





O's – Scheduling

The background is a dark gray. There are two large white circles. One is in the top-left corner, partially cut off. The other is in the center-right, containing the text.

Using Kotlin
via Eclipse Java Neon



To Simulate How
'Os'
Schedules processes

```
First FirstC FirstC FirstComeFirstServe.kt ShortestJobFirst.kt PriorityQueue.kt RoundRobin.kt
41 43 1 1 package FirstComeFirstServe
42 44 2 2
43 45 3 3 var btList: MutableList<Process> = mutableListOf()
44 46 4 4 var startTime: MutableList<Double> = mutableListOf()
45 47 5 5 var totalTime = 0
46 48 6 6
47 49 7 7 fun main(args: Array<String>) {
48 50 8 8     init()
49 51 9 9     process()
50 52 10 10     summarize()
51 53 11 11 }
52 54 12 12
53 55 13 13 fun init() {
54 56 14 14     print("Number of process:")
55 57 15 15     val nProcess = readLine()!!.toInt()
56 58 16 16
57 59 17 17     var arriveAt = 0
58 60 18 18     print("Enter burst time for each process\n")
59 61 19 19     print("%3s %9s %16s".format("At", "Process", "Burst time(ms)\n"))
60 62 20 20     for (i in arriveAt..(nProcess - 1)) {
61 63 21 21         print("%3s %8s %13s".format("%i", "P${i+1}", ""))
62 64 22 22         var currBurstTime = readLine()!!.toInt()
63 65 23 23         btList.add(Process(i, i, currBurstTime))
64 66 24 24     }
65 67 25 25     print("\n")
66 68 26 26 }
67 69 27 27
68 70 28 28 fun process() {
69 71 29 29     print("Scheduling scheme: First Come First Serve\n")
70 72 30 30     for (i in 0..(btList.size - 1)) {
71 73 31 31         startTime.add(totalTime.toDouble())
72 74 32 32         print("|")
73 75 33 33         for (j in 0..btList[i].burstTime - 1) {
74 76 34 34             totalTime += 1
75 77 35 35             print(if (j == btList[i].burstTime / 2) "P${i+1}" else "-")
76 78 36 36         }
77 79 37 37         print("|")
78 80 38 38     }
79 81 39 39     print("\n\n")
80 82 40 40 }
81 83 41 41
82 84 42 42 fun summarize() {
83 85 43 43     var waitTimeList: MutableList<Double> = mutableListOf()
84 86 44 44     print("In summary:\n")
85 87 45 45     for (i in 0..(btList.size - 1)) {
86 88 46 46         val waitTime = startTime[i] - btList[i].arrivalTime
87 89 47 47         waitTimeList.add(waitTime.toDouble())
88 90 48 48         print("P${i+1} waited %5.2f ms\n".format(waitTime))
89 91 49 49     }
90 92 50 50     var waitTimeTotal = waitTimeList.sum()
91 93 51 51     print("Since there are ${btList.size} processes,\n")
92 94 52 52     print("The average waiting time is ${waitTimeTotal / btList.size} ms")
93 95 53 53 }
94 96 54 54
95 97 55 55
96 98 56 56
97 99 57 57
98 100 58 58
99 101 59 59
100 102 60 60
101 103 61 61
102 104 62 62
103 105 63 63
```

1.First Come First Serve

```
First FirstC FirstC FirstComeFirstServe.kt ShortestJobFirst.kt PriorityQueue.kt RoundRobin.kt
41 43 1 1 package FirstComeFirstServe
42 44 2 2
43 45 3 3 var btList: MutableList<Process> = mutableListOf()
44 46 4 4 var startTime: MutableList<Double> = mutableListOf()
45 47 5 5 var totalTime = 0
46 48 6 6
47 49 7 7 fun main(args: Array<String>) {
48 50 8 8     init()
49 51 9 9     process()
50 52 10 10     summarize()
51 53 11 11 }
52 54 12 12
53 55 13 13 fun init() {
54 56 14 14     print("Number of process:")
55 57 15 15     val nProcess = readLine()!!.toInt()
56 58 16 16
57 59 17 17     var arriveAt = 0
58 60 18 18     print("Enter burst time for each process\n")
59 61 19 19     print("%3s %9s %16s".format("At", "Process", "Burst time(ms)\n"))
60 62 20 20     for (i in arriveAt..(nProcess - 1)) {
61 63 21 21         print("%3s %8s %13s".format("%i", "P${i+1}", ""))
62 64 22 22         var currBurstTime = readLine()!!.toInt()
63 65 23 23         btList.add(Process(i, i, currBurstTime))
64 66 24 24     }
65 67 25 25     print("\n")
66 68 26 26 }
67 69 27 27
68 70 28 28 fun process() {
69 71 29 29     print("Scheduling scheme: First Come First Serve\n")
70 72 30 30     for (i in 0..(btList.size - 1)) {
71 73 31 31         startTime.add(totalTime.toDouble())
72 74 32 32         print("|")
73 75 33 33         for (j in 0..btList[i].burstTime - 1) {
74 76 34 34             totalTime += 1
75 77 35 35             print(if (j == btList[i].burstTime / 2) "P${i+1}" else "-")
76 78 36 36         }
77 79 37 37         print("|")
78 80 38 38     }
79 81 39 39     print("\n\n")
80 82 40 40 }
81 83 41 41
82 84 42 42 fun summarize() {
83 85 43 43     var waitTimeList: MutableList<Double> = mutableListOf()
84 86 44 44     print("In summary:\n")
85 87 45 45     for (i in 0..(btList.size - 1)) {
86 88 46 46         val waitTime = startTime[i] - btList[i].arrivalTime
87 89 47 47         waitTimeList.add(waitTime.toDouble())
88 90 48 48         print("P${i+1} waited %5.2f ms\n".format(waitTime))
89 91 49 49     }
90 92 50 50     var waitTimeTotal = waitTimeList.sum()
91 93 51 51     print("Since there are ${btList.size} processes,\n")
92 94 52 52     print("The average waiting time is ${waitTimeTotal / btList.size} ms")
93 95 53 53 }
94 96 54 54
95 97 55 55
96 98 56 56
97 99 57 57
98 100 58 58
99 101 59 59
100 102 60 60
101 103 61 61
102 104 62 62
103 105 63 63
```

The key
is to execute the process
in order of their arrival

```

1 package FirstComeFirstServe
2
3 var btList: MutableList<Process> = mutableListOf()
4 var startTime: MutableList<Double> = mutableListOf()
5 var totaltime = 0
6
7 fun main(args: Array<String>) {
8     init()
9     process()
10    summarize()
11 }
12
13 fun init() {
14     print("Number of process:")
15     val nProcess = readLine()!!.toInt()
16
17     var arriveAt = 0
18     print("Enter burst time for each process\n")
19     print("%3s %9s %16s".format("At", "Process", "Burst time(ms)\n"))
20     for (i in arriveAt..(nProcess - 1)) {
21         print("%3s %8s %13s".format("$i", "P${i+1}", ""))
22         var currBurstTime = readLine()!!.toInt()
23         btList.add(Process(i, i, currBurstTime))
24     }
25     print("\n")
26 }
27
28 fun process() {
29     print("Scheduling scheme: First Come First Serve\n")
30     for (i in 0..(btList.size - 1)) {
31         startTime.add(totaltime.toDouble())
32         print("|")
33         for (j in 0..btList[i].burstTime - 1) {
34             totaltime += 1
35             print(if (j == btList[i].burstTime / 2) "P${i+1}" else "-")
36         }
37         print("|")
38     }
39     print("\n\n")
40 }
41

```

As soon as the process arrives..

...

We directly follows the order its come in

The waiting time could be calculate as follows

Number of process:4

Enter burst time for each process

At	Process	Burst time(ms)
0	P1	5
1	P2	8
2	P3	2
3	P4	4

Scheduling scheme: First Come First Serve

|--P1--||----P2---||-P3||--P4-|

In summary:

P1 waited 0.00 ms

P2 waited 4.00 ms

P3 waited 11.00 ms

P4 waited 12.00 ms

Since there are 4 processes,

The average waiting time is 6.75 ms

The waiting could be calculate as follows

Finished time

—

(Arrival time
+ Burst time)

Number of process:4

Enter burst time for each process

At	Process	Burst time(ms)
----	---------	----------------

0	P1	5
---	----	---

1	P2	8
---	----	---

2	P3	2
---	----	---

3	P4	4
---	----	---

Scheduling scheme: First Come First Serve

|--P1--||----P2---||-P3||--P4-|

In summary:

P1 waited 0.00 ms

P2 waited 4.00 ms

P3 waited 11.00 ms

P4 waited 12.00 ms

Since there are 4 processes,

The average waiting time is 6.75 ms


```

41 43 1 package ShortestJobFirst
42 44 2
43 45 3 var arrivedList: MutableList<Process> = mutableListOf()
44 46 4 var btList: MutableList<Process> = mutableListOf()
45 47 5
46 48 6 var finishTime: MutableList<Double> = mutableListOf()
47 49 7
48 50 8 var totalTime = 0
49 51 9
50 52 10
51 53 11 fun main(args: Array<String>) {
52 54 12     init()
53 55 13     process()
54 56 14     summarize()
55 57 15 }
56 58 16
57 59 17 fun init() {
58 60 18     print("Number of process:")
59 61 19     val nProcess = readLine()!!.toInt()
60 62 20
61 63 21     var arriveAt = 0
62 64 22     print("Enter burst time for each process\n")
63 65 23     print("%3s %9s %16s".format("At", "Process", "Burst time(ms)\n"))
64 66 24     for (i in arriveAt..(nProcess - 1)) {
65 67 25         print("%3s %8s %13s".format("$i", "P${i + 1}", ""))
66 68 26         var currBurstTime = readLine()!!.toInt()
67 69 27         btList.add(Process(i, i + 1, currBurstTime))
68 70 28         totalTime += currBurstTime
69 71 29         finishTime.add(0.0)
70 72 30     }
71 73 31     print("\n\n")
72 74 32     // print("total time = $totalTime")
73 75 33 }
74 76 34
75 77 35 fun process() {
76 78 36     print("Scheduling scheme: Shortest Job First\n")
77 79 37
78 80 38     // store a string that will print after everything is finished
79 81 39     var output: MutableList<String> = mutableListOf()
80 82 40
81 83 41     //loop through every single turn
82 84 42     for (i in 0..totalTime - 1) {
83 85 43         // arrived
84 86 44         if (i < btList.size) {
85 87 45             arrivedList.add(btList[i].copy())
86 88 46         }
87 89 47
88 90 48         var minIndex = 0
89 91 49         // find shortest job in arrivedList
90 92 50         // skipped if only one process is left
91 93 51         for (j in 0..arrivedList.size - 1) {
92 94 52             if ((arrivedList.size != 1) && (arrivedList[minIndex].burstTime > arrivedList[j].burstTime)) {
93 95 53                 minIndex = j
94 96 54             }
95 97 55         }
96 98 56         // print shortest job
97 99 57         output.add("P${arrivedList[minIndex].processNum}")
98 100 58
99 101 59
100 102 60         // remove the process if remaining bursttime = 0
101 103 61         arrivedList[minIndex].burstTime -= 1
102 104 62         if (arrivedList[minIndex].burstTime == 0) {
103 105 63             finishTime[arrivedList[minIndex].processNum - 1] = i+1.toDouble()

```

2.Shortest Job First

```

41 43 1 package ShortestJobFirst
42 44 2
43 45 3 var arrivedList: MutableList<Process> = mutableListOf()
44 46 4 var btList: MutableList<Process> = mutableListOf()
45 47 5
46 48 6 var finishTime: MutableList<Double> = mutableListOf()
47 49 7
48 50 8 var totalTime = 0
49 51 9
50 52 10
51 53 11 fun main(args: Array<String>) {
52 54 12     init()
53 55 13     process()
54 56 14     summarize()
55 57 15 }
56 58 16
57 59 17 fun init() {
58 60 18     print("Number of process:")
59 61 19     val nProcess = readLine()!!.toInt()
60 62 20
61 63 21     var arriveAt = 0
62 64 22     print("Enter burst time for each process\n")
63 65 23     print("%3s %9s %16s".format("At", "Process", "Burst time(ms)\n"))
64 66 24     for (i in arriveAt..(nProcess - 1)) {
65 67 25         print("%3s %8s %13s".format("$i", "P${i + 1}", ""))
66 68 26         var currBurstTime = readLine()!!.toInt()
67 69 27         btList.add(Process(i, i + 1, currBurstTime))
68 70 28         totalTime += currBurstTime
69 71 29         finishTime.add(0.0)
70 72 30     }
71 73 31     print("\n\n")
72 74 32     // print("total time = $totalTime")
73 75 33 }
74 76 34
75 77 35 fun process() {
76 78 36     print("Scheduling scheme: Shortest Job First\n")
77 79 37
78 80 38     // store a string that will print after everything is finished
79 81 39     var output: MutableList<String> = mutableListOf()
80 82 40
81 83 41     //loop through every single turn
82 84 42     for (i in 0..totalTime - 1) {
83 85 43         // arrived
84 86 44         if (i < btList.size) {
85 87 45             arrivedList.add(btList[i].copy())
86 88 46         }
87 89 47
88 90 48         var minIndex = 0
89 91 49         // find shortest job in arrivedList
90 92 50         // skipped if only one process is left
91 93 51         for (j in 0..arrivedList.size - 1) {
92 94 52             if ((arrivedList.size != 1) && (arrivedList[minIndex].burstTime > arrivedList[j].burstTime)) {
93 95 53                 minIndex = j
94 96 54             }
95 97 55         }
96 98 56         // print shortest job
97 99 57         output.add("P${arrivedList[minIndex].processNum}")
98 100 58
99 101 59
100 102 60         // remove the process if remaining bursttime = 0
101 103 61         arrivedList[minIndex].burstTime -= 1
102 104 62         if (arrivedList[minIndex].burstTime == 0) {
103 105 63             finishTime[arrivedList[minIndex].processNum - 1] = i+1.toDouble()

```

Focuses on
shortest process first,
before actually touching
time-consuming ones

```

13 process()
14     summarize()
15 }
16
17 fun init() {
18     print("Number of process:")
19     val nProcess = readLine()!!.toInt()
20
21     var arriveAt = 0
22     print("Enter burst time for each process\n")
23     print("%3s %9s %16s".format("At", "Process", "Burst time(ms)\n"))
24     for (i in arriveAt..(nProcess - 1)) {
25         print("%3s %8s %13s".format("$i", "P${i + 1}", ""))
26         var currBurstTime = readLine()!!.toInt()
27         btList.add(Process(i, i + 1, currBurstTime))
28         totalTime += currBurstTime
29         finishTime.add(0.0)
30     }
31     print("\n\n")
32     // print("total time = $totaltime")
33 }
34
35 fun process() {
36     print("Scheduling scheme: Shortest Job First\n")
37
38     // store a string that will print after everything is finished
39     var output: MutableList<String> = mutableListOf()
40
41     //loop through every single turn
42     for (i in 0..totalTime - 1) {
43         // arrived
44         if (i < btList.size) {
45             arrivedList.add(btList[i].copy())
46         }
47
48         var minIndex = 0
49         // find shortest job in arrivedList
50         // skipped if only one process is left
51         for (j in 0..arrivedList.size - 1) {
52             if ((arrivedList.size != 1) && (arrivedList[minIndex].burstTime > arrivedList[j].burstTime)) {
53                 minIndex = j
54             }
55         }
56         // print shortest job
57         output.add("P${arrivedList[minIndex].processNum}")
58     }
59 }

```

Loops through the process pool
to pick the shortest one

Since P1 burst time is 6,
as P2 arrives
with less burst time.

The algorithm instantly shift to work on P2

Number of process:5
Enter burst time for each process

At	Process	Burst time(ms)
0	P1	6
1	P2	4
2	P3	1
3	P4	3
4	P5	5

Scheduling scheme: Shortest Job First

|P1||P2||P3||-P2-||-P4-||--P1--||--P5--|

In summary:

P1 waited 8.00 ms

P2 waited 1.00 ms

P3 waited 0.00 ms

P4 waited 3.00 ms

P5 waited 10.00 ms

Since there are 5 processes,

The average waiting time is 4.4 ms

Same goes for when P3 arrives.

After the P3 is done,
Automatically looks for the next shortest
job
that is left in the pool

Number of process:5
Enter burst time for each process

At	Process	Burst time(ms)
0	P1	6
1	P2	4
2	P3	1
3	P4	3
4	P5	5

Scheduling scheme: Shortest Job First

|P1||P2||P3||-P2-||-P4-||--P1--||--P5--|

In summary:

P1 waited 8.00 ms

P2 waited 1.00 ms

P3 waited 0.00 ms

P4 waited 3.00 ms

P5 waited 10.00 ms

Since there are 5 processes,

The average waiting time is 4.4 ms

The waiting time can be calculate the same
way as the first algorithm

Number of process:5
Enter burst time for each process

At	Process	Burst time(ms)
0	P1	6
1	P2	4
2	P3	1
3	P4	3
4	P5	5

Scheduling scheme: Shortest Job First

|P1||P2||P3||-P2-||-P4-||--P1--||--P5--|

In summary:

P1 waited 8.00 ms

P2 waited 1.00 ms

P3 waited 0.00 ms

P4 waited 3.00 ms

P5 waited 10.00 ms

Since there are 5 processes,

The average waiting time is 4.4 ms

The waiting time can be calculate the same way as the first algorithm

Finished time
—
(Arrival time
+ Burst time)

Number of process:5
Enter burst time for each process

At	Process	Burst time(ms)
0	P1	6
1	P2	4
2	P3	1
3	P4	3
4	P5	5

Scheduling scheme: Shortest Job First

|P1||P2||P3||-P2-||-P4-||--P1--||--P5--|

In summary:

P1 waited 8.00 ms

P2 waited 1.00 ms

P3 waited 0.00 ms

P4 waited 3.00 ms

P5 waited 10.00 ms

Since there are 5 processes,

The average waiting time is 4.4 ms

```

41 fun process() {
42     print("Scheduling scheme: Priority Queue\n")
43
44     // store a string that will print after everything is finished
45     var output: MutableList<String> = mutableListOf()
46
47     //loop through every single turn
48     for (i in 0..totaltime - 1) {
49         // arrived
50         if (i < btList.size) {
51             arrivedList.add(btList[i].copy())
52         }
53
54         var priorIndex = 0
55         // find highest priority job in arrivedList
56         // skipped if only one process is left
57         for (j in 0..arrivedList.size - 1) {
58             if ((arrivedList.size != 1) && (arrivedList[priorIndex].priority > arrivedList[j].priority)) {
59                 priorIndex = j
60             }
61         }
62         // print highest priority job
63         output.add("P${arrivedList[priorIndex].processNum}")
64
65         // remove the process if reaming bursttime = 0
66         arrivedList[priorIndex].burstTime -= 1
67         if (arrivedList[priorIndex].burstTime == 0) {
68             finishTime[arrivedList[priorIndex].processNum - 1] = i + 1.toDouble()
69             arrivedList.removeAt(priorIndex)
70         }
71     }
72     print(priorityGantz(output))
73     print("\n")
74 }
75
76 fun priorityGantz(list: MutableList<String>): String {
77     var output = ""
78     var i = 0
79     while (i < list.size) {
80         var count = 0
81         var j = i
82         while (j < list.size - 1 && list[j] == list[j + 1]) {
83             count++
84             j++
85         }
86         output += "|"
87         for (z in 0..count) {
88             output += if (z == count / 2) "${list[j]}" else "-"
89         }
90         output += "|"
91         i = j + 1
92     }
93     return output
94 }
95
96 fun summarize() {
97     var waitTimeList: MutableList<Double> = mutableListOf()
98     print("In summary:\n")
99     for (i in 0..(btList.size - 1)) {
100         val waitTime = finishTime[i] - btList[i].burstTime - btList[i].arrivalTime
101         waitTimeList.add(waitTime.toDouble())
102         print("P${i + 1} waited %5.2f ms\n".format(waitTime))
103     }
104 }

```

3.Priority Queue


```

41 fun process() {
42     print("Scheduling scheme: Priority Queue\n")
43
44     // store a string that will print after everything is finished
45     var output: MutableList<String> = mutableListOf()
46
47     //loop through every single turn
48     for (i in 0..totaltime - 1) {
49         // arrived
50         if (i < btList.size) {
51             arrivedList.add(btList[i].copy())
52         }
53
54         var priorIndex = 0
55         // find highest priority job in arrivedList
56         // skipped if only one process is left
57         for (j in 0..arrivedList.size - 1) {
58             if ((arrivedList.size != 1) && (arrivedList[priorIndex].priority > arrivedList[j].priority)) {
59                 priorIndex = j
60             }
61         }
62         // print highest priority job
63         output.add("P${arrivedList[priorIndex].processNum}")
64
65         // remove the process if reaming bursttime = 0
66         arrivedList[priorIndex].burstTime -- 1
67         if (arrivedList[priorIndex].burstTime == 0) {
68             finishTime[arrivedList[priorIndex].processNum - 1] = i + 1.toDouble()
69             arrivedList.removeAt(priorIndex)
70         }
71     }
72     print(priorityGantz(output))
73     print("\n")
74 }
75
76 fun priorityGantz(list: MutableList<String>): String {
77     var output = ""
78     var i = 0
79     while (i < list.size) {
80         var count = 0
81         var j = i
82         while (j < list.size - 1 && list[j] == list[j + 1]) {
83             count++
84             j++
85         }
86         output += "|"
87         for (z in 0..count) {
88             output += if (z == count / 2) "${list[j]}" else "-"
89         }
90         output += "|"
91         i = j + 1
92     }
93     return output
94 }
95
96 fun summarize() {
97     var waitTimeList: MutableList<Double> = mutableListOf()
98     print("In summary:\n")
99     for (i in 0..btList.size - 1) {
100         val waitTime = finishTime[i] - btList[i].burstTime - btList[i].arrivalTime
101         waitTimeList.add(waitTime.toDouble())
102         print("P${i + 1} waited %5.2f ms\n".format(waitTime))
103     }
104 }

```

Add priority variable;
Determines which jobs to be executed fist

```

43 fun process() {
44     print("Scheduling scheme: Priority Queue\n")
45
46     // store a string that will print after everything is finished
47     var output: MutableList<String> = mutableListOf()
48
49     //loop through every single turn
50     for (i in 0..totaltime - 1) {
51         // arrived
52         if (i < btList.size) {
53             arrivedList.add(btList[i].copy())
54         }
55
56         var priorIndex = 0
57         // find highest priority job in arrivedList
58         // skipped if only one process is left
59         for (j in 0..arrivedList.size - 1) {
60             if ((arrivedList.size != 1) && (arrivedList[priorIndex].priority > arrivedList[j].pri
61                 priorIndex = j
62             }
63         }
64         // print highest priority job
65         output.add("P${arrivedList[priorIndex].processNum}")
66
67
68         // remove the process if reaming burstime = 0
69         arrivedList[priorIndex].burstTime -= 1
70         if (arrivedList[priorIndex].burstTime == 0) {
71             finishTime[arrivedList[priorIndex].processNum - 1] = i + 1.toDouble()
72             arrivedList.removeAt(priorIndex)
73         }
74     }
75     print(priorityGantz(output))
76     print("\n")
77 }
78
79 fun priorityGantz(list: MutableList<String>): String {
80     var output = ""
81     var i = 0
82     while (i < list.size) {
83         var count = 0
84         var j = i
85         while (j < list.size - 1 && list[j] == list[j + 1]) {
86             count++
87             j++
88         }
89         output += "|"
90         for (z in 0..count) {
91             output += if (z == count / 2) "${list[i]}" else "-"

```

Loops through the process pool
to pick the shortest one

```

43 fun process() {
44     print("Scheduling scheme: Priority Queue\n")
45
46     // store a string that will print after everything is finished
47     var output: MutableList<String> = mutableListOf()
48
49     //loop through every single turn
50     for (i in 0..totaltime - 1) {
51         // arrived
52         if (i < btList.size) {
53             arrivedList.add(btList[i].copy())
54         }
55
56         var priorIndex = 0
57         // find highest priority job in arrivedList
58         // skipped if only one process is left
59         for (j in 0..arrivedList.size - 1) {
60             if ((arrivedList.size != 1) && (arrivedList[priorIndex].priority > arrivedList[j].priority)) {
61                 priorIndex = j
62             }
63         }
64         // print highest priority job
65         output.add("P${arrivedList[priorIndex].processNum}")
66
67
68         // remove the process if reaming burstime = 0
69         arrivedList[priorIndex].burstTime -= 1
70         if (arrivedList[priorIndex].burstTime == 0) {
71             finishTime[arrivedList[priorIndex].processNum - 1] = i + 1.toDouble()
72             arrivedList.removeAt(priorIndex)
73         }
74     }
75     print(priorityGantz(output))
76     print("\n")
77 }
78
79 fun priorityGantz(list: MutableList<String>): String {
80     var output = ""
81     var i = 0
82     while (i < list.size) {
83         var count = 0
84         var j = i
85         while (j < list.size - 1 && list[j] == list[j + 1]) {
86             count++
87             j++
88         }
89         output += "|"
90         for (z in 0..count) {
91             output += if (z == count / 2) "${list[i]}" else "-"

```

If a higher priority task arrives,
pause the current one to prioritize it first

Since P2 has higher priority,
immediately switch to P2.

Number of process:4

Enter burst time for each process

At	Process	Burst time(ms)
0	P1	4
1	P2	5
2	P3	4
3	P4	6

Enter priority for each process

At	Process	Priority
0	P1	4
1	P2	2
2	P3	3
3	P4	1

Scheduling scheme: Priority Queue

|P1||P2-||--P4---||-P2-||-P3--||-P1-|

In summary:

P1 waited 15.00 ms

P2 waited 6.00 ms

P3 waited 10.00 ms

P4 waited 0.00 ms

Since there are 4 processes,

The average waiting time is 7.75 ms

The same goes for when P4 which has highest priority arrives

Number of process:4

Enter burst time for each process

At	Process	Burst time(ms)
0	P1	4
1	P2	5
2	P3	4
3	P4	6

Enter priority for each process

At	Process	Priority
0	P1	4
1	P2	2
2	P3	3
3	P4	1

Scheduling scheme: Priority Queue

|P1||P2-||--P4---||-P2-||-P3--||-P1-|

In summary:

P1 waited 15.00 ms

P2 waited 6.00 ms

P3 waited 10.00 ms

P4 waited 0.00 ms

Since there are 4 processes,

The average waiting time is 7.75 ms

The waiting time can still be calculate the same way as the first algorithm

Number of process:4

Enter burst time for each process

At	Process	Burst time(ms)
0	P1	4
1	P2	5
2	P3	4
3	P4	6

Enter priority for each process

At	Process	Priority
0	P1	4
1	P2	2
2	P3	3
3	P4	1

Scheduling scheme: Priority Queue

|P1||P2-||--P4---||-P2-||-P3--||-P1-|

In summary:

P1 waited 15.00 ms

P2 waited 6.00 ms

P3 waited 10.00 ms

P4 waited 0.00 ms

Since there are 4 processes,

The average waiting time is 7.75 ms

The waiting time can still be calculate the same way as the first algorithm

Finished time

—

(Arrival time
+ Burst time)

Number of process:4

Enter burst time for each process

At	Process	Burst time(ms)
0	P1	4
1	P2	5
2	P3	4
3	P4	6

Enter priority for each process

At	Process	Priority
0	P1	4
1	P2	2
2	P3	3
3	P4	1

Scheduling scheme: Priority Queue

|P1||P2-||--P4---||-P2-||-P3--||-P1-|

In summary:

P1 waited 15.00 ms

P2 waited 6.00 ms

P3 waited 10.00 ms

P4 waited 0.00 ms

Since there are 4 processes,

The average waiting time is 7.75 ms

```

43 1 1 41 fun process() {
44 2 2 42     print("Scheduling scheme: Round Robin\n")
45 3 3 43     var arrivedList: MutableList<Process> = mutableListOf()
46 4 4 44
47 5 5 45     // store a string that will print after everything is finished
48 6 6 46     var output: MutableList<String> = mutableListOf()
49 7 7 47     //output.add("P${arrivedList[minIndex].processNum}")
50 8 8 48
51 9 9 49     var turn = 0
52 10 10 50     var quantumCount = 0
53 11 11 51
54 12 12 52     for (i in 0..totaltime - 1) {
55 13 13 53
56 14 14 54         if (i < btList.size) {
57 15 15 55             arrivedList.add(btList[i].copy())
58 16 16 56         }
59 17 17 57
60 18 18 58         while (arrivedList.size > 0 && arrivedList[turn].burstTime == 0) {
61 19 19 59             turn = (turn + 1) % arrivedList.size
62 20 20 60         }
63 21 21 61
64 22 22 62         output.add("P${arrivedList[turn].processNum}")
65 23 23 63         arrivedList[turn].burstTime--
66 24 24 64
67 25 25 65         quantumCount = (quantumCount + 1) % quantum
68 26 26 66         if (arrivedList[turn].burstTime == 0) {
69 27 27 67             //quantumCount = (quantumCount + 1) % quantum
70 28 28 68             finishTime[turn] = i.toDouble()
71 29 29 69             // if (arrivedList.size > 0) turn = (turn + 1) % arrivedList.size
72 30 30 70         }
73 31 31 71
74 32 32 72         if (arrivedList.size > 0 && i != 0 && quantumCount == 0) {
75 33 33 73             turn = (turn + 1) % arrivedList.size
76 34 34 74         }
77 35 35 75     }
78 36 36 76
79 37 37 77     print(RRGantz(output))
80 38 38 78     print("\n")
81 39 39 79 }
82 40 40 80
83 41 41 81 fun RRGantz(list: MutableList<String>): String {
84 42 42 82     var output = ""
85 43 43 83     var cq = 0
86 44 44 84
87 45 45 85     var i = 0
88 46 46 86
89 47 47 87     while (i < list.size - 1) {
90 48 48 88         var j = i
91 49 49 89         if (cq == 0) {
92 50 50 90             output += "|"
93 51 51 91         }
94 52 52 92
95 53 53 93         var count = 0
96 54 54 94         while (j < list.size - 1 && cq < quantum - 1 && list[j] == list[j + 1]) {
97 55 55 95             j++
98 56 56 96             cq = (cq + 1) % quantum
99 57 57 97             count++
100 58 58 98         }
101 59 59 99
102 60 60 100         for (k in 0..count) {
103 61 61 101             output += "${list[j]}"
104 62 62 102         }
105 63 63 103         i = i + 1

```

4.RoundRobin


```
FirstC FirstC Fi FirstComeFirstServe.kt ShortestJobFirst.kt PriorityQueue.kt RoundRobin.kt x
43 1 1 41 fun process() {
44 2 2 42 print("Scheduling scheme: Round Robin\n")
45 3 3 43 var arrivedList: MutableList<Process> = mutableListOf()
46 4 4 44
47 5 5 45 // store a string that will print after everything is finished
48 6 6 46 var output: MutableList<String> = mutableListOf()
49 7 7 47 //output.add("P${arrivedList[minIndex].processNum}")
50 8 8 48
51 9 9 49 var turn = 0
52 10 10 50 var quantumCount = 0
53 11 11 51
54 12 12 52 for (i in 0..totaltime - 1) {
55 13 13 53
56 14 14 54 if (i < btlist.size) {
57 15 15 55 arrivedList.add(btlist[i].copy())
58 16 16 56 }
59 17 17 57
60 18 18 58 while (arrivedList.size > 0 && arrivedList[turn].burstTime == 0) {
61 19 19 59 turn = (turn + 1) % arrivedList.size
62 20 20 60 }
63 21 21 61
64 22 22 62 output.add("P${arrivedList[turn].processNum}")
65 23 23 63 arrivedList[turn].burstTime--
66 24 24 64
67 25 25 65 quantumCount = (quantumCount + 1) % quantum
68 26 26 66 if (arrivedList[turn].burstTime == 0) {
69 27 27 67 //quantumCount = (quantumCount + 1) % quantum
70 28 28 68 finishTime[turn] = i.toDouble()
71 29 29 69 // if (arrivedList.size > 0) turn = (turn + 1) % arrivedList.size
72 30 30 70 }
73 31 31 71
74 32 32 72 if (arrivedList.size > 0 && i != 0 && quantumCount == 0) {
75 33 33 73 turn = (turn + 1) % arrivedList.size
76 34 34 74 }
77 35 35 75 }
78 36 36 76
79 37 37 77 print(RRGantz(output))
80 38 38 78 print("\n")
81 39 39 79 }
82 40 40 80
83 41 41 81 fun RRGantz(list: MutableList<String>): String {
84 42 42 82 var output = ""
85 43 43 83 var cq = 0
86 44 44 84
87 45 45 85 var i = 0
88 46 46 86
89 47 47 87 while (i < list.size - 1) {
90 48 48 88 var j = i
91 49 49 89 if (cq == 0) {
92 50 50 90 output += "|"
93 51 51 91 }
94 52 52 92
95 53 53 93 var count = 0
96 54 54 94 while (j < list.size - 1 && cq < quantum - 1 && list[j] == list[j + 1]) {
97 55 55 95 j++
98 56 56 96 cq = (cq + 1) % quantum
99 57 57 97 count++
100 58 58 98 }
101 59 59 99
102 60 60 100 for (k in 0..count) {
103 61 61 101 output += "${list[j]}"
104 62 62 102 }
105 63 63 103 i = i + 1
```

Introduce quantum;
Fairly divides tasks amongst
processes in the pool

```

40
41 fun process() {
42     print("Scheduling scheme: Round Robin\n")
43     var arrivedList: MutableList<Process> = mutableListOf()
44
45     // store a string that will print after everything is finished
46     var output: MutableList<String> = mutableListOf()
47     //output.add("P${arrivedList[minIndex].processNum}")
48
49     var turn = 0
50     var quantumCount = 0
51
52     for (i in 0..totaltime - 1) {
53
54         if (i < btList.size) {
55             arrivedList.add(btList[i].copy())
56         }
57
58         while (arrivedList.size > 0 && arrivedList[turn].burstTime == 0) {
59             turn = (turn + 1) % arrivedList.size
60         }
61
62         output.add("P${arrivedList[turn].processNum}")
63         arrivedList[turn].burstTime--
64
65         quantumCount = (quantumCount + 1) % quantum
66         if (arrivedList[turn].burstTime == 0) {
67             //quantumCount = (quantumCount + 1) % quantum
68             finishTime[turn] = i.toDouble()
69             // if (arrivedList.size > 0) turn = (turn + 1) % arrivedList.size
70         }
71
72         if (arrivedList.size > 0 && i != 0 && quantumCount == 0) {
73             turn = (turn + 1) % arrivedList.size
74         }
75     }
76
77     print(RRGantz(output))
78     print("\n")
79 }
80
81 fun RRGantz(list: MutableList<String>): String {
82     var output = ""
83     var cq = 0
84
85     var i = 0
86
87     while (i < list.size - 1) {
88         var j = i
89         if (j == 0) {

```

Each tasks will be execute
concurrently and equally

```

40
41 fun process() {
42     print("Scheduling scheme: Round Robin\n")
43     var arrivedList: MutableList<Process> = mutableListOf()
44
45     // store a string that will print after everything is finished
46     var output: MutableList<String> = mutableListOf()
47     //output.add("P${arrivedList[minIndex].processNum}")
48
49     var turn = 0
50     var quantumCount = 0
51
52     for (i in 0..totaltime - 1) {
53
54         if (i < btList.size) {
55             arrivedList.add(btList[i].copy())
56         }
57
58         while (arrivedList.size > 0 && arrivedList[turn].burstTime == 0) {
59             turn = (turn + 1) % arrivedList.size
60         }
61
62         output.add("P${arrivedList[turn].processNum}")
63         arrivedList[turn].burstTime--
64
65         quantumCount = (quantumCount + 1) % quantum
66         if (arrivedList[turn].burstTime == 0) {
67             //quantumCount = (quantumCount + 1) % quantum
68             finishTime[turn] = i.toDouble()
69             // if (arrivedList.size > 0) turn = (turn + 1) % arrivedList.size
70         }
71
72         if (arrivedList.size > 0 && i != 0 && quantumCount == 0) {
73             turn = (turn + 1) % arrivedList.size
74         }
75     }
76
77     print(RRGantz(output))
78     print("\n")
79 }
80
81 fun RRGantz(list: MutableList<String>): String {
82     var output = ""
83     var cq = 0
84
85     var i = 0
86
87     while (i < list.size - 1) {
88         var j = i
89         if (j == 0) {

```

Ensures all tasks progress

Executes each task for a turn of 3
(quantum)

Number of process:5

Quantum is:3

Enter burst time for each process

At	Process	Burst time(ms)
0	P1	4
1	P2	5
2	P3	4
3	P4	3
4	P5	5

Scheduling scheme: Round Robin

|P1P1P1|P2P2P2|P3P3P3|P4P4P4|P5P5P5|P1P2P2|P3P5P5

In summary:

P1 waited 12.00 ms

P2 waited 13.00 ms

P3 waited 15.00 ms

P4 waited 9.00 ms

P5 waited 16.00 ms

Since there are 5 processes,

The average waiting time is 13.0 ms

Notice when one task is done
and the quantum is yet complete, it utilize
the remaining turn on the next task

In P1 to P2 to P2
(6th quantum)

Number of process:5

Quantum is:3

Enter burst time for each process

At	Process	Burst time(ms)
0	P1	4
1	P2	5
2	P3	4
3	P4	3
4	P5	5

Scheduling scheme: Round Robin

|P1P1P1|P2P2P2|P3P3P3|P4P4P4|P5P5P5|P1P2P2|P3P5P5

In summary:

P1 waited 12.00 ms

P2 waited 13.00 ms

P3 waited 15.00 ms

P4 waited 9.00 ms

P5 waited 16.00 ms

Since there are 5 processes,

The average waiting time is 13.0 ms

Since in round robin, we neglect arrival time.

Number of process:5

Quantum is:3

Enter burst time for each process

At	Process	Burst time(ms)
0	P1	4
1	P2	5
2	P3	4
3	P4	3
4	P5	5

Scheduling scheme: Round Robin

|P1P1P1|P2P2P2|P3P3P3|P4P4P4|P5P5P5|P1P2P2|P3P5P5

In summary:

P1 waited 12.00 ms

P2 waited 13.00 ms

P3 waited 15.00 ms

P4 waited 9.00 ms

P5 waited 16.00 ms

Since there are 5 processes,

The average waiting time is 13.0 ms

Hence a difference in waiting time calculation

Number of process:5

Quantum is:3

Enter burst time for each process

At	Process	Burst time(ms)
0	P1	4
1	P2	5
2	P3	4
3	P4	3
4	P5	5

Scheduling scheme: Round Robin

|P1P1P1|P2P2P2|P3P3P3|P4P4P4|P5P5P5|P1P2P2|P3P5P5

In summary:

P1 waited 12.00 ms

P2 waited 13.00 ms

P3 waited 15.00 ms

P4 waited 9.00 ms

P5 waited 16.00 ms

Since there are 5 processes,

The average waiting time is 13.0 ms

Hence a difference in waiting time
calculation

Finished time

—
Burst time

Number of process:5

Quantum is:3

Enter burst time for each process

At	Process	Burst time(ms)
----	---------	----------------

0	P1	4
---	----	---

1	P2	5
---	----	---

2	P3	4
---	----	---

3	P4	3
---	----	---

4	P5	5
---	----	---

Scheduling scheme: Round Robin

|P1P1P1|P2P2P2|P3P3P3|P4P4P4|P5P5P5|P1P2P2|P3P5P5

In summary:

P1 waited 12.00 ms

P2 waited 13.00 ms

P3 waited 15.00 ms

P4 waited 9.00 ms

P5 waited 16.00 ms

Since there are 5 processes,

The average waiting time is 13.0 ms



Happy Coding!

THANK YOU