Nhóm: 8
Thông tin thành viên:
- Dương Thuận Trí- 22521517
- Trương Quốc Vinh- 22521682
- Nguyễn Thị Kim Ngân- 21521174
- Nông Tiến Dũng- 20521213

Lớp: 1

HỆ ĐIỀU HÀNH
BÁO CÁO LAB 4

CHECKLIST

3.5. BÀI TẬP THỰC HÀNH

	BT 1	BT 2
Vẽ lưu đồ giải thuật	☐	☐
Chạy tay lưu đồ giải thuật	☐	☐
Hiện thực code	☐	☐
Chạy code và kiểm chứng	☐	☐

3.6. BÀI TẬP ÔN TẬP

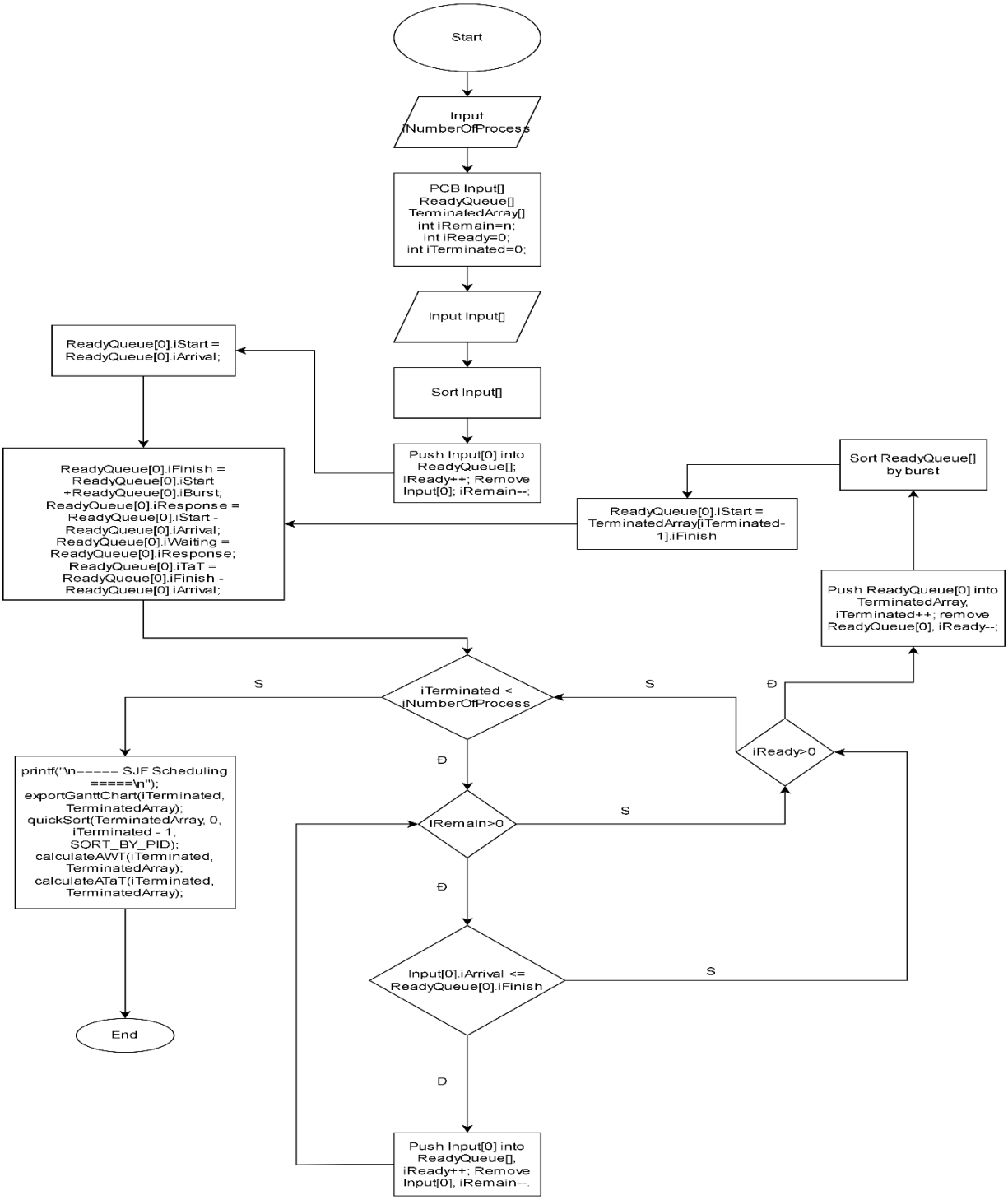
	BT 1
Vẽ lưu đồ giải thuật	☐
Chạy tay lưu đồ giải thuật	☐
Hiện thực code	☐
Chạy code và kiểm chứng	

Tự chấm điểm: 10

2.5. BÀI TẬP THỰC HÀNH

1. Giải thuật Shortest-Job-First

- **Lưu đồ:**



➢ **Chạy tay lưu đồ:**

Process	Arrival Time	Burst Time
P1	0	12
P2	2	7
P3	5	8
P4	9	3
P5	12	6

- Tại giây 0:

Input[5]	ReadyQueue[5]	TerminatedArray[5]
P2(2, 7)	P1(0, 12)	
P3(5, 8)		
P4(9, 3)		
P5(12, 6)		
iRemain = 4	iReady = 1	iTerminated = 0

- Tại giây 12:

Input[5]	ReadyQueue[5]	TerminatedArray[5]
	P4(9, 3)	P1(0, 12)
	P5(12, 6)	
	P2(2, 7)	
	P3(5, 8)	
iRemain = 0	iReady = 4	iTerminated = 1

- Tại giây 15:

Input[5]	ReadyQueue[5]	TerminatedArray[5]
	P5(12, 6)	P1(0, 12)
	P2(2, 7)	P4(9, 3)
	P3(5, 8)	
iRemain = 0	iReady = 3	iTerminated = 2

- Tại giây 21:

Input[5]	ReadyQueue[5]	TerminatedArray[5]
	P2(2, 7)	P1(0, 12)
	P3(5, 8)	P4(9, 3)
		P5(12, 6)
iRemain = 0	iReady = 2	iTerminated = 3

- Tại giây 28:

Input[5]	ReadyQueue[5]	TerminatedArray[5]
	P3(5, 8)	P1(0, 12)
		P4(9, 3)
		P5(12, 6)
		P2(2, 7)
iRemain = 0	iReady = 1	iTerminated = 4

- Tại giây 36:

Input[5]	ReadyQueue[5]	TerminatedArray[5]
		P1(0, 12)
		P4(9, 3)
		P5(12, 6)
		P2(2, 7)
		P3(5, 8)
iRemain = 0	iReady = 0	iTerminated = 5

➢ **Code:**

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <time.h>
#include <limits.h>
#define SORT_BY_ARRIVAL 0
#define SORT_BY_PID 1
#define SORT_BY_BURST 2
#define SORT_BY_START 3
typedef struct
{
    int iPID;
    int iArrival, iBurst;
    int iStart, iFinish, iWaiting, iResponse, iTaT;
} PCB;

void inputProcess(int n, PCB P[])
{
        srand((unsigned) time(NULL));

    for(int i = 0 ; i < n;i++){
        P[i].iPID= i+1;
        P[i].iArrival = rand() % 21;
        P[i].iBurst = (rand()% 11) +2;
    }
}
void printProcess(int n, PCB P[])
{
    for(int i = 0 ; i < n;i++){
        printf("PID: %d\n",P[i].iPID);
        printf("Arrival Time: %d\n",P[i].iArrival);
        printf("Burst Time: %d\n",P[i].iBurst);
        printf("Start Time: %d\n",P[i].iStart);
        printf("Finish Time: %d\n",P[i].iFinish);
        printf("Waiting Time: %d\n",P[i].iWaiting);
        printf("Response Time: %d\n",P[i].iResponse);
        printf("Turn Around Time: %d\n\n",P[i].iTaT);

    }
}
void exportGanttChart(int n, PCB P[])
{
```

```
    printf("\nGantt Chart\n");
    for(int i = 0 ; i < n;i++){
        printf("P%d\t",P[i].iPID);
    }
    printf("\n");
    for(int i = 0 ; i < n;i++){
        printf("%d\t",P[i].iStart);
    }
    printf("%d\t",P[n-1].iFinish);
    printf("\n");
}
void pushProcess(int *n, PCB P[], PCB Q)
{
    P[*n] = Q;
    (*n)++;
}

void removeProcess(int *n, int index, PCB P[]){
    for(int i = index ; i < *n - 1;i++){
        P[i] = P[i+1];
    }
    (*n)--;
}
int swapProcess(PCB *P, PCB *Q) {
    PCB temp = *P;
    *P = *Q;
    *Q = temp;
}
int partition (PCB P[], int low, int high, int iCriteria) {
    if (iCriteria == SORT_BY_ARRIVAL) {
        int pivot = P[high].iArrival;

        int i = (low-1);

        for(int j = low; j <= high; j++)
        {
            if (P[j].iArrival < pivot)
            {
                i++;
                swapProcess(&P[i], &P[j]);
            }
        }
        swapProcess(&P[i+1], &P[high]);
        return (i+1);
    } else if (iCriteria == SORT_BY_PID) { // Sort by PID
```

```
        int pivot = P[high].iPID;

        int i = (low-1);

        for(int j = low; j <= high; j++)
        {
            if (P[j].iPID < pivot)
            {
                i++;
                swapProcess(&P[i], &P[j]);
            }
        }
        swapProcess(&P[i+1], &P[high]);
        return (i+1);
    } else if (iCriteria == SORT_BY_BURST) {
        int pivot = P[high].iBurst;

        int i = (low-1);

        for(int j = low; j <= high; j++)
        {
            if (P[j].iBurst < pivot)
            {
                i++;
                swapProcess(&P[i], &P[j]);
            }
        }
        swapProcess(&P[i+1], &P[high]);
        return (i+1);
    } else if (iCriteria == SORT_BY_START) {
        int pivot = P[high].iStart;

        int i = (low-1);

        for(int j = low; j <= high; j++)
        {
            if (P[j].iStart < pivot)
            {
                i++;
                swapProcess(&P[i],&P[j]);
            }
        }
        swapProcess(&P[i+1], &P[high]);
        return (i+1);
    }
```

```c
}

void quickSort(PCB P[], int low, int high, int iCriteria) {
    if(low < high) {

    int pi = partition(P, low, high, iCriteria);

    quickSort(P, low, pi-1, iCriteria);
    quickSort(P, pi+1, high, iCriteria);
    }
}
void calculateAWT(int n, PCB P[])
{
    int sum = 0;
    for(int i = 0 ; i < n;i++){
        sum += P[i].iWaiting;
    }
    printf("Average Waiting Time: %f\n",(float)sum/n);
}
void calculateATaT(int n, PCB P[]){
    int sum = 0;
    for(int i = 0 ; i < n;i++){
        sum += P[i].iTaT;
    }
    printf("Average itat Time: %f\n",(float)sum/n);
}


int main()
{
    PCB Input[10];
    PCB ReadyQueue[10];
    PCB TerminatedArray[10];
    int iNumberOfProcess;


    printf("Please input number of Process: ");
    scanf("%d", &iNumberOfProcess);

    int iRemain = iNumberOfProcess, iReady = 0, iTerminated = 0;
    inputProcess(iNumberOfProcess, Input);

    quickSort(Input, 0, iNumberOfProcess - 1,SORT_BY_ARRIVAL);
```

```
    pushProcess(&iReady, ReadyQueue, Input[0]);
    removeProcess(&iRemain, 0, Input);

    ReadyQueue[0].iStart = ReadyQueue[0].iArrival;
    ReadyQueue[0].iFinish = ReadyQueue[0].iStart + ReadyQueue[0].iBurst;
    ReadyQueue[0].iResponse = ReadyQueue[0].iStart -ReadyQueue[0].iArrival;
    ReadyQueue[0].iWaiting = ReadyQueue[0].iResponse;
    ReadyQueue[0].iTaT = ReadyQueue[0].iFinish -ReadyQueue[0].iArrival;


    bool flag=false;
    while (iTerminated < iNumberOfProcess)
    {
        while (iRemain > 0)
        {
            if (Input[0].iArrival <= ReadyQueue[0].iFinish)
            {
                pushProcess(&iReady, ReadyQueue, Input[0]);
                removeProcess(&iRemain, 0, Input);
                flag=false;
                continue;
            }
            else{
                if(iReady==0){
                     flag=true;
                     pushProcess(&iReady, ReadyQueue, Input[0]);
                      removeProcess(&iRemain, 0, Input);
                      break;
                }
                else {flag=false;break;}
            }

        }
        if (iReady > 0)
        {
            pushProcess(&iTerminated, TerminatedArray,ReadyQueue[0]);
            removeProcess(&iReady, 0, ReadyQueue);
            quickSort(ReadyQueue, 0, iReady-1,SORT_BY_BURST);

            if(flag)  ReadyQueue[0].iStart = ReadyQueue[0].iArrival;
            else ReadyQueue[0].iStart = TerminatedArray[iTerminated - 1].iFinish;
            ReadyQueue[0].iFinish = ReadyQueue[0].iStart + ReadyQueue[0].iBurst;
            ReadyQueue[0].iResponse = ReadyQueue[0].iStart -ReadyQueue[0].iArrival;
            ReadyQueue[0].iWaiting = ReadyQueue[0].iResponse;
            ReadyQueue[0].iTaT = ReadyQueue[0].iFinish - ReadyQueue[0].iArrival;
```

```
        }
    }
            printProcess(iTerminated, TerminatedArray);

    printf("\n===== SJF Scheduling =====\n");
    exportGanttChart(iTerminated, TerminatedArray);
    quickSort(TerminatedArray, 0, iTerminated - 1,SORT_BY_PID);
    calculateAWT(iTerminated, TerminatedArray);
    calculateATaT(iTerminated, TerminatedArray);
    return 0;
}
```

- Test case 1:

```
PS D:\Study\Classes\HK3\HĐH\ThucHanh\Lab4> gcc sjf.c -o sjf
PS D:\Study\Classes\HK3\HĐH\ThucHanh\Lab4> ./sjf
Please input number of Process: 5
PID: 4
Arrival Time: 5
Burst Time: 9
Start Time: 5
Finish Time: 14
Waiting Time: 0
Response Time: 0
Turn Around Time: 9

PID: 1
Arrival Time: 11
Burst Time: 6
Start Time: 14
Finish Time: 20
Waiting Time: 3
Response Time: 3
Turn Around Time: 9

PID: 5
Arrival Time: 18
Burst Time: 6
Start Time: 20
Finish Time: 26
Waiting Time: 2
Response Time: 2
Turn Around Time: 8

 PID: 3
 Arrival Time: 5
 Burst Time: 7
 Start Time: 26
 Finish Time: 33
 Waiting Time: 21
 Response Time: 21
 Turn Around Time: 28

 PID: 2
 Arrival Time: 20
 Burst Time: 10
 Start Time: 33
 Finish Time: 43
 Waiting Time: 13
 Response Time: 13
 Turn Around Time: 23


 ===== SJF Scheduling =====

 Gantt Chart
 P4      P1      P5      P3      P2
 5       14      20      26      33      43
 Average Waiting Time: 7.800000
 Average itat Time: 15.400000
```

- Test case 2:

```
PS D:\Study\Classes\HK3\HĐH\ThucHanh\Lab4> ./sjf
Please input number of Process: 5
PID: 2
Arrival Time: 8
Burst Time: 6
Start Time: 8
Finish Time: 14
Waiting Time: 0
Response Time: 0
Turn Around Time: 6

PID: 4
Arrival Time: 9
Burst Time: 4
Start Time: 14
Finish Time: 18
Waiting Time: 5
Response Time: 5
Turn Around Time: 9

PID: 3
Arrival Time: 16
Burst Time: 5
Start Time: 18
Finish Time: 23
Waiting Time: 2
Turn Around Time: 7
```

```
PID: 5
Arrival Time: 16
Burst Time: 5
Start Time: 23
Finish Time: 28
Waiting Time: 7
Response Time: 7
Turn Around Time: 12

PID: 1
Arrival Time: 8
Burst Time: 5
Start Time: 28
Finish Time: 33
Waiting Time: 20
Response Time: 20
Turn Around Time: 25


===== SJF Scheduling =====

Gantt Chart
P2      P4      P3      P5      P1
8       14      18      23      28      33
Average Waiting Time: 6.800000
Average itat Time: 11.800000
```

- Test case 3:

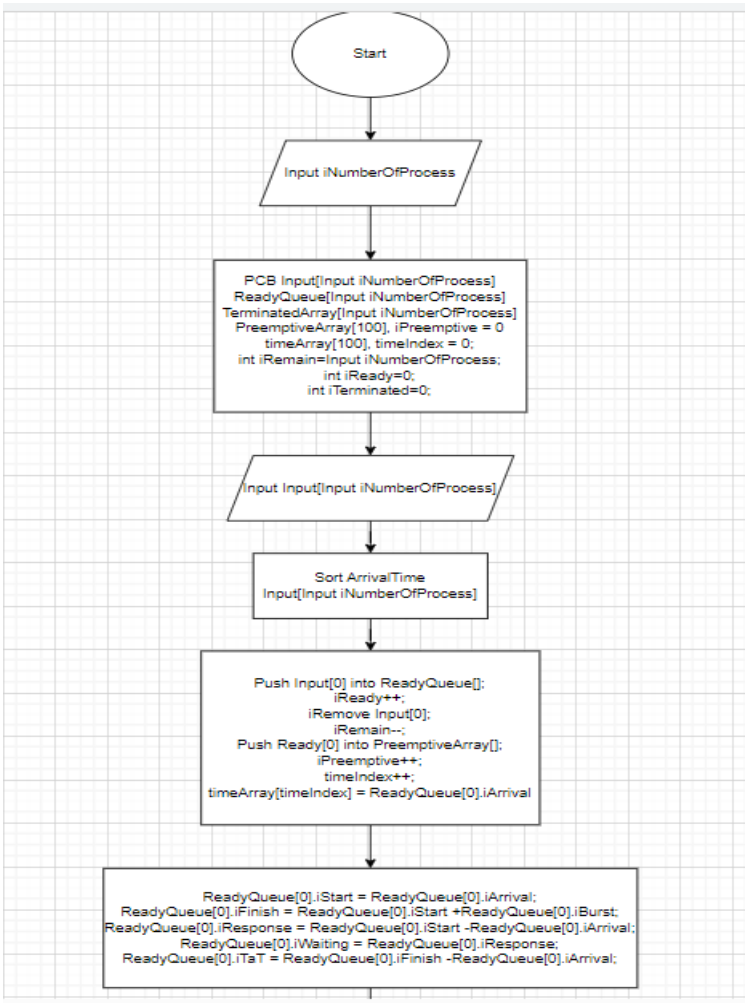
```
PS D:\Study\Classes\HK3\HĐH\ThucHanh\Lab4> ./sjf
Please input number of Process: 5
PID: 2
Arrival Time: 0
Burst Time: 9
Start Time: 0
Finish Time: 9
Waiting Time: 0
Response Time: 0
Turn Around Time: 9

PID: 1
Arrival Time: 4
Burst Time: 3
Start Time: 9
Finish Time: 12
Waiting Time: 5
Response Time: 5
Turn Around Time: 8

PID: 3
Arrival Time: 5
Burst Time: 4
Start Time: 12
Finish Time: 16
Waiting Time: 7
Response Time: 7
Turn Around Time: 11
```

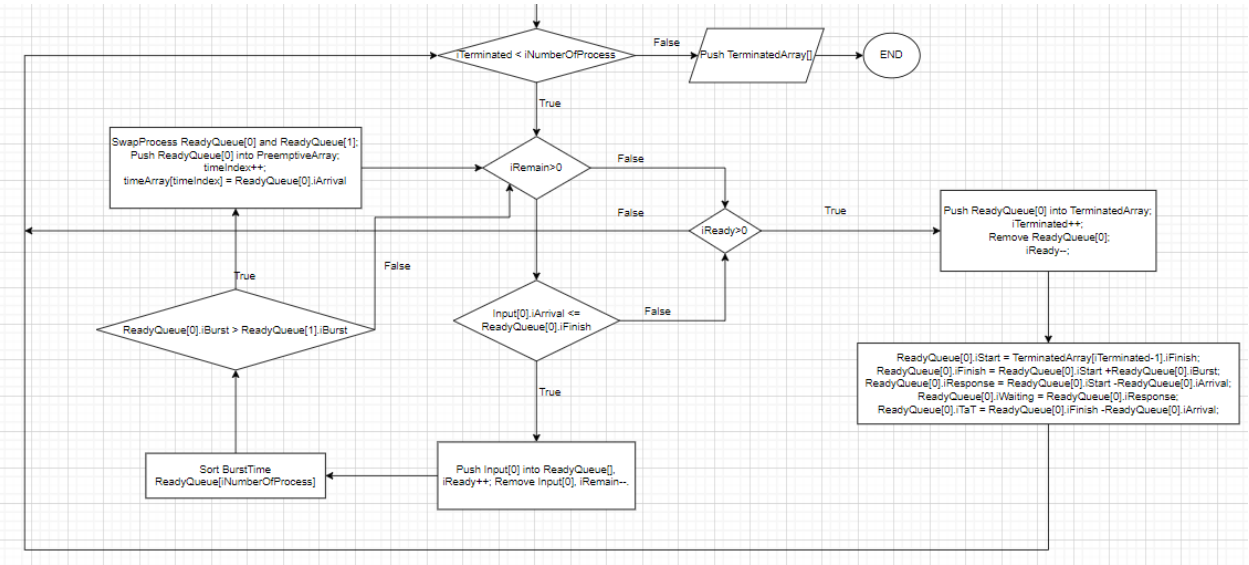
```
PID: 4
Arrival Time: 2
Burst Time: 7
Start Time: 16
Finish Time: 23
Waiting Time: 14
Response Time: 14
Turn Around Time: 21

PID: 5
Arrival Time: 7
Burst Time: 8
Start Time: 23
Finish Time: 31
Waiting Time: 16
Response Time: 16
Turn Around Time: 24


===== SJF Scheduling =====

Gantt Chart
P2      P1      P3      P4      P5
0       9       12      16      23      31
Average Waiting Time: 8.400000
Average itat Time: 14.600000
```

2. Giải thuật Shortest-Remaining-Time-First hoặc Round Robin

Trả lời: **Giải thuật Shortest-Remaining-Time-First**

- **Lưu đồ:**





Link lưu đồ: https://app.diagrams.net/#G1v88DIaL9lPyO3XHpyHlyouRyuEkaeudo

- Test case:

Process	Arrival Time	Burst Time
P1	0	12
P2	2	7
P3	5	8
P4	9	3
P5	12	6

- Các tiến trình được đưa vào. Tại thời điểm 0, tiến trình P1 được thêm vào ReadyQueue[]

Input[5]	ReadyQueue[5]	TerminatedArray[5]
P2(2, 7)	P1(0, 12)	
P3(5, 8)		
P4(9, 3)		
P5(12, 6)		
iRemain = 4	iReady = 1	iTerminated = 0

- Trong khi P1 chạy đến thời điểm 2 thì P2 được đưa vào ReadyQueue[].

Input[5]	ReadyQueue[5]	TerminatedArray[5]
P3(5, 8)	P1(0, 12)	
P4(9, 3)	P2(2, 7)	
P5(12, 6)		
iRemain = 3	iReady = 2	iTerminated = 0

- Khi P2 vào thì P2 trưng dụng và chiếm CPU (vì burst time của P2 nhỏ hơn burst time còn lại của P1) . Trong lúc đó thì P3 cũng vào ReadyQueue[] tại thời điểm 5.

Input[5]	ReadyQueue[5]	TerminatedArray[5]
P4(9, 3)	P2(2, 7)	
P5(12, 6)	P3(5, 8)	
	P1(0, 10)	
iRemain = 2	iReady = 3	iTerminated = 0

- Sau khi P2 thực thi xong và được đưa vào TerminatedArray[], tại thời điểm 9 thì P4 cũng vào ReadyQueue[] và nó chiếm CPU để thực thi. (vì có burst time nhỏ nhất trong các tiến trình có trong ReadyQueue[] (sort by burst) .

Input[5]	ReadyQueue[5]	TerminatedArray[5]
P5(12, 6)	P4(9, 3)	P2(2, 7)
	P3(5, 8)	
	P1(0, 10)	
iRemain = 1	iReady = 3	iTerminated = 1

- Sau khi P4 thực thi xong và được đưa vào TerminatedArray[], tại thời điểm 12 thì P5 cũng được đưa vào ReadyQueue[] và nó chiếm CPU để thực thi. (vì có burst time nhỏ nhất trong các tiến trình có trong ReadyQueue[] (sort by burst).

Input[5]	ReadyQueue[5]	TerminatedArray[5]
	P5(12, 6)	P2(2, 7)
	P3(5, 8)	P4(9, 3)
	P1(0, 10)	
iRemain = 0	iReady = 3	iTerminated = 2

- Các tiến trình tiếp tục thực thi lần lượt theo thứ tự P5, P3, P1 (tiến trình có burst time nhỏ hơn sẽ được thực thi trước (sort by burst)).

Input[5]	ReadyQueue[5]	TerminatedArray[5]
		P2(2, 7)
		P4(9, 3)
		P5(12, 6)

		P3(5, 8)
		P1(0, 10)
iRemain = 0	iReady = 0	iTerminated = 5

Sau khi các tiến trình thực thi xong thì sẽ lần lượt được chuyển sang TerminatedArray[], lúc này iTerminated = 5, chính là số lượng tiến trình ban đầu nhập vào và giải thuật kết thúc tại thời điểm 36.

- **Code:**

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <limits.h>

#define SORT_BY_ARRIVAL 0
#define SORT_BY_PID 1
#define SORT_BY_BURST 2
#define SORT_BY_START 3


typedef struct
{
    int iPID;
    int iArrival, iBurst;
    int iStart, iFinish, iWaiting, iResponse, iTaT;
} PCB;

void inputProcess(int n, PCB P[])
{
```

```
    int i;
    srand(time(NULL));
    for (int i = 0; i < n; i++)
    {
        P[i].iPID = i + 1;
        P[i].iArrival = rand() % 21;
        P[i].iBurst = rand() % 11 + 2;
        P[i].iStart = P[i].iFinish = P[i].iWaiting = P[i].iResponse
        = P[i].iTaT = 0;
    }
}

void printProcess(int n, PCB P[])
{
    printf("\n_____\n");
    printf("| PID | Arrival | Burst | Start | Finish | Turnaround | \n");
    printf("-----\n");
    int i;
    for (i = 0; i < n; i++)
    {
        printf("| P%-4d | %-7d | %-5d | %-5d | %-6d | %-10d |\n",
        P[i].iPID, P[i].iArrival, P[i].iBurst, P[i].iStart,
        P[i].iFinish, P[i].iTaT);
    }
    printf("\n");
}
```

```
void exportGanttChart (int n, PCB P[], int m, int timeArray[]) {
    int nameArray[n];

    // Khoi tao gia tri cho mang luu ten cac tien trinh
    for (int i = 0; i < n; i++) {
        nameArray[i] = P[i].iPID;
    }

    // In ten tien trinh
    int nameSize = n;
    int timeSize = m;
    int timeIndex = 0;
    int nameIndex = 0;
```

```
    while (timeIndex < timeSize - 1) {
        int count = timeArray[timeIndex + 1] - timeArray[timeIndex];
        if (nameArray[nameIndex] == 0) {
            for (int i = 0; i <= count; i++) {
                printf(" ");
            }
        } else {
            count = (count - 2) / 2;
            for (int i = 0; i <= count; i++) {
                printf(" ");
            }
            printf("P%d", nameArray[nameIndex]);
            if ((timeArray[timeIndex + 1] - timeArray[timeIndex]) % 2 == 0) {
                count -= 1;
            }
            for (int i = 0; i <= count; i++) {
                printf(" ");
            }
        }
        nameIndex++;
        timeIndex++;
    }
    printf("\n");
```

```
    // In gian do gantt
    timeIndex = 0;
    while(timeIndex < timeSize - 1) {
        int count = timeArray[timeIndex + 1] - timeArray[timeIndex];
        printf("|");
        for (int i = 0; i < count; i++) {
            printf("-");
        }
        timeIndex++;
    }
    printf("|");
    printf("\n");

    // In chi so
    timeIndex = 0;
    while(timeIndex < timeSize - 1) {
```

```
        int count = timeArray[timeIndex + 1] - timeArray[timeIndex];
        if (timeArray[timeIndex] >= 10) {
            count--;
        }
        printf("%d", timeArray[timeIndex]);
        for (int i = 0; i < count; i++) {
            printf(" ");
        }
        timeIndex++;
    }
    printf("%d", timeArray[timeIndex]);
};

void pushProcess(int *n, PCB P[], PCB Q)
{
    P[*n] = Q;
    (*n)++;
}

void removeProcess(int *n, int index, PCB P[])
{
    for (int i = index; i < *n - 1; i++)
    {
        P[i] = P[i + 1];
    }
    (*n)--;
}
```

```
int swapProcess(PCB *P, PCB *Q)
{
    PCB temp;
    temp = *P;
    *P = *Q;
    *Q = temp;
};

int partition(PCB P[], int low, int high, int iCriteria)
{
    int vt = low - 1;
    switch (iCriteria)
    {
    case SORT_BY_ARRIVAL:
    {
        int pivot = P[high].iArrival;
        for (int i = low; i < high; i++)
        {
            if (P[i].iArrival < pivot)
            {
                vt++;
                swapProcess(&P[vt], &P[i]);
            }
        }
    }
    break;

    case SORT_BY_BURST:
    {
        int pivot = P[high].iBurst;
        int pivot1 = P[high].iArrival;
        for (int i = low; i < high; i++)
        {
            if (P[i].iBurst < pivot)
            {
                vt++;
                swapProcess(&P[vt], &P[i]);
            } else if (P[i].iBurst == pivot  && P[i].iArrival < pivot1)
                {
                    vt++;
                    swapProcess(&P[vt], &P[i]);
                }
        }
    }
    break;
```

```
    case SORT_BY_BURST:
    {
        int pivot = P[high].iBurst;
        int pivot1 = P[high].iArrival;
        for (int i = low; i < high; i++)
        {
            if (P[i].iBurst < pivot)
            {
                vt++;
                swapProcess(&P[vt], &P[i]);
            } else if (P[i].iBurst == pivot  && P[i].iArrival < pivot1)
                {
                    vt++;
                    swapProcess(&P[vt], &P[i]);
                }
        }
    }
    break;

    case SORT_BY_PID:
    {
        int pivot = P[high].iPID;
        for (int i = low; i < high; i++)
        {
            if (P[i].iPID < pivot)
```

```
            {
                vt++;
                swapProcess(&P[vt], &P[i]);
            }
        }
    }
    break;

    case SORT_BY_START:
    {
        int pivot = P[high].iStart;
        for (int i = low; i < high; i++)
        {
            if (P[i].iStart < pivot)
            {
                vt++;
                swapProcess(&P[vt], &P[i]);
            }
        }
    }
    break;

    default:
        break;
    }

    vt++;
    swapProcess(&P[vt], &P[high]);
    return vt;
}
```

```c
void quickSort(PCB P[], int low, int high, int iCriteria)
{
    if (low < high)
    {
        int index = partition(P, low, high, iCriteria);
        quickSort(P, low, index - 1, iCriteria);
        quickSort(P, index + 1, high, iCriteria);
    }
}
```

```c
void calculateAWT(int n, PCB P[])
{
    float awt = 0;
    for(int i=0; i<n; i++)
    {
        awt += P[i].iWaiting;
    }
    awt /= n;
    printf("\nThoi gian doi trung binh: %f", awt);
}

void calculateATaT(int n, PCB P[])
{
    float atat = 0;
    for(int i=0; i<n; i++)
    {
        atat += P[i].iTaT;
    }
    atat /= n;
    printf("\nThoi gian hoan thanh: %f", atat);
    printf("\n");
}
```

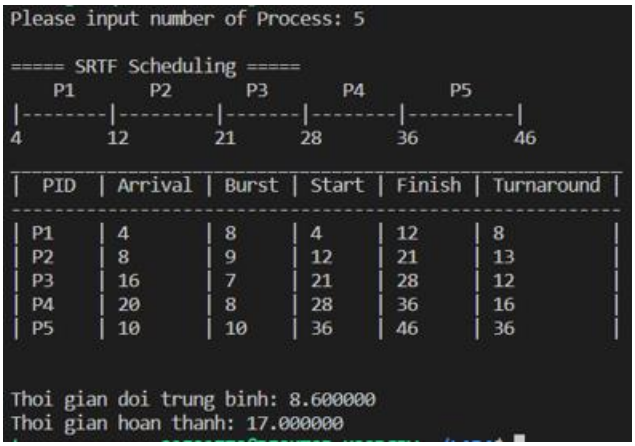
```c
int main()
{
    PCB Input[10];
    PCB ReadyQueue[10];
    PCB TerminatedArray[10];
    int iNumberOfProcess;
    printf("Please input number of Process: ");
    scanf("%d", &iNumberOfProcess);
    int iRemain = iNumberOfProcess, iReady = 0, iTerminated = 0;
    inputProcess(iNumberOfProcess, Input);
    quickSort(Input, 0, iNumberOfProcess - 1, SORT_BY_ARRIVAL);
    pushProcess(&iReady, ReadyQueue, Input[0]);
    removeProcess(&iRemain, 0, Input);
    ReadyQueue[0].iStart =  ReadyQueue[0].iArrival;
    ReadyQueue[0].iFinish = ReadyQueue[0].iStart + ReadyQueue[0].iBurst;
    ReadyQueue[0].iResponse = ReadyQueue[0].iStart - ReadyQueue[0].iArrival;
```

```
ReadyQueue[0].iWaiting =  ReadyQueue[0].iResponse;
ReadyQueue[0].iTaT = ReadyQueue[0].iFinish - ReadyQueue[0].iArrival;

while (iTerminated < iNumberOfProcess)
{
    while (iRemain > 0)
    {
        if (Input[0].iArrival <= ReadyQueue[0].iFinish)
        {
            pushProcess(&iReady, ReadyQueue, Input[0]);
            quickSort(ReadyQueue, 1, iReady - 1, SORT_BY_BURST);
            removeProcess(&iRemain, 0, Input);
            continue;
            }
            else
            break;
            }
            if (iReady > 0)
        {
            pushProcess(&iTerminated, TerminatedArray, ReadyQueue[0]);
            removeProcess(&iReady, 0, ReadyQueue);
            ReadyQueue[0].iStart = TerminatedArray
            [iTerminated - 1].iFinish;
            ReadyQueue[0].iFinish = ReadyQueue[0].iStart
            + ReadyQueue[0].iBurst;
            ReadyQueue[0].iResponse = ReadyQueue[0].iStart
            - ReadyQueue[0].iArrival;
            ReadyQueue[0].iWaiting = ReadyQueue[0].iResponse;
            ReadyQueue[0].iTaT = ReadyQueue[0].iFinish
            - ReadyQueue[0].iArrival;
            }
```

```
            if (iReady == 0 && iRemain > 0)
             {
                 pushProcess(&iReady, ReadyQueue, Input[0]);
                 ReadyQueue[0].iStart = ReadyQueue[0].iArrival;
                 ReadyQueue[0].iFinish = ReadyQueue[0].iStart
                 + ReadyQueue[0].iBurst;
                 ReadyQueue[0].iResponse = ReadyQueue[0].iStart
                  -  ReadyQueue[0].iArrival;
                 ReadyQueue[0].iWaiting = ReadyQueue[0].iResponse;
```

```
                    ReadyQueue[0].iTaT = ReadyQueue[0].iFinish
                    - ReadyQueue[0].iArrival;
                    removeProcess(&iRemain, 0, Input);

        }
}
        printf("\n===== SJF Scheduling =====\n");
        exportGanttChart(iTerminated, TerminatedArray);
        quickSort(TerminatedArray, 0, iTerminated - 1, SORT_BY_PID);
        printProcess(iTerminated, TerminatedArray);
        calculateAWT(iTerminated, TerminatedArray);
        calculateATaT(iTerminated, TerminatedArray);
        return 0;
}
```

- Test case 1:



- Giản đồ Gantt:

P1 P2 P3 P4 P5

4 12 21 28 36 46

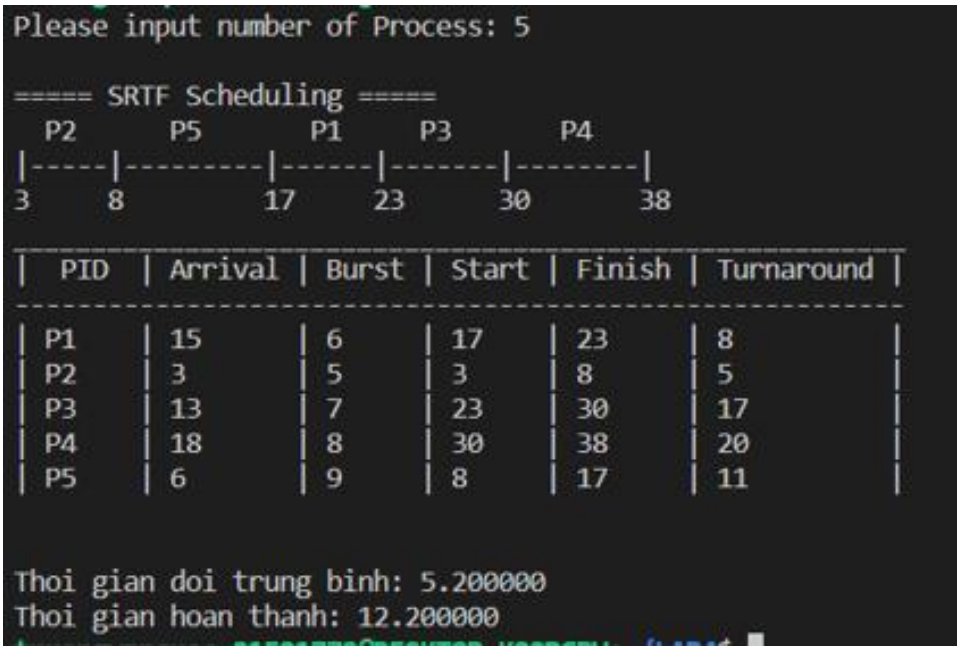
- Thời gian đợi:

+ P1 = 0
+ P2 = 4
+ P3 = 5
+ P4 = 8
+ P5 = 26
+ Thời gian đợi trung bình: awt = 8.6

- Thời gian hoàn thành:

+ P1 = 8
+ P2 = 13
+ P3 = 12
+ P4 = 16
+ P5 = 36
+ Thời gian hoàn thành trung bình: atat = 17

- Test case 2:



- Giản đồ Gantt:

P2 P5 P1 P3 P4

|-------|------------|--------|--------|------------|

5 8 17 23 30 38

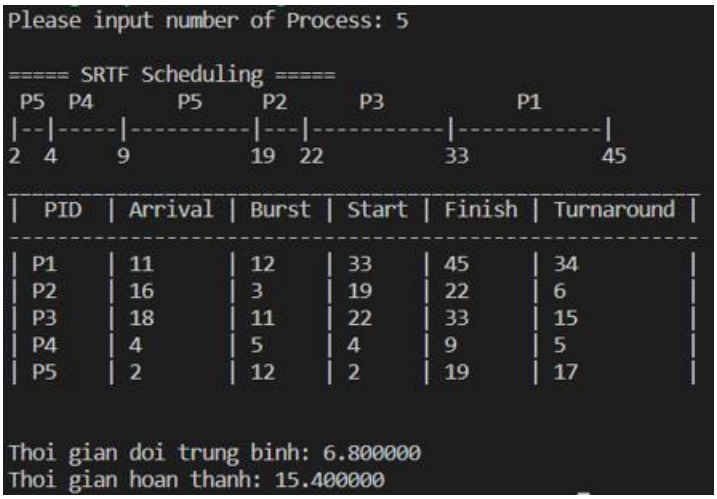
- Thời gian đợi:

+ P1 = 2
+ P2 = 0
+ P3 = 10
+ P4 = 12
+ P5 = 2
+ Thời gian đợi trung bình: awt = 5.2

- Thời gian hoàn thành:

+ P1 = 8
+ P2 = 5
+ P3 = 17
+ P4 = 20
+ P5 = 11
+ Thời gian hoàn thành trung bình: atat = 12.2

- Test case 3:



- Giản đồ Gantt:

P5 P4 P5 P2 P3 P1

2 4 9 19 22 33 45

- Thời gian đợi:

+ P1 = 22
+ P2 = 3
+ P3 = 4
+ P4 = 0
+ P5 = 5
+ Thời gian đợi trung bình: awt = 6.8

- Thời gian hoàn thành:

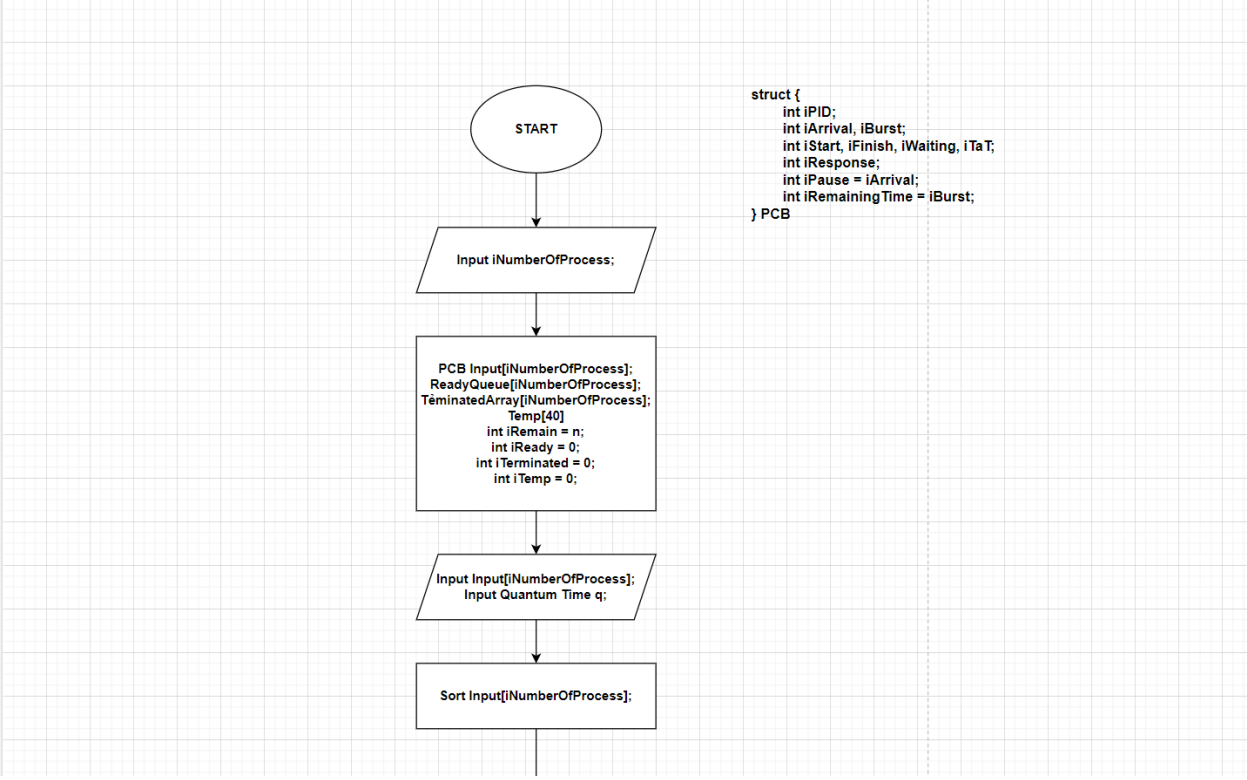
+ P1 = 34
+ P2 = 16
+ P3 = 15
+ P4 = 5
+ P5 = 17
+ Thời gian hoàn thành trung bình: atat = 15.4

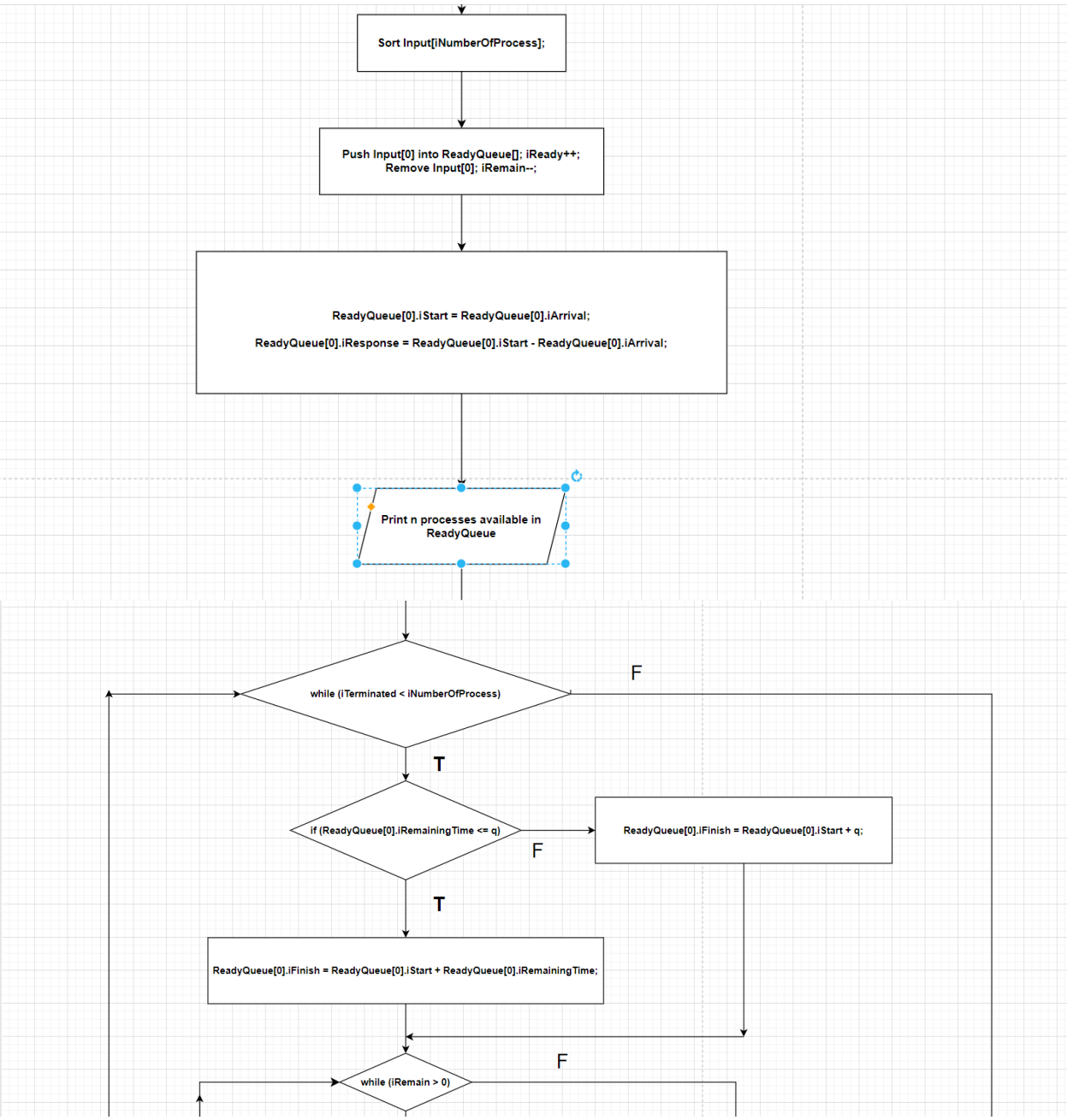
2.6. BÀI TẬP ÔN TẬP

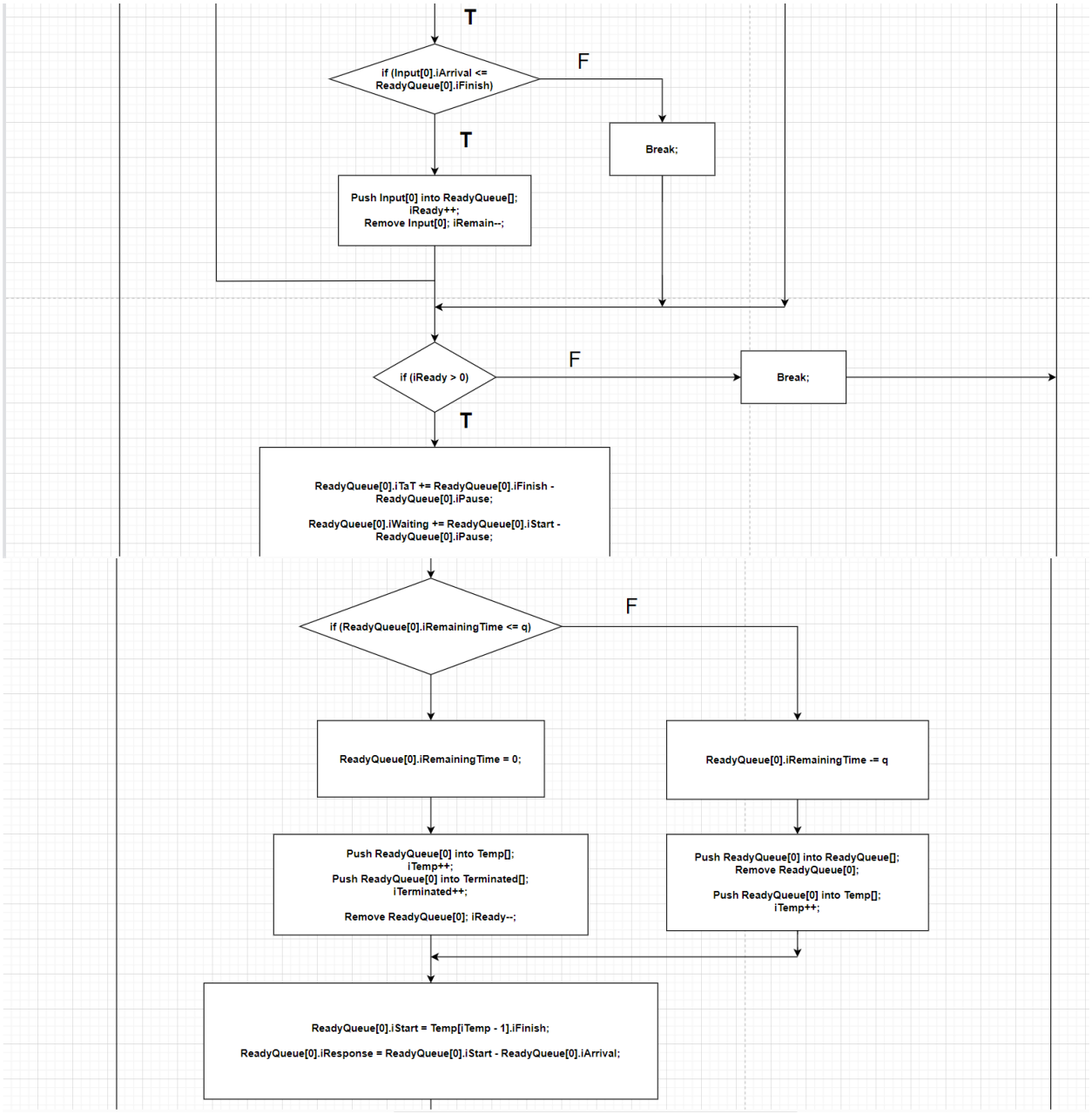
1. Giải thuật Shortest-Remaining-Time-First hoặc Round Robin

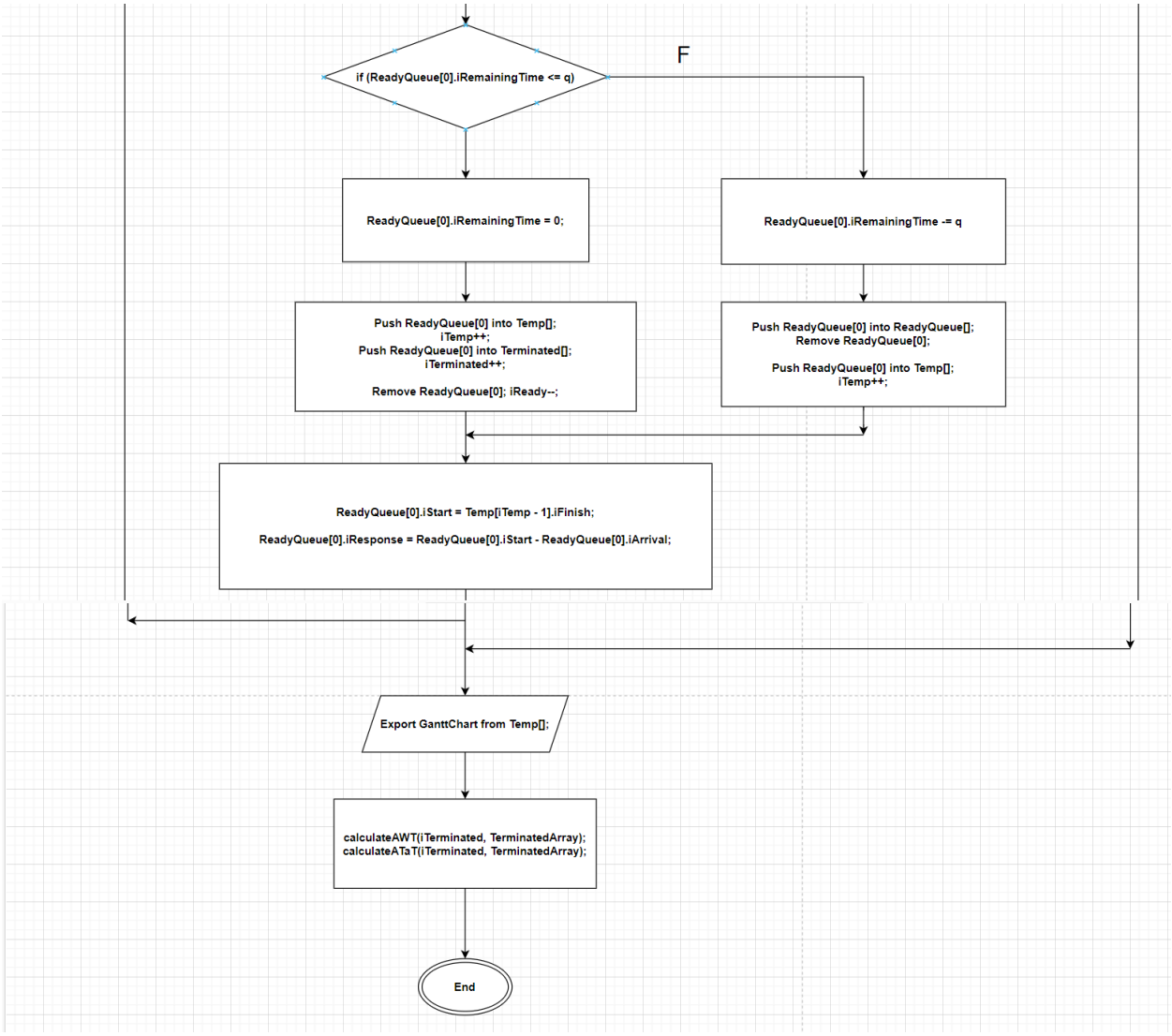
Trả lời: **Round Robin**

- **Lưu đồ:**




Sort Input[iNumberOfProcess];
Push Input[0] into ReadyQueue[]; iReady++;
Remove Input[0]; iRemain--;
ReadyQueue[0].iStart = ReadyQueue[0].iArrival;
ReadyQueue[0].iResponse = ReadyQueue[0].iStart - ReadyQueue[0].iArrival;
Print n processes available in ReadyQueue
while (iTerminated < iNumberOfProcess)
F
T
if (ReadyQueue[0].iRemainingTime <= q)
F
ReadyQueue[0].iFinish = ReadyQueue[0].iStart + q;
T
ReadyQueue[0].iFinish = ReadyQueue[0].iStart + ReadyQueue[0].iRemainingTime;
F
while (iRemain > 0)


T
if (Input[0].iArrival <= ReadyQueue[0].iFinish)
F
T
Break;
Push Input[0] into ReadyQueue[];
iReady++;
Remove Input[0]; iRemain--;
if (iReady > 0)
F
Break;
T
ReadyQueue[0].iTaT += ReadyQueue[0].iFinish - ReadyQueue[0].iPause;
ReadyQueue[0].iWaiting += ReadyQueue[0].iStart - ReadyQueue[0].iPause;
if (ReadyQueue[0].iRemainingTime <= q)
F
ReadyQueue[0].iRemainingTime = 0;
ReadyQueue[0].iRemainingTime -= q
Push ReadyQueue[0] into Temp[];
iTemp++;
Push ReadyQueue[0] into Terminated[];
iTerminated++;
Remove ReadyQueue[0]; iReady--;
Push ReadyQueue[0] into ReadyQueue[];
Remove ReadyQueue[0];
Push ReadyQueue[0] into Temp[];
iTemp++;
ReadyQueue[0].iStart = Temp[iTemp - 1].iFinish;
ReadyQueue[0].iResponse = ReadyQueue[0].iStart - ReadyQueue[0].iArrival;



Link lưu đồ:
https://viewer.diagrams.net/?tags=%7B%7D&highlight=0000ff&edit=_blank&layers=1&nav=1&title=Round%20Robin.drawio#R7V1bc5s4FP41ntk%2BuMMd%2B9HOrd1tmzTxbtJ96WCj2GwwuIATu79%2BxdUgDiA74mInM53GCAGSjr5zl9QTz5abK0dbLb
7aQiJ7AqdyeuJ5TxAGAo%2F%2F9wu2YYFscWHB3DH0sIjfEdwZy1FUGFdbGzpv

- **Test case:**

Cho 5 tiến trình có số liệu như sau (cho q = 4):

Process	Arrival Time	Burst Time
P1	0	6
P2	3	4
P3	7	3
P4	10	5
P5	12	1

Đầu tiên ta có:

Input	Ready Queue	Temp	Terminated
P1 (0,6)			
P2 (3, 4)			
P3 (7, 3)			
P4 (10, 5)			
P5 (12, 1)			
iRemain = 5	iReady = 0	iTemp = 0	iTerminated = 0

P1 được vào hàng chờ Ready:

Input	Ready Queue	Temp	Terminated
P2 (3, 4)	P1 (0, 6)		
P3 (7, 3)			
P4 (10, 5)			
P5 (12, 1)			
iRemain = 4	iReady = 1	iTemp = 0	iTerminated = 0

Vì burst time của P1 (iBurst = 6) lớn hơn quantum time (q = 4) nên P1 sẽ thực hiện trong khoảng thời gian tối đa là 4 giây. Như vậy, thời điểm kết thúc của P1 là 4.

Trong khoảng thời gian từ giây 0 đến giây 4 có P2 xuất hiện (iArrival = 3). Lúc này P2 sẽ được thêm vào hàng chờ Ready

Input	Ready Queue	Temp	Terminated

P3 (7, 3)	P1 (0, 6)		
P4 (10, 5)	P2 (3, 4)		
P5 (12, 1)			
iRemain = 3	iReady = 2	iTemp = 0	iTerminated = 0

Sau khi P1 thực hiện xong sẽ được đẩy xuống cuối của hàng chờ Ready, đồng thời sẽ cập nhật lại iRemainingTime = 2 (đã thực hiện 4 giây trước đó), iPause sẽ đánh dấu lần vào hàng chờ Ready lần thứ 2 tại giây thứ 4. P1 cũng sẽ được đẩy qua hàng Temp (các chỉ số thể hiện ở cột Temp là **(iStart, iFinish)**, ở các cột còn lại là (iArrival, iBurst)):

Input	Ready Queue	Temp	Terminated
P3 (7, 3)	P2 (3, 4)	P1 (0, 4)	
P4 (10, 5)	P1 (4, 2)		
P5 (12, 1)			
iRemain = 3	iReady = 2	iTemp = 1	iTerminated = 0

P2.iStart = P1.iFinish (= 4)

Tương tự như trên, các tiến trình sẽ thực thi tuần tự đến khi iTerminated = iNumberOfProcess = 5:

Input	Ready Queue	Temp	Terminated
P4 (10, 5)	P2 (3, 4)	P1 (0, 4)	
P5 (12, 1)	P1 (4, 2)		
	P3 (7, 3)		
iRemain = 2	iReady = 3	iTemp = 1	iTerminated = 0

Input	Ready Queue	Temp	Terminated
P4 (10, 5)	P1 (4, 2)	P1 (0, 4)	P2 (3, 4)
P5 (12, 1)	P3 (7, 3)	P2 (4, 8)	

iRemain = 2	iReady = 2	iTemp = 2	iTerminated = 1

Input	Ready Queue	Temp	Terminated
P4 (10, 5)	P3 (7, 3)	P1 (0, 4)	P2 (3, 4)
P5 (12, 1)		P2 (4, 8)	P1 (0, 6)
		P1 (8, 10)	
iRemain = 2	iReady = 1	iTemp = 3	iTerminated = 2

Input	Ready Queue	Temp	Terminated
	P3 (7, 3)	P1 (0, 4)	P2 (3, 4)
	P4 (10, 5)	P2 (4, 8)	P1 (0, 6)
	P5 (12, 1)	P1 (8, 10)	
iRemain = 0	iReady = 3	iTemp = 3	iTerminated = 2

Input	Ready Queue	Temp	Terminated
	P3 (7, 3)	P1 (0, 4)	P2 (3, 4)
	P4 (10, 5)	P2 (4, 8)	P1 (0, 6)
	P5 (12, 1)	P1 (8, 10)	
iRemain = 0	iReady = 3	iTemp = 3	iTerminated = 2

Input	Ready Queue	Temp	Terminated
	P4 (10, 5)	P1 (0, 4)	P2 (3, 4)
	P5 (12, 1)	P2 (4, 8)	P1 (0, 6)
		P1 (8, 10)	P3 (7, 3)
		P3 (10, 13)	
iRemain = 0	iReady = 2	iTemp = 4	iTerminated = 3

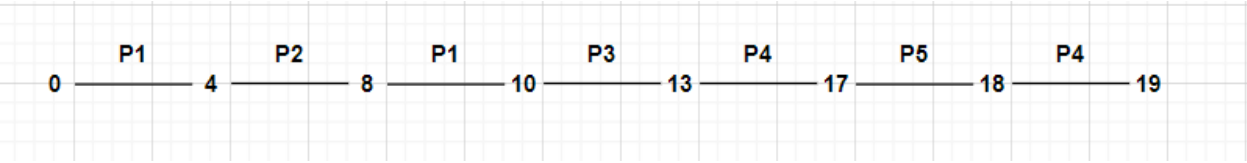
Input	Ready Queue	Temp	Terminated
	P5 (12, 1)	P1 (0, 4)	P2 (3, 4)

	P4 (17, 1)	P2 (4, 8)	P1 (0, 6)
		P1 (8, 10)	P3 (7, 3)
		P3 (10, 13)	
		P4 (13, 17)	
iRemain = 0	iReady = 2	iTemp = 5	iTerminated = 3

Input	Ready Queue	Temp	Terminated
	P4 (17, 1)	P1 (0, 4)	P2 (3, 4)
		P2 (4, 8)	P1 (0, 6)
		P1 (8, 10)	P3 (7, 3)
		P3 (10, 13)	P5 (12, 1)
		P4 (13, 17)	
		P5 (17, 18)	
iRemain = 0	iReady = 1	iTemp = 6	iTerminated = 4

Input	Ready Queue	Temp	Terminated
		P1 (0, 4)	P2 (3, 4)
		P2 (4, 8)	P1 (0, 6)
		P1 (8, 10)	P3 (7, 3)
		P3 (10, 13)	P5 (12, 1)
		P4 (13, 17)	P4 (10, 5)
		P5 (17, 18)	
		P4 (18, 19)	
iRemain = 0	iReady = 0	iTemp = 6	iTerminated = 5

Cuối cùng ta sẽ vẽ biểu đồ Gantt dựa vào số liệu và thứ tự tiến trình của cột Temp, tính Turnaround Time trung bình và Waiting Time trung bình dựa vào cột Terminated.



iATaT = (10+5+6+9+6)/5 = 7.2

iAWT = (4+1+3+4+5)/5 = 3.4

- **Code:**

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <stdbool.h>
#include <limits.h>
#define SORT_BY_ARRIVAL 0
#define SORT_BY_PID 1
#define SORT_BY_BURST 2
#define SORT_BY_START 3

typedef struct{
    int iPID;
    int iArrival, iBurst;
    int iStart, iFinish, iWaiting, iResponse, iTaT;
    int iPause, iRemaining;
} PCB;

void inputProcess(int n, PCB P[]) {
    srand((unsigned) time(NULL));

    for (int i = 0; i < n; i++){
         P[i].iPID= i+1;
        P[i].iArrival = rand() % 21;
        P[i].iBurst = (rand()% 11) +2;
        P[i].iRemaining = P[i].iBurst;
        P[i].iPause = P[i].iArrival;
        P[i].iStart = P[i].iFinish = P[i].iResponse = P[i].iTaT =
P[i].iWaiting = 0;
    }
}

void printProcess(int n, PCB P[]) {
    for (int i = 0; i < n; i++){
        printf("P%d(%d, %d)\n", P[i].iPID, P[i].iArrival, P[i].iBurst);
    }
}

void exportGanttChart (int n, PCB P[]) {
    printf("Gantt Chart:\n");
    printf("%d", P[0].iStart);
    for (int i = 0; i < n; i++){
        printf("___P[%d]___%d", P[i].iPID, P[i].iFinish);
    }
```

```
    printf("\n");
}

void pushProcess(int *n, PCB P[], PCB Q) {
    P[*n] = Q;
    (*n)++;
}

void removeProcess(int *n, int index, PCB P[]) {
    for(int i = index; i < *n; i++){
        P[i]=P[i+1];
    }
    (*n)--;
}
void swapProcess(PCB *P, PCB *Q) {
    PCB temp = *P;
    *P = *Q;
    *Q = temp;
}

int partition (PCB P[], int low, int high, int iCriteria) {
    if (iCriteria == SORT_BY_ARRIVAL) {
        int pivot = P[high].iArrival;

        int i = (low-1);

        for(int j = low; j <= high; j++)
        {
            if (P[j].iArrival < pivot)
            {
                i++;
                swapProcess(&P[i], &P[j]);
            }
        }
        swapProcess(&P[i+1], &P[high]);
        return (i+1);
    } else if (iCriteria == SORT_BY_PID) { // Sort by PID
        int pivot = P[high].iPID;

        int i = (low-1);

        for(int j = low; j <= high; j++)
        {
            if (P[j].iPID < pivot)
```

```
            {
                i++;
                swapProcess(&P[i], &P[j]);
            }
        }
        swapProcess(&P[i+1], &P[high]);
        return (i+1);
    } else if (iCriteria == SORT_BY_BURST) {
        int pivot = P[high].iBurst;

        int i = (low-1);

        for(int j = low; j <= high; j++)
        {
            if (P[j].iBurst < pivot)
            {
                i++;
                swapProcess(&P[i], &P[j]);
            }
        }
        swapProcess(&P[i+1], &P[high]);
        return (i+1);
    } else if (iCriteria == SORT_BY_START) {
        int pivot = P[high].iStart;

        int i = (low-1);

        for(int j = low; j <= high; j++)
        {
            if (P[j].iStart < pivot)
            {
                i++;
                swapProcess(&P[i],&P[j]);
            }
        }
        swapProcess(&P[i+1], &P[high]);
        return (i+1);
    }
}

void quickSort(PCB P[], int low, int high, int iCriteria) {
    if(low < high) {

    int pi = partition(P, low, high, iCriteria);
```

```
    quickSort(P, low, pi-1, iCriteria);
    quickSort(P, pi+1, high, iCriteria);
    }
}

void calculateAWT(int n, PCB P[]) {
    int sum = 0;
    float avg = 0;
    for (int i = 0; i < n; i++) {
        sum += P[i].iWaiting;
    }
    if (n > 0) avg = (float) sum/n;
    printf("Average Waiting time: %f\n", avg);
}

void calculateATaT(int n, PCB P[]) {
    int sum = 0;
    float avg = 0;
    for (int i = 0; i < n; i++) {
        sum += P[i].iTaT;
    }
    if (n > 0) avg = (float) sum/n;
    printf("Average Turnaround time: %f\n", avg);
}

int main()
{
    PCB Input[10];
    PCB ReadyQueue[10];
    PCB Temp[40];
    PCB TerminatedArray[10];
    int iNumberOfProcess;
    printf("Please input number of Process: ");
    scanf("%d", &iNumberOfProcess);
    int iRemain = iNumberOfProcess, iReady = 0, iTerminated = 0, iTemp =
0;
    inputProcess(iNumberOfProcess, Input);
    quickSort(Input, 0, iNumberOfProcess - 1, SORT_BY_ARRIVAL);
    printf("--Input--\n");
    printProcess(iRemain, Input);
    int q;
    printf("\nPlease input quantum time: ");
    scanf("%d", &q);
```

```
    pushProcess(&iReady, ReadyQueue, Input[0]);
    removeProcess(&iRemain, 0, Input);
    ReadyQueue[0].iStart = ReadyQueue[0].iArrival;
    ReadyQueue[0].iResponse = ReadyQueue[0].iStart -
ReadyQueue[0].iArrival;
    bool flag=false;
    while (iTerminated < iNumberOfProcess) {
        if (ReadyQueue[0].iRemaining <= q)
            ReadyQueue[0].iFinish = ReadyQueue[0].iStart +
ReadyQueue[0].iRemaining;
        else
            ReadyQueue[0].iFinish = ReadyQueue[0].iStart + q;
        while (iRemain > 0) {
            if (Input[0].iArrival <= ReadyQueue[0].iFinish) {
                flag=false;
                pushProcess(&iReady, ReadyQueue, Input[0]);
                removeProcess(&iRemain, 0, Input);
                continue;
            }
            else{
                     if(iReady==0){
                    flag=true;
                    pushProcess(&iReady, ReadyQueue, Input[0]);
                     removeProcess(&iRemain, 0, Input);
                     break;
                     }
                   else {flag=false;break;}
            }
        }
        if (iReady > 0) {
            ReadyQueue[0].iTaT += ReadyQueue[0].iFinish -
ReadyQueue[0].iPause;
            ReadyQueue[0].iWaiting += ReadyQueue[0].iStart -
ReadyQueue[0].iPause;
            ReadyQueue[0].iPause = ReadyQueue[0].iFinish;
            if (ReadyQueue[0].iRemaining <= q) {
                ReadyQueue[0].iRemaining = 0;
                pushProcess(&iTemp, Temp, ReadyQueue[0]);
                pushProcess(&iTerminated, TerminatedArray, ReadyQueue[0]);
                removeProcess(&iReady, 0, ReadyQueue);
            }
            else {
                ReadyQueue[0].iRemaining -= q;
                pushProcess(&iTemp, Temp, ReadyQueue[0]);
```

```
                pushProcess(&iReady, ReadyQueue, ReadyQueue[0]);
                removeProcess(&iReady, 0, ReadyQueue);
            }
            if(flag)  ReadyQueue[0].iStart = ReadyQueue[0].iArrival;
            else ReadyQueue[0].iStart = Temp[iTemp-1].iFinish;
            ReadyQueue[0].iResponse = ReadyQueue[0].iStart -
  ReadyQueue[0].iArrival;
        }
    }
    printProcess(iTerminated, TerminatedArray);
    printf("\n===== Round Robin Scheduling =====\n");
    exportGanttChart(iTemp, Temp);
    calculateAWT(iTerminated, TerminatedArray);
    calculateATaT(iTerminated, TerminatedArray);
    return 0;
}
```

Test case 1:

```
PS D:\Study\Classes\HK3\HDH\ThucHanh\Lab4> gcc rr.c -o rr
PS D:\Study\Classes\HK3\HDH\ThucHanh\Lab4> ./rr
Please input number of Process: 5
--Input--
P2(0, 12)
P1(5, 6)
P4(10, 8)
P5(12, 10)
P3(13, 4)

Please input quantum time: 3
P1(5, 6)
P3(13, 4)
P4(10, 8)
P5(12, 10)

===== Round Robin Scheduling =====
Gantt Chart:
0___P[2]___3___P[2]___6___P[1]___9___P[2]___12___P[1]___15___P[4]___18___P[5]___21___P[2]___24___P[3]___27___P[4]___30___P[5]___33___P[3]___34___P[4]___36_
__P[5]___39___P[5]___40
Average Waiting time: 13.800000
Average Turnaround time: 21.799999
```

Test case 2:

```
PS D:\Study\Classes\HK3\HDH\ThucHanh\Lab4> ./rr
Please input number of Process: 5
--Input--
P1(2, 8)
P4(6, 12)
P3(17, 10)
P2(18, 4)
P5(19, 9)

Please input quantum time: 4
P1(2, 8)
P2(18, 4)
P4(6, 12)
P3(17, 10)
P5(19, 9)

===== Round Robin Scheduling =====
Gantt Chart:
2___P[1]___6___P[4]___10___P[1]___14___P[4]___18___P[3]___22___P[2]___26___P[4]___30___P[5]___34___P[3]___38___P[5]___42___P[3]___44___P[5]___45
Average Waiting time: 10.800000
Average Turnaround time: 19.400000
```

Test case 3:

```
PS D:\Study\Classes\HK3\HĐH\ThucHanh\Lab4> ./rr
Please input number of Process: 5
--Input--
P4(2, 7)
P5(5, 8)
P2(9, 7)
P1(19, 7)
P3(20, 3)

Please input quantum time: 5
P4(2, 7)
P5(5, 8)
P2(9, 7)
P3(20, 3)
P1(19, 7)

===== Round Robin Scheduling =====
Gantt Chart:
2___P[4]___7___P[5]___12___P[4]___14___P[2]___19___P[5]___22___P[1]___27___P[2]___29___P[3]___32___P[1]___34
Average Waiting time: 8.800000
Average Turnaround time: 15.200000
```