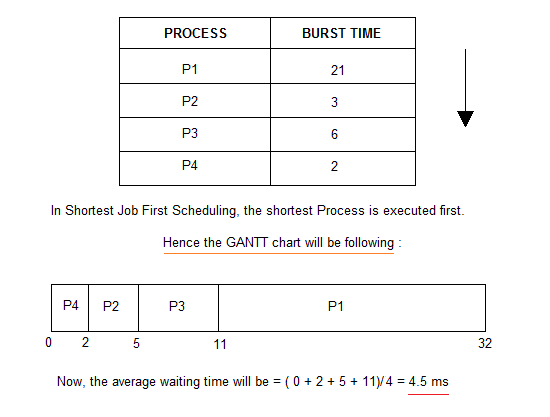
Shortest Job First (SJF) Scheduling

Shortest Job First scheduling works on the process with the shortest **burst time** or **duration** first.

* This is the best approach to minimize waiting time.
* This is used in [Batch Systems](https://www.studytonight.com/operating-system/types-of-os" \t "_blank).
* It is of two types:
  1. Non Pre-emptive
  2. Pre-emptive
* To successfully implement it, the burst time/duration time of the processes should be known to the processor in advance, which is practically not feasible all the time.
* This scheduling algorithm is optimal if all the jobs/processes are available at the same time. (either Arrival time is 0 for all, or Arrival time is same for all)

Non Pre-emptive Shortest Job First

Consider the below processes available in the ready queue for execution, with **arrival time** as 0 for all and given **burst times**.



As you can see in the **GANTT chart** above, the process **P4** will be picked up first as it has the shortest burst time, then **P2**, followed by **P3** and at last **P1**.

We scheduled the same set of processes using the [First come first serve](https://www.studytonight.com/operating-system/first-come-first-serve" \t "_blank) algorithm in the previous tutorial, and got average waiting time to be 18.75 ms, whereas with SJF, the average waiting time comes out 4.5 ms.

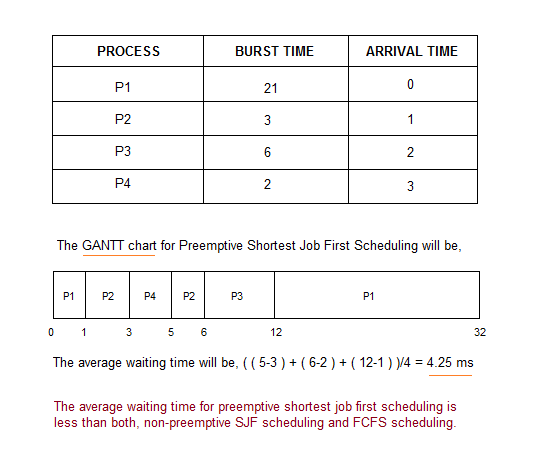
Problem with Non Pre-emptive SJF

If the **arrival time** for processes are different, which means all the processes are not available in the ready queue at time 0, and some jobs arrive after some time, in such situation, sometimes process with short burst time have to wait for the current process's execution to finish, because in Non Pre-emptive SJF, on arrival of a process with short duration, the existing job/process's execution is not halted/stopped to execute the short job first.

This leads to the problem of **Starvation**, where a shorter process has to wait for a long time until the current longer process gets executed. This happens if shorter jobs keep coming, but this can be solved using the concept of **aging**.

Pre-emptive Shortest Job First

In Preemptive Shortest Job First Scheduling, jobs are put into ready queue as they arrive, but as a process with **short burst time** arrives, the existing process is preempted or removed from execution, and the shorter job is executed first.



As you can see in the **GANTT chart** above, as **P1** arrives first, hence it's execution starts immediately, but just after 1 ms, process **P2** arrives with a **burst time** of 3 ms which is less than the burst time of **P1**, hence the process **P1**(1 ms done, 20 ms left) is preemptied and process **P2** is executed.

As **P2** is getting executed, after 1 ms, **P3** arrives, but it has a burst time greater than that of **P2**, hence execution of **P2** continues. But after another millisecond, **P4** arrives with a burst time of 2 ms, as a result **P2**(2 ms done, 1 ms left) is preemptied and **P4** is executed.

After the completion of **P4**, process **P2** is picked up and finishes, then **P2** will get executed and at last **P1**.

The Pre-emptive SJF is also known as **Shortest Remaining Time First**, because at any given point of time, the job with the shortest remaining time is executed first.

Starting with the **Advantages:** of [Shortest Job First](https://www.studytonight.com/operating-system/shortest-job-first) scheduling algorithm.

* According to the definition, short processes are executed first and then followed by longer processes.
* The throughput is increased because more processes can be executed in less amount of time.

And the **Disadvantages:**

* The time taken by a process must be known by the CPU beforehand, which is not possible.
* Longer processes will have more waiting time, eventually they'll suffer starvation.

**Note:** Preemptive Shortest Job First scheduling will have the same advantages and disadvantages as those for SJF.

Priority CPU Scheduling

In this tutorial we will understand the priority scheduling algorithm, how it works and its advantages and disadvantages.

In the [Shortest Job First](https://www.studytonight.com/operating-system/shortest-job-first" \t "_blank) scheduling algorithm, the priority of a process is generally the inverse of the CPU burst time, i.e. the larger the burst time the lower is the priority of that process.

In case of priority scheduling the priority is not always set as the inverse of the CPU burst time, rather it can be internally or externally set, but yes the scheduling is done on the basis of priority of the process where the process which is most urgent is processed first, followed by the ones with lesser priority in order.

Processes with same priority are executed in FCFS manner.

The priority of process, when internally defined, can be decided based on **memory requirements**, **time limits** ,**number of open files**, **ratio of I/O burst to CPU burst** etc.

Whereas, external priorities are set based on criteria outside the operating system, like the importance of the process, funds paid for the computer resource use, makrte factor etc.

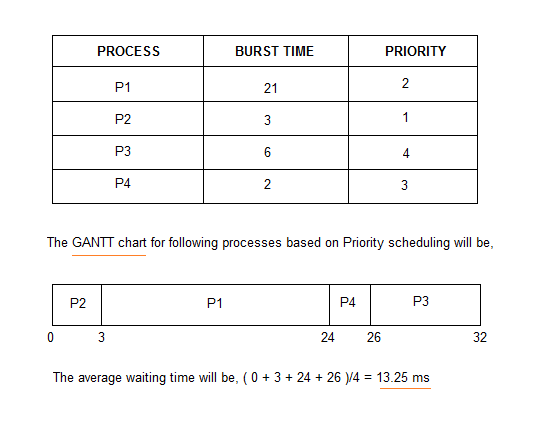
Types of Priority Scheduling Algorithm

Priority scheduling can be of two types:

1. **Preemptive Priority Scheduling**: If the new process arrived at the ready queue has a higher priority than the currently running process, the CPU is preempted, which means the processing of the current process is stoped and the incoming new process with higher priority gets the CPU for its execution.
2. **Non-Preemptive Priority Scheduling**: In case of non-preemptive priority scheduling algorithm if a new process arrives with a higher priority than the current running process, the incoming process is put at the head of the ready queue, which means after the execution of the current process it will be processed.

Example of Priority Scheduling Algorithm

Consider the below table fo processes with their respective CPU burst times and the priorities.



As you can see in the GANTT chart that the processes are given CPU time just on the basis of the priorities.

Problem with Priority Scheduling Algorithm

In priority scheduling algorithm, the chances of **indefinite blocking** or **starvation**.

A process is considered blocked when it is ready to run but has to wait for the CPU as some other process is running currently.

But in case of priority scheduling if new higher priority processes keeps coming in the ready queue then the processes waiting in the ready queue with lower priority may have to wait for long durations before getting the CPU for execution.

In 1973, when the IBM 7904 machine was shut down at MIT, a low-priority process was found which was submitted in 1967 and had not yet been run.

Using Aging Technique with Priority Scheduling

To prevent starvation of any process, we can use the concept of **aging** where we keep on increasing the priority of low-priority process based on the its waiting time.

For example, if we decide the aging factor to be **0.5** for each day of waiting, then if a process with priority **20**(which is comparitively low priority) comes in the ready queue. After one day of waiting, its priority is increased to **19.5** and so on.

Doing so, we can ensure that no process will have to wait for indefinite time for getting CPU time for processing.

**Advantages** of [Priority Scheduling](https://www.studytonight.com/operating-system/priority-scheduling):

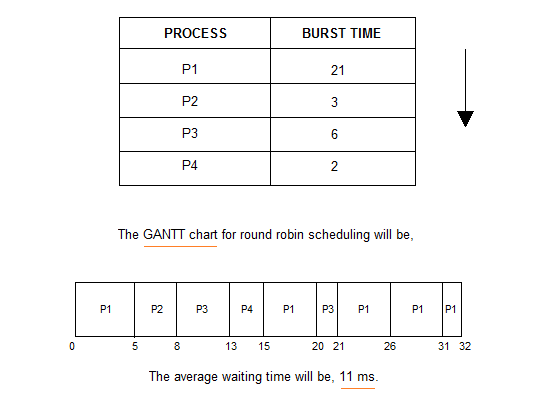
* The priority of a process can be selected based on memory requirement, time requirement or user preference. For example, a high end game will have better graphics, that means the process which updates the screen in a game will have higher priority so as to achieve better graphics performance.

Some **Disadvantages:**

* A second scheduling algorithm is required to schedule the processes which have same priority.
* In preemptive priority scheduling, a higher priority process can execute ahead of an already executing lower priority process. If lower priority process keeps waiting for higher priority processes, starvation occurs.

Round Robin Scheduling

* A fixed time is allotted to each process, called **quantum**, for execution.
* Once a process is executed for given time period that process is preemptied and other process executes for given time period.
* Context switching is used to save states of preemptied processes.



Round Robin (RR)

Here are some **Advantages:** of using the [Round Robin Scheduling](https://www.studytonight.com/operating-system/round-robin-scheduling):

* Each process is served by the CPU for a fixed time quantum, so all processes are given the same priority.
* Starvation doesn't occur because for each round robin cycle, every process is given a fixed time to execute. No process is left behind.

and here comes the **Disadvantages:**

* The throughput in RR largely depends on the choice of the length of the time quantum. If time quantum is longer than needed, it tends to exhibit the same behavior as FCFS.
* If time quantum is shorter than needed, the number of times that CPU switches from one process to another process, increases. This leads to decrease in CPU efficiency.