scrores: 8

TASK 01: GIẢI THUẬT SJF

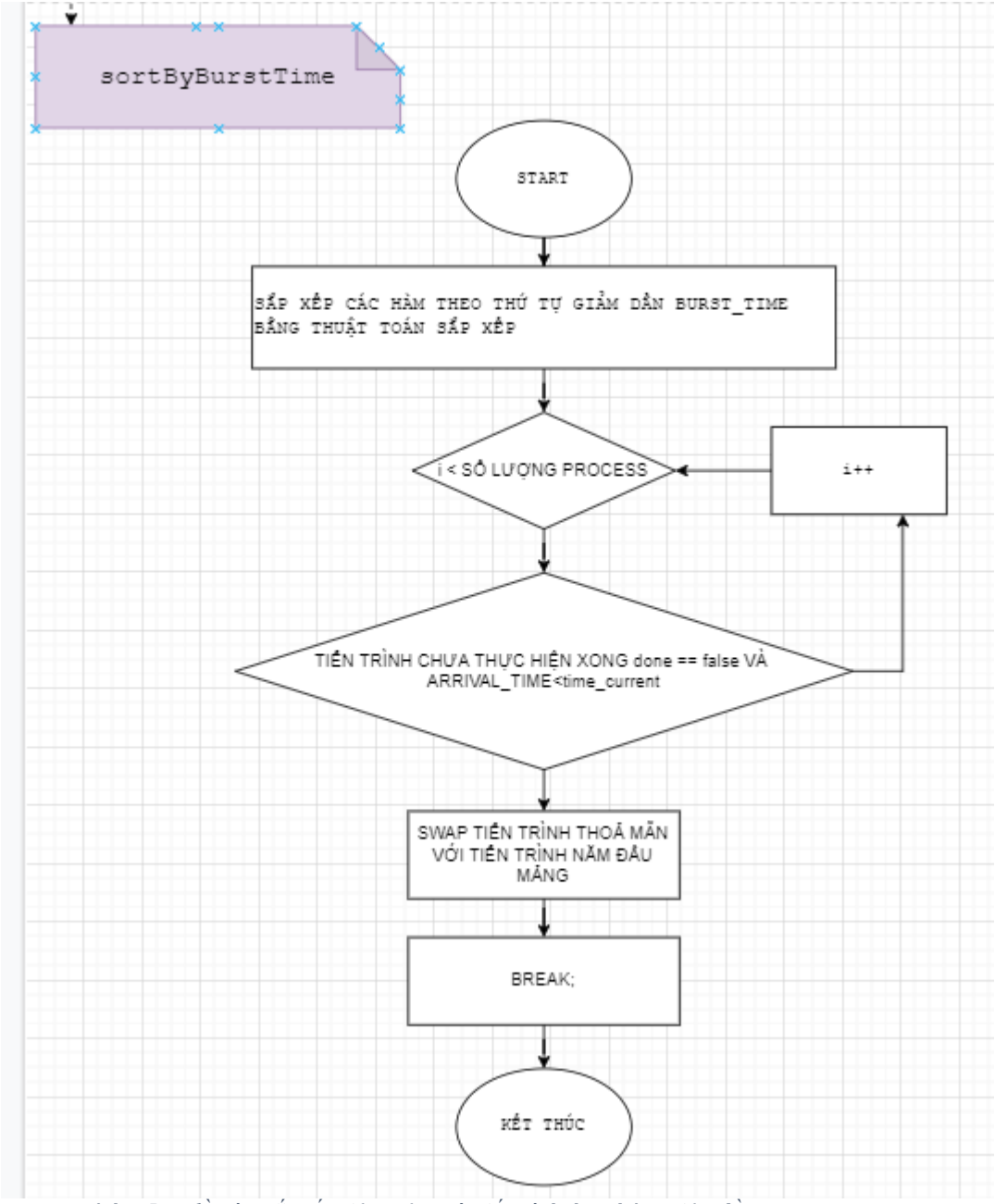
1. Lưu đồ giải thuật.



Hình 1 Lưu đồ thuật toán SJF



Hình 2: Lưu đồ kiểm tra trạng thái hoàn thành của các tiến trình



Hình 3:Lưu đồ Hàm sắp xếp giảm mảng các tiến trình theo thứ tự giảm dần

2. Code giải thuật.

```
#include <iostream>
#include <string>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <vector>
```

```cpp
using namespace std;

struct PROCESS
{
    string NAME;
    int BURST_TIME;
    int ARRIVAL_TIME;
    int REPONSE_TIME;
    int TURN_AROUND_TIME;
    int FINISH_TIME;
    int WAITING_TIME;
    bool done = false;
};

double avg_waiting_time = 0;
double avg_turn_around_time = 0;


void Add_Process(int &size, PROCESS *&P, PROCESS process)
{
    PROCESS *newArr = new PROCESS[size + 1];
    for (int index = 0; index < size; index++)
    {
        newArr[index] = P[index];
    }
    newArr[size] = process;
    size++;
    P = newArr;
}

void swap(PROCESS &p1, PROCESS &p2)
{
    PROCESS tmp;
    tmp = p1;
    p1 = p2;
    p2 = tmp;
}

void sortByArrivalTime(PROCESS *&P, int size)
{
    for (int i = 0; i < size; i++)
    {
        for (int j = i + 1; j < size; j++)
        {
            if (P[i].ARRIVAL_TIME > P[j].ARRIVAL_TIME)
```

```
            {
                swap(P[i], P[j]);
            }
        }
    }
}

void sortByBurstTime(PROCESS *&P, int n, int time_current)
{
    for (int i = 0; i < n; i++)
    {
        for (int j = i + 1; j < n; j++)
        {
            if (P[i].BURST_TIME > P[j].BURST_TIME)
            {
                swap(P[i], P[j]);
            }
        }
    }
       for (int i = 0; i < n; i++)
        {
            if (P[i].done == false &&P[i].ARRIVAL_TIME<time_current)
            {
                swap(P[i], P[0]);
                break;
            }
        }
}

void Input_Process(PROCESS *&P, int &size)
{
    cout << "Input the number of Processes: ";
    cin >> size;

    PROCESS *newArr = new PROCESS[size];

    for (int i = 0; i < size; i++)
    {
        cin.ignore(32767, '\n');
        cout << "----------------[" << i << "]-----------------\n";
        cout << "ARRIVAL_TIME: ";
        cin >> newArr[i].ARRIVAL_TIME;

        cout << "BUSRT_TIME: ";
        cin >> newArr[i].BURST_TIME;
```

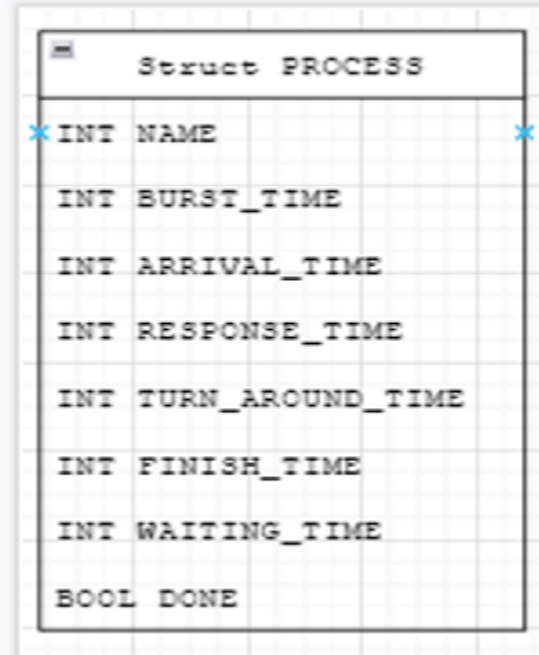
```
        newArr[i].NAME = (char)('A' + i);

    }
    P = newArr;
}

string space(int n)
{
    string res;
    for (int i = 0; i < n; i++)
    {
        res += " ";
    }
    return res;
}

void Output_Process(int numberOfprocesses, PROCESS *P)
{
    sortByArrivalTime(P, numberOfprocesses);
    string Attribute[6] = {"Job", "Arrival_Time", "Burst_Time", "Finish_Time",
"Turnaround_Time", "Waiting_Time"};

    int num_columns = sizeof(Attribute) / sizeof(Attribute[0]);
    vector<vector<string>> board;
    vector<string> row {"\tSHORTEST JOB FIRST\n" } ;
    board.push_back(row);

    for (int i = 0; i < numberOfprocesses + 1; i++)
    {
        vector<string> row;
        if (i == 0)
        {
            for (int j = 0; j < num_columns; j++)
            {
                if (j == 0)
                    row.push_back("|" + Attribute[0] + "|");
                else
                    row.push_back(Attribute[j] + "|");
            }
            board.push_back(row);
        }
        else
        {
            int ele = i - 1;
```

```
            string Value[6] = {P[ele].NAME, to_string(P[ele].ARRIVAL_TIME),
to_string(P[ele].BURST_TIME), to_string(P[ele].FINISH_TIME),
to_string(P[ele].TURN_AROUND_TIME), to_string(P[ele].WAITING_TIME)};
            for (int j = 0; j < num_columns; j++)
            {
                if (j == 0)
                    row.push_back("|" + space(Attribute[0].length() - Value[0].length())
+ Value[0] + "|");
                else
                    row.push_back(space(Attribute[j].length() - Value[j].length()) +
Value[j] + "|");
            }
            board.push_back(row);

        }

    }
       row = {"Average Turn Around Time: " +to_string(avg_turn_around_time)+"\n",
                        "Average Turn Waiting Time: "
 +to_string(avg_waiting_time)+"\n"  } ;
        board.push_back(row);


    for (int i = 0; i < numberOfprocesses + 3; i++)
    {
        for (int j = 0; j < num_columns; j++)
        {
            if (i==numberOfprocesses+2)
            {
                cout << board[i][0].c_str();
                cout << board[i][1].c_str();
                break;
            }
            if (i==0)
            {
                cout << board[i][0].c_str();
                break;
            }
            cout << board[i][j].c_str();
        }
        cout << "\n";
    }
}

bool isCompleted(PROCESS *P, int Size)
```

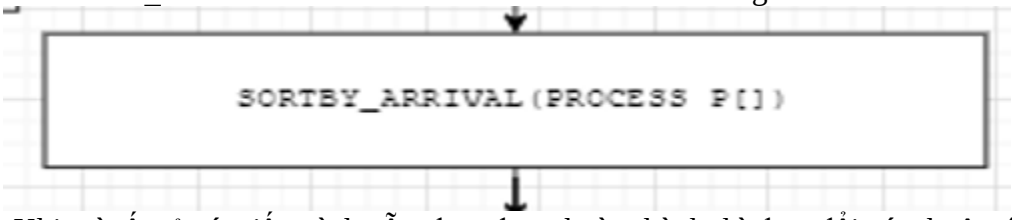
```
{
    for (int i = 0; i < Size; i++)
    {
        if (!P[i].done)
        {
            return false;
        }
    }
    return true;
}

void AVG(PROCESS *P,int Size){
    for (int i = 0; i < Size; i++)
    {
        avg_turn_around_time += (double)P[i].TURN_AROUND_TIME/Size;
        avg_waiting_time += (double)P[i].WAITING_TIME/Size;
    }
}


int main()
{
    int numberOfprocesses = 1;
    PROCESS *P = new PROCESS[numberOfprocesses];
    Input_Process(P, numberOfprocesses);
    sortByArrivalTime(P, numberOfprocesses);
    int i = 0;
    int time_current = 0;
    while (!isCompleted(P, numberOfprocesses))
    {
        time_current += P[i].BURST_TIME;
        P[i].REPONSE_TIME = time_current - P[i].BURST_TIME;
        P[i].TURN_AROUND_TIME = time_current - P[i].ARRIVAL_TIME;
        P[i].FINISH_TIME = time_current;
        P[i].WAITING_TIME = P[i].TURN_AROUND_TIME-P[i].BURST_TIME;
        P[i].done = true;
        sortByBurstTime(P,numberOfprocesses,time_current);
    }
    AVG(P,numberOfprocesses);
    Output_Process(numberOfprocesses, P);

    return 0;
}
```

3. DIỄN GIẢI.

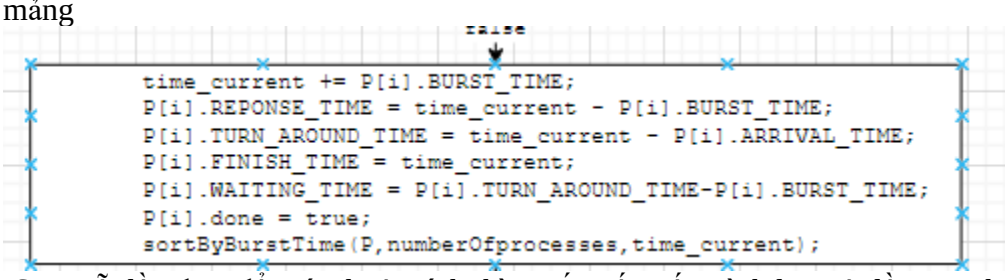
- Đầu tiên tạo kiểu dữ liệu tiến trình với các thuộc tính theo Lưu đồ trên.



-Sau khi nhập dữ liệu cho mảng các tiến trình thì ta sắp tiến trình theo thứ tự giảm dần theo ARRIVAL_TIME đưa tiến trình tới sớm nhất lên đầu mảng.



-Khi mà tất cả các tiến trình vẫn chưa được hoàn thành thì thay đổi các thuộc tính của tiến trình nằm đầu mảng



-Sau mỗi lần thay đổi các thuộc tính thì ta sắp xếp tiến trình lại một lần nữa theo thứ tự giảm dần về BURST_TIME theo một điều kiện đặc biệt nhằm Tìm ra tiến trình có BURST_TIME ngắn nhất và đưa nó lên đầu mảng. Chi tiết của hàm sortByBurstTime xem tại *Hình 3*

-Cứ như thế sau mỗi lần lặp thì tiến trình có BurstTime Nhỏ nhất sẽ được đưa lên đầu mảng và thực thi cho đến khi tất cả các tiến trình vào trạng thái hoàn thành.

4. Tính đúng đắn của giải thuật và code.

Output SJF

Gantt Chart

A	D	C	B	E
0	12	18	27	41	56

Job	Arrival Time	Burst Time	Finish Time	Turnaround Time	Waiting Time
A	0	12	12	12	0
B	3	14	41	38	24
D	5	6	18	13	7
C	7	9	27	20	11
E	14	15	56	42	27
			Average	125 / 5 = 25	69 / 5 = 13.8

```
        SHORTEST JOB FIRST

|Job|Arrival_Time|Burst_Time|Finish_Time|Turnaround_Time|Waiting_Time|
|  A|           0|        12|         12|             12|           0|
|  B|           3|        14|         41|             38|          24|
|  C|           5|         6|         18|             13|           7|
|  D|           7|         9|         27|             20|          11|
|  E|          14|        15|         56|             42|          27|
Average Turn Around Time: 25.000000
Average Turn Waiting Time: 13.800000
```

Hình 4: Test Case 01 Chạy Bằng Tay Và Kết quả code

Output

Gantt Chart

A	D	E	B	C
0	12	15	21	28	36

Job	Arrival Time	Burst Time	Finish Time	Turnaround Time	
A	0	12	12	12	
B	2	7	28	26	
C	5	8	36	31	
D	9	3	15	6	
E	12	6	21	9	3
			Average	84 / 5 = 16.8	48 / 5 = 9.6

```
        SHORTEST JOB FIRST

|Job|Arrival_Time|Burst_Time|Finish_Time|Turnaround_Time|Waiting_Time|
|  A|           0|        12|         12|             12|           0|
|  B|           2|         7|         28|             26|          19|
|  C|           5|         8|         36|             31|          23|
|  D|           9|         3|         15|              6|           3|
|  E|          12|         6|         21|              9|           3|
Average Turn Around Time: 16.800000
Average Turn Waiting Time: 9.600000
```

Hình 5: Test Case 02 Chạy Bằng Tay Và Kết quả code

Output SJF

Gantt Chart

A	C	D	E	B
0	8	11	17	29	48

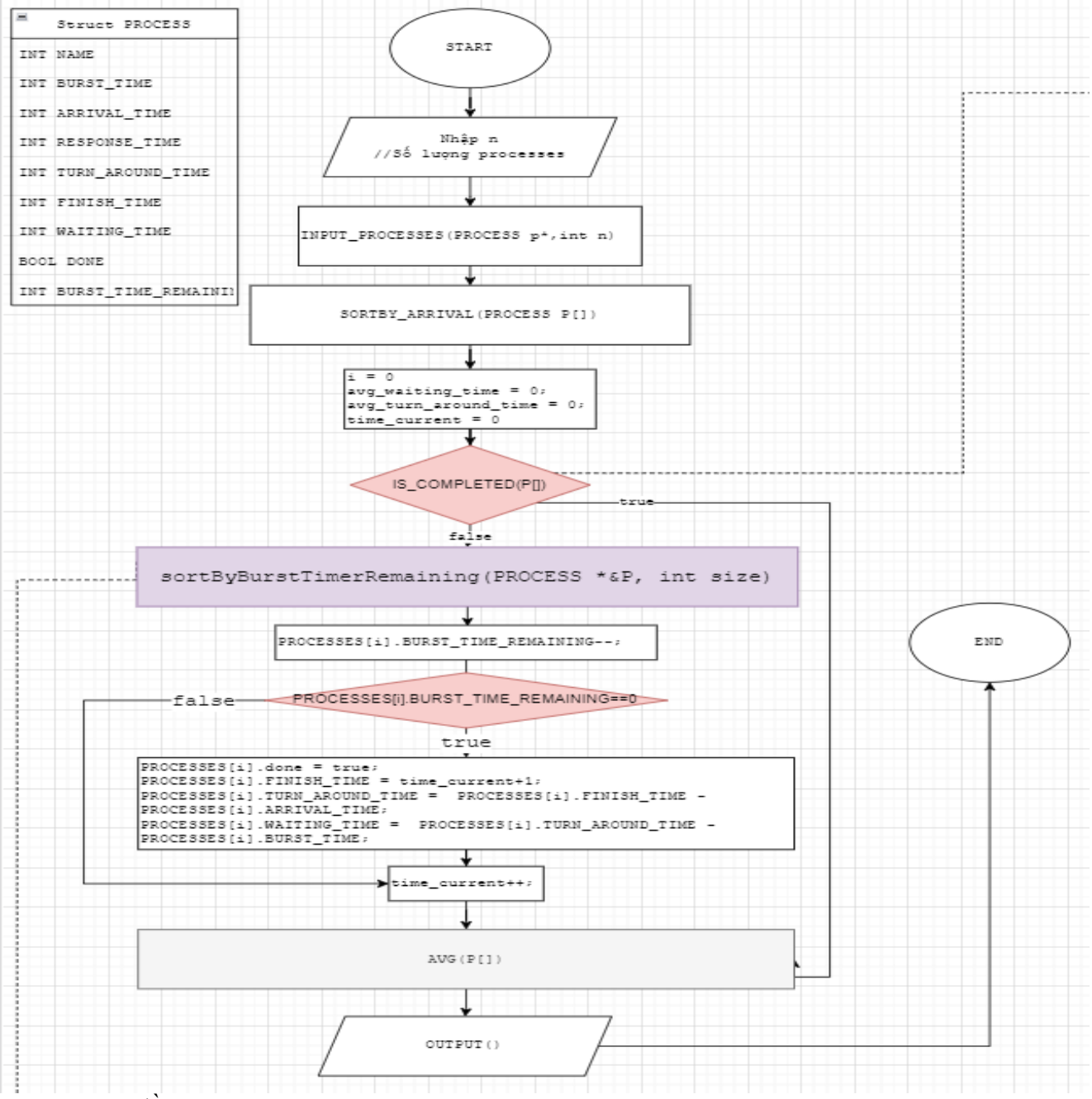
Job	Arrival Time	Burst Time	Finish Time	Turnaround Time	Waiting Time
A	0	8	8	8	0
B	2	19	48	46	27
C	4	3	11	7	4
D	5	6	17	12	6
E	7	12	29	22	10
			Average	95 / 5 = 19	47 / 5 = 9.4

```
        SHORTEST JOB FIRST

|Job|Arrival_Time|Burst_Time|Finish_Time|Turnaround_Time|Waiting_Time|
|  A|           0|         8|          8|              8|           0|
|  B|           2|        19|         48|             46|          27|
|  C|           4|         3|         11|              7|           4|
|  D|           5|         6|         17|             12|           6|
|  E|           7|        12|         29|             22|          10|
Average Turn Around Time: 19.000000
Average Turn Waiting Time: 9.400000
```

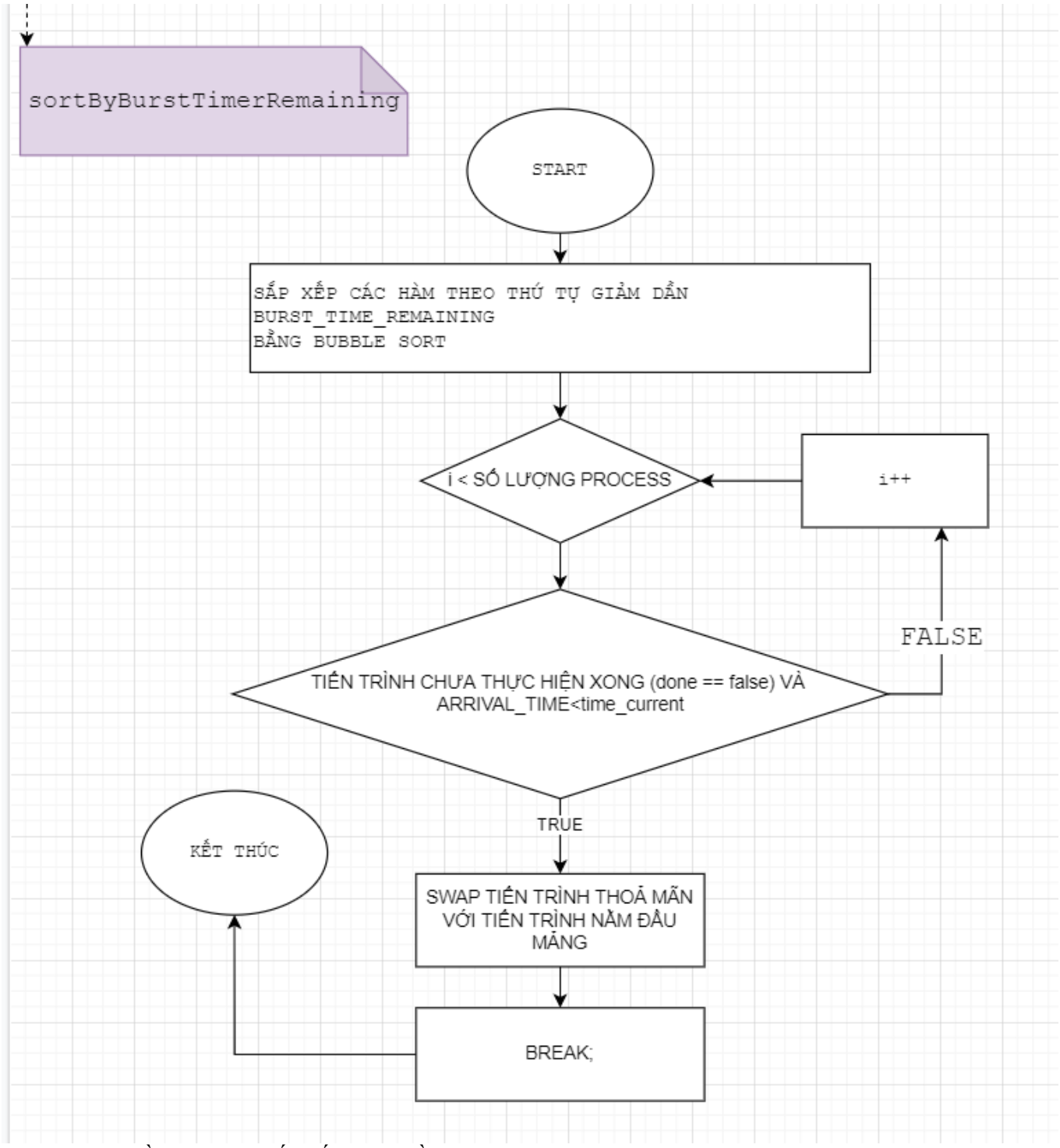
Hình 6: Test Case 03 Chạy Bằng Tay Và Kết quả code

TASK 02: GIẢI THUẬT SRTF

1. Lưu đồ giải thuật SRTF.



Hình 4: Lưu đồ giải thuật SRTF



Hình 5: Lưu đồ thuật toán sắp xếp giảm dần theo REMAINING_BURST_TIME

2. Code giải thuật SRTF.

```
#include <iostream>
#include <string>
#include <string.h>
#include <stdio.h>
```

```cpp
#include <stdlib.h>
#include <vector>

using namespace std;

struct PROCESS
{
    string NAME;
    int BURST_TIME;
    int ARRIVAL_TIME;
    int REPONSE_TIME;
    int TURN_AROUND_TIME;
    int FINISH_TIME;
    int WAITING_TIME;
    bool done = false;
    int BURST_TIME_REMAINING;
};

double avg_waiting_time = 0;
double avg_turn_around_time = 0;

void Add_Process(int &size, PROCESS *&P, PROCESS process)
{
    PROCESS *newArr = new PROCESS[size + 1];
    for (int index = 0; index < size; index++)
    {
        newArr[index] = P[index];
    }
    newArr[size] = process;
    size++;
    P = newArr;
}

void swap(PROCESS &p1, PROCESS &p2)
{
    PROCESS tmp;
    tmp = p1;
    p1 = p2;
    p2 = tmp;
}

void sortByArrivalTime(PROCESS *&P, int size)
{
    for (int i = 0; i < size; i++)
    {
```

```
        for (int j = i + 1; j < size; j++)
        {
            if (P[i].ARRIVAL_TIME > P[j].ARRIVAL_TIME)
            {
                swap(P[i], P[j]);
            }
        }
    }
}

void sortByBurstTimeRemaining(PROCESS *&P, int n, int time_current)
{
     for (int i = 0; i < n; i++)
    {
        for (int j = i + 1; j < n; j++)
        {
            if (P[i].BURST_TIME_REMAINING > P[j].BURST_TIME_REMAINING)
            {
                swap(P[i], P[j]);
            }
        }
    }
       for (int i = 0; i < n; i++)
        {
            if (P[i].done == false &&P[i].ARRIVAL_TIME<=time_current)
            {
                swap(P[i], P[0]);
                break;
            }
        }
}

void Input_Process(PROCESS *&P, int &size)
{
    cout << "Input the number of Processes: ";
    cin >> size;

    PROCESS *newArr = new PROCESS[size];

    for (int i = 0; i < size; i++)
    {
        cin.ignore(32767, '\n');
        cout << "----------------[" << i << "]-----------------\n";
        cout << "ARRIVAL_TIME: ";
        cin >> newArr[i].ARRIVAL_TIME;
```

```
        cout << "BUSRT_TIME: ";
        cin >> newArr[i].BURST_TIME;

        newArr[i].NAME = (char)('A' + i);
         newArr[i].BURST_TIME_REMAINING = newArr[i].BURST_TIME;

    }
    P = newArr;
}

string space(int n)
{
    string res;
    for (int i = 0; i < n; i++)
    {
        res += " ";
    }
    return res;
}

void Output_Process(int numberOfprocesses, PROCESS *P)
{
    sortByArrivalTime(P, numberOfprocesses);
    string Attribute[6] = {"Job", "Arrival_Time", "Burst_Time", "Finish_Time",
"Turnaround_Time", "Waiting_Time"};

    int num_columns = sizeof(Attribute) / sizeof(Attribute[0]);
    vector<vector<string>> board;
    vector<string> row {"\tShortest Remaining Time First, SRTF\n" } ;

    board.push_back(row);
    for (int i = 0; i < numberOfprocesses + 1; i++)
    {
        vector<string> row;
        if (i == 0)
        {
            for (int j = 0; j < num_columns; j++)
            {
                if (j == 0)
                    row.push_back("|" + Attribute[0] + "|");
                else
                    row.push_back(Attribute[j] + "|");
            }
            board.push_back(row);
```

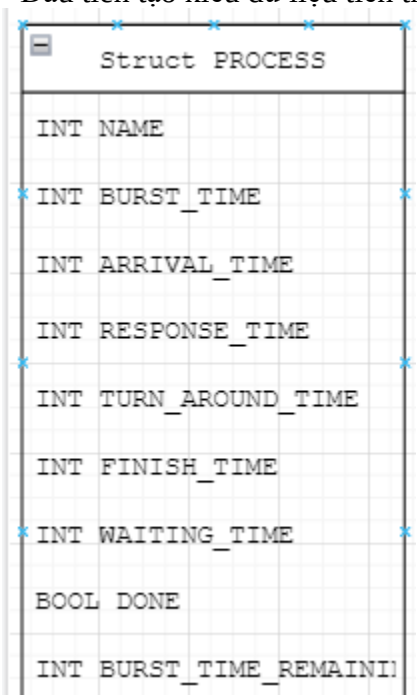
```
        }
        else
        {
            int ele = i - 1;
            string Value[6] = {P[ele].NAME, to_string(P[ele].ARRIVAL_TIME),
to_string(P[ele].BURST_TIME), to_string(P[ele].FINISH_TIME),
to_string(P[ele].TURN_AROUND_TIME), to_string(P[ele].WAITING_TIME)};
            for (int j = 0; j < num_columns; j++)
            {
                if (j == 0)
                    row.push_back("|" + space(Attribute[0].length() - Value[0].length())
+ Value[0] + "|");
                else
                    row.push_back(space(Attribute[j].length() - Value[j].length()) +
Value[j] + "|");
            }
            board.push_back(row);

        }

    }
         row = {"Average Turn Around Time: " +to_string(avg_turn_around_time)+"\n",
                        "Average Turn Waiting Time: "
 +to_string(avg_waiting_time)+"\n"  } ;
        board.push_back(row);


    for (int i = 0; i < numberOfprocesses + 2+1; i++)
    {
        for (int j = 0; j < num_columns; j++)
        {
            if (i==numberOfprocesses+1+1)
            {
                cout << board[i][0].c_str();
                cout << board[i][1].c_str();
                break;
            }
             if (i==0)
            {
                cout << board[i][0].c_str();
                break;
            }

            cout << board[i][j].c_str();
        }
```

```
        cout << "\n";
    }
}

bool isCompleted(PROCESS *P, int Size)
{
    for (int i = 0; i < Size; i++)
    {
        if (!P[i].done)
        {
            return false;
        }
    }
    return true;
}

void AVG(PROCESS *P,int Size){
    for (int i = 0; i < Size; i++)
    {
        avg_turn_around_time += (double)P[i].TURN_AROUND_TIME/Size;
        avg_waiting_time += (double)P[i].WAITING_TIME/Size;
    }
}

int main()
{
    int numberOfprocesses = 1;
    PROCESS *PROCESSES = new PROCESS[numberOfprocesses];
    Input_Process(PROCESSES, numberOfprocesses);

    int i = 0;
     for(int time_current=0;!isCompleted(PROCESSES,numberOfprocesses); time_current++)
    {
        sortByBurstTimeRemaining(PROCESSES,numberOfprocesses,time_current);
        PROCESSES[i].BURST_TIME_REMAINING--;

        if(PROCESSES[i].BURST_TIME_REMAINING==0)
        {
            PROCESSES[i].done = true;
            PROCESSES[i].FINISH_TIME = time_current+1;
            PROCESSES[i].TURN_AROUND_TIME =  PROCESSES[i].FINISH_TIME -
 PROCESSES[i].ARRIVAL_TIME;
            PROCESSES[i].WAITING_TIME =  PROCESSES[i].TURN_AROUND_TIME -
PROCESSES[i].BURST_TIME;
        }
```

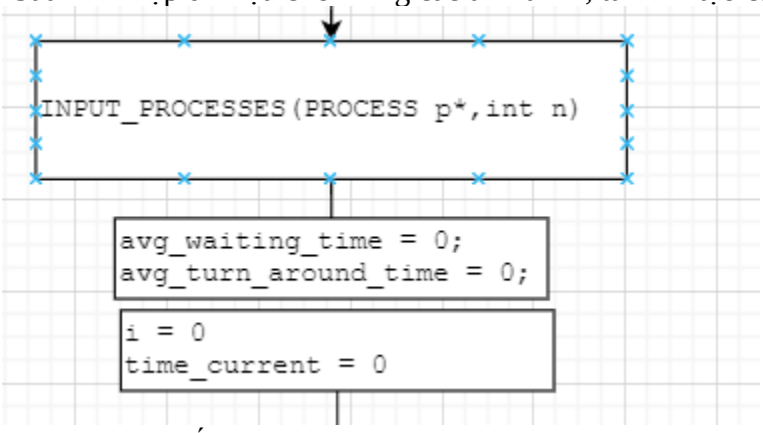
```
    }

    AVG(PROCESSES,numberOfprocesses);
    Output_Process(numberOfprocesses, PROCESSES);

    return 0;
}
```

3. DIỄN GIẢI SRTF

- Đầu tiên tạo kiểu dữ liệu tiến trình với các thuộc tính theo Lưu đồ trên.



-Sau khi nhập dữ liệu cho mảng các tiến trình, ta khởi tạo các biến cần thiết cho vòng lặp



- Khi mà các tiến trình chưa hoàn thành (isCompleted!=true):

+Ta sẽ sắp xếp lại các mảng theo thứ tự giảm dần theo RemainingBurstTime cứ sau mỗi vòng lặp, hàm sắp xếp này đồng thời cũng đẩy tiến trình có RemainingBurstTime ngắn nhất lên đầu mảng.

+Cứ sau mỗi vòng lặp ta giảm dần RemainingBurstTime và tăng dần current_time
+Sẽ có một lệnh if nhằm chọn ra tiến trình đã thực thi xong (BURST_TIME_REMAINING == 0)

Và thực thi:

```
PROCESSES[i].done = true;
PROCESSES[i].FINISH_TIME = time_current+1;
PROCESSES[i].TURN_AROUND_TIME =  PROCESSES[i].FINISH_TIME -
PROCESSES[i].ARRIVAL_TIME;
PROCESSES[i].WAITING_TIME =  PROCESSES[i].TURN_AROUND_TIME -
PROCESSES[i].BURST_TIME;
```

-Sau khi tất cả các tiến trình hoàn thành, ta sẽ tính toán các chỉ số liên quan và in ra màn hình.

4. Tính đúng đắn của giải thuật SRTF và code.

Output SRTF

Gantt Chart

A	D	A	C	B	E
0	5	11	18	27	41	56

Job	Arrival Time	Burst Time	Finish Time	Turnaround Time	Waiting Time
A	0	12	18	18	6
B	3	14	41	38	24
D	5	6	11	6	0
C	7	9	27	20	11
E	14	15	56	42	27
			Average	124 / 5 = 24.8	68 / 5 = 13.6

```
        Shortest Remaining Time First, SRTF

|Job|Arrival_Time|Burst_Time|Finish_Time|Turnaround_Time|Waiting_Time|
|  A|           0|        12|         18|             18|           6|
|  B|           3|        14|         41|             38|          24|
|  C|           5|         6|         11|              6|           0|
|  D|           7|         9|         27|             20|          11|
|  E|          14|        15|         56|             42|          27|
Average Turn Around Time: 24.800000
Average Turn Waiting Time: 13.600000
```

Hình 4: Test Case 01 Chạy Bằng Tay Và Kết quả code

Output SRTF

Gantt Chart

A	B	D	E	C	A
0	2	9	12	18	26	36

Job	Arrival Time	Burst Time	Finish Time	Turnaround Time	Waiting Time
A	0	12	36	36	24
B	2	7	9	7	0
C	5	8	26	21	13
D	9	3	12	3	0
E	12	6	18	6	0
			Average	73 / 5 = 14.6	37 / 5 = 7.4

```
        Shortest Remaining Time First, SRTF

|Job|Arrival_Time|Burst_Time|Finish_Time|Turnaround_Time|Waiting_Time|
|  A|           0|        12|         36|             36|          24|
|  B|           2|         7|          9|              7|           0|
|  C|           5|         8|         26|             21|          13|
|  D|           9|         3|         12|              3|           0|
|  E|          12|         6|         18|              6|           0|
Average Turn Around Time: 14.600000
Average Turn Waiting Time: 7.400000
```

Hình 6 Test Case 02 Chạy Bằng Tay Và Kết quả code

Output SRTF

Gantt Chart

A	C	A	D	E	B
0	4	7	11	17	29	48

Job	Arrival Time	Burst Time	Finish Time	Turnaround Time	Waiting Time
A	0	8	11	11	3
B	2	19	48	46	27
C	4	3	7	3	0
D	5	6	17	12	6
E	7	12	29	22	10
			Average	94 / 5 = 18.8	46 / 5 = 9.2

```
        Shortest Remaining Time First, SRTF

|Job|Arrival_Time|Burst_Time|Finish_Time|Turnaround_Time|Waiting_Time|
|  A|           0|         8|         11|             11|           3|
|  B|           2|        19|         48|             46|          27|
|  C|           4|         3|          7|              3|           0|
|  D|           5|         6|         17|             12|           6|
|  E|           7|        12|         29|             22|          10|
Average Turn Around Time: 18.800000
Average Turn Waiting Time: 9.200000
```

Hình 7:Test Case 03 Chạy Bằng Tay Và Kết quả code

TASK 03: GIẢI THUẬT RR

1.Lưu đồ giải thuật.

2.Code giải thuật.

3.Tính đúng đắn của giải thuật.

4. Tính đúng đắn của code.