Queuing analysis is one of the most important tools for those involved with computer and network analysis. It can be used to provide approximate answers to a host of questions, such as:

- What happens to file retrieval time when disk I/O utilization goes up?
- Does response time change if both processor speed and the number of users on the system are doubled?
- How many lines should a time-sharing system have on a dial-in rotary?
- How many terminals are needed in an on line inquiry center, and how much idle time will the operators have?

The number of questions that can be addressed with a queuing analysis is endless and touches on virtually every area in computer science. The ability to make such an analysis is an essential tool for those involved in this field.

Although the theory of queuing is mathematically complex, the application of queuing theory to the analysis of performance is, in many cases, remarkably straightforward. A knowledge of elementary statistical concepts (means and standard deviations) and a basic understanding of the applicability of queuing theory is all that is required. Armed with these, the analyst can often make a queuing analysis on the back of an envelope using readily available queuing tables, or with the use of simple computer programs that occupy only a few lines of code.

The purpose of this paper is to provide a practical guide to queuing analysis. A subset, although a very important subset, of the subject is addressed. In the final section, pointers to additional references are provided. An annex to this paper reviews some elementary concepts in probability and statistics.

WHY QUEUING ANALYSIS?

There are many cases when it is important to be able to project the effect of some change in a design: either the load on a system is expected to increase or a design change is contemplated. For example, an organization supports a number of terminals, personal computers, and workstations on a 100-Mbps local area network (LAN). An additional department in the building is to be cut over onto the network. Can the existing LAN handle the increased workload, or would it be better to provide a second LAN with a bridge between the two? There are other cases in which no facility exists but, on the basis of expected demand, a system design needs to be created. For example, a department intends to equip all of its personnel with a personal computer and to configure these into a LAN with a file server. Based on experience elsewhere in the company, the load generated by each PC can be estimated.

The concern is system performance. In an interactive or real-time application, often the parameter of concern is response time. In other cases, throughput is the principal issue. In any case, projections of performance are to be made on the basis of existing load information or on the basis of estimated load for a new environment. A number of approaches are possible:

1. Do an after-the-fact analysis based on actual values.
2. Make a simple projection by scaling up from existing experience to the expected future environment.
3. Develop an analytic model based on queuing theory.
4. Program and run a simulation model.

Option 1 is no option at all: we will wait and see what happens. This leads to unhappy users and to unwise purchases. Option 2 sounds more promising. The analyst may take the position that it is impossible to project future demand with any degree of certainty. Therefore, it is pointless to attempt some exact modeling procedure. Rather, a rough-and-ready projection will provide ballpark estimates. The problem with this approach is that the behavior of most systems

under a changing load is not what one would intuitively expect. If there is an environment in which there is a shared facility (e.g., a network, a transmission line, a time-sharing system), then the performance of that system typically responds in an exponential way to increases in demand.

Figure 1 is a typical example. The upper line shows what happens to user response time on a shared facility as the load on that facility increases. The load is expressed as a fraction of capacity. Thus, if we are dealing with a input from a disk that is capable of transferring 1000 blocks per second, then a load of 0.5 represents a transfer of 500 blocks per second, and the response time is the amount of time it takes to retransmit any incoming block. The lower line is a simple projection[1] based on a knowledge of the behavior of the system up to a load of 0.5. Note that while things appear rosy when the simple projection is made, performance on the system will in fact collapse beyond a load of about 0.8 to 0.9.

Thus, a more exact prediction tool is needed. Option 3 is to make use of an analytic model, which is one that can be expressed as a set of equations that can be solved to yield the desired parameters (response time, throughput, etc.). For computer, operating-system, and networking problems, and indeed for many practical real-world problems, analytic models based on queuing theory provide a reasonably good fit to reality. The disadvantage of queuing theory is that a number of simplifying assumptions must be made to derive equations for the parameters of interest.

The final approach is a simulation model. Here, given a sufficiently powerful and flexible simulation programming language, the analyst can model reality in great detail and avoid making many of the assumptions required of queuing theory. However, in most cases, a simulation model is not needed or at least is not advisable as a first step in the analysis. For one thing, both existing measurements and projections of future load carry with them a certain margin of error. Thus, no matter how good the simulation model, the value of the results are limited by the quality of the input. For another, despite the many assumptions required of queuing theory, the results that are produced often come quite close to those that would be produced by a more careful simulation analysis. Furthermore, a queuing analysis can literally be accomplished in a matter of minutes for a well-defined problem, whereas simulation exercises can take days, weeks, or longer to program and run.

Accordingly, it behooves the analyst to master the basics of queuing analysis.

QUEUING MODELS

The Single-Server Queue

The simplest queuing system is depicted in Figure 2. The central element of the system is a server, which provides some service to items. Items from some population of items arrive at the system to be served. If the server is idle, an item is served immediately. Otherwise, an arriving item joins a waiting line[2]. When the server has completed serving an item, the item departs. If there are items waiting in the queue, one is immediately dispatched to the server. The server in this model can represent anything that performs some function or service for a collection of items. Examples: a processor provides service to processes; a transmission line provides a transmission service to packets or frames of data; an I/O device provides a read or write service for I/O requests.

---

[1] In fact, the lower line is based on fitting a third-order polynomial to the data available up to a load of 0.5.

[2] The waiting line is referred to as a queue in some treatments in the literature; it is also common to refer to the entire system as a queue. Unless otherwise noted, we use the term *queue* to mean waiting line.

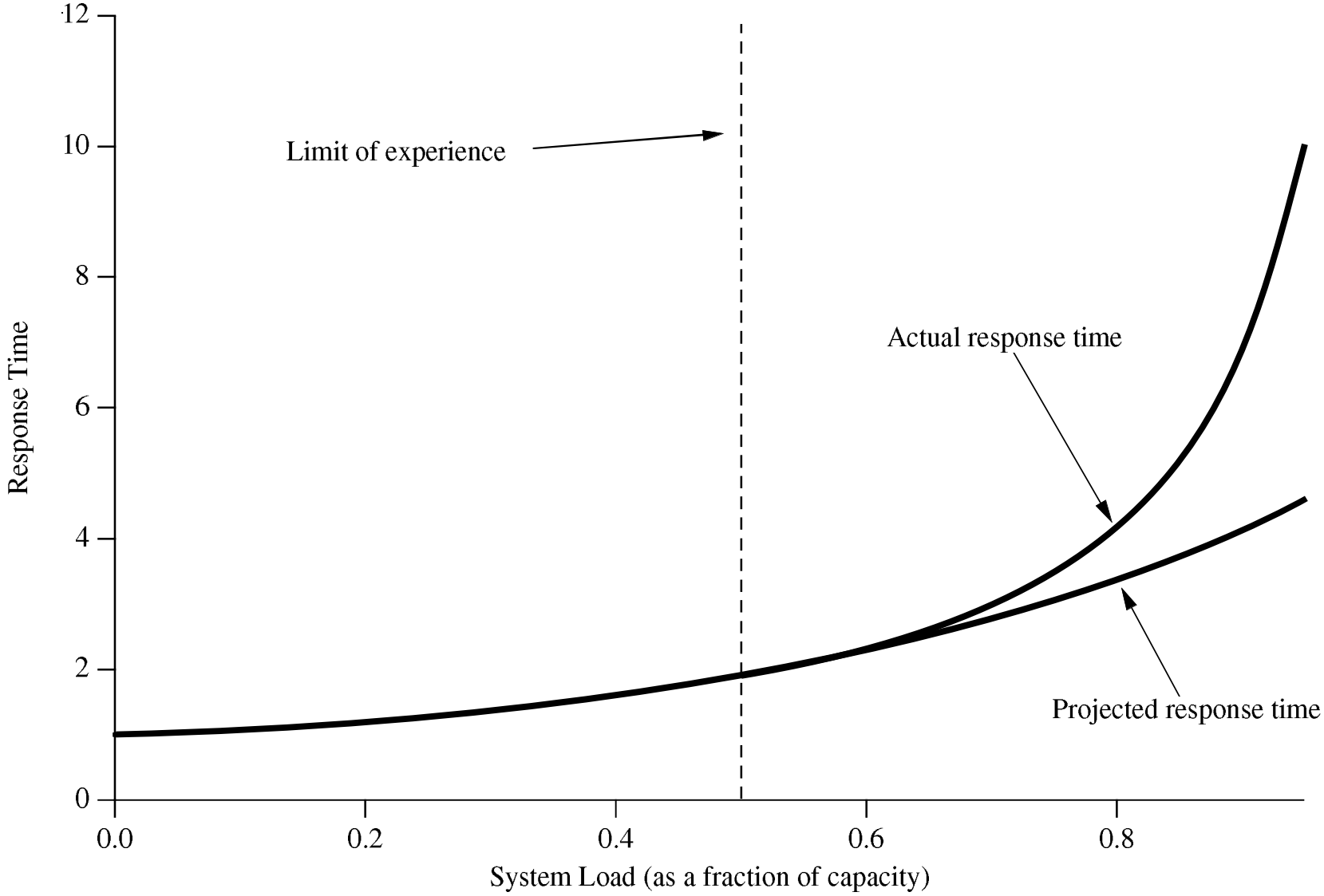


Figure 1 Projected Versus Actual Response Time

Queue Parameters

Figure 2 also illustrates some important parameters associated with a queuing model. Items arrive at the facility at some average rate (items arriving per second) λ. At any given time, a certain number of items will be waiting in the queue (zero or more); the average number waiting is *w*, and the mean time that an item must wait is T_w. T_w is averaged over all incoming items, including those that do not wait at all. The server handles incoming items with an average service time T_S; this is the time interval between the dispatching of an item to the server and the departure of that item from the server. Utilization, ρ, is the fraction of time that the server is busy, measured over some interval of time. Finally, two parameters apply to the system as a whole. The average number of items resident in the system, including the item being served (if any) and the items waiting (if any), is *r*; and the average time that an item spends in the system, waiting and being served, is T_r; we refer to this as the mean residence time.[3]

If we assume that the capacity of the queue is infinite, then no items are ever lost from the system; they are just delayed until they can be served. Under these circumstances, the departure rate equals the arrival rate. As the arrival rate, which is the rate of traffic passing through the system, increases, the utilization increases and with it, congestion. The queue becomes longer, increasing waiting time. At $\rho = 1$, the server becomes saturated, working 100% of the time. Thus, the theoretical maximum input rate that can be handled by the system is:

$$\lambda_{max} = \frac{1}{T_S}$$

However, queues become very large near system saturation, growing without bound when $\rho = 1$. Practical considerations, such as response time requirements or buffer sizes, usually limit the input rate for a single server to 70-90% of the theoretical maximum.

To proceed, we need to make some assumption about this model:

- **Item population:** Typically, we assume an infinite population. This means that the arrival rate is not altered by the loss of population. If the population is finite, then the population available for arrival is reduced by the number of items currently in the system; this would typically reduce the arrival rate proportionally.
- **Queue size:** Typically, we assume an infinite queue size. Thus, the waiting line can grow without bound. With a finite queue, it is possible for items to be lost from the system. In practice, any queue is finite. In many cases, this will make no substantive difference to the analysis. We address this issue briefly, below.
- **Dispatching discipline:** When the server becomes free, and if there is more than one item waiting, a decision must be made as to which item to dispatch next. The simplest approach is first-in, first-out; this discipline is what is normally implied when the term *queue* is used. Another possibility is last-in, first-out. One that you might encounter in practice is a dispatching discipline based on service time. For example, a packet-switching node may choose to dispatch packets on the basis of shortest first (to generate the most outgoing packets) or longest first (to minimize processing time relative to transmission time). Unfortunately, a discipline based on service time is very difficult to model analytically.

---

[3] Again, in some of the literature, this is referred to as the mean queuing time, while other treatments use mean queuing time to mean the average time spent waiting in the queue (before being served).

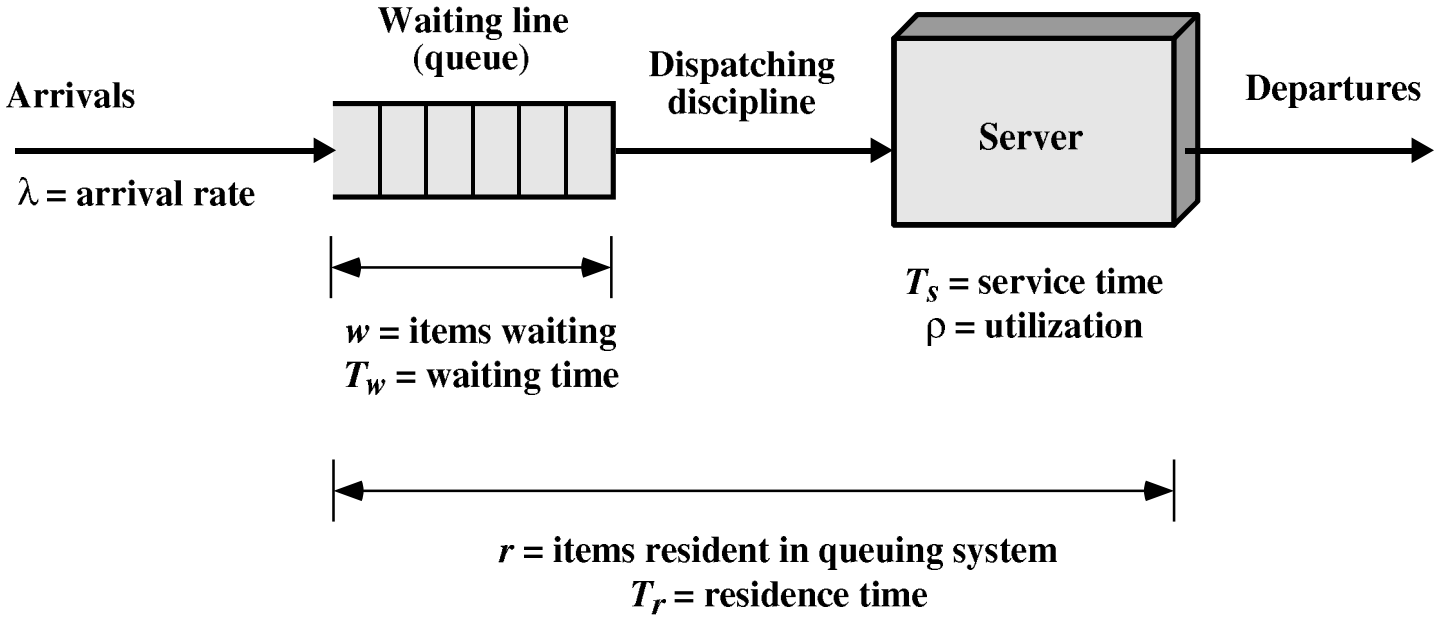


Figure 2 Queuing System Structure and Parameters for Single-Server Queue

Table 1 summarizes the notation that is used in Figure 2 and introduces some other parameters that are useful. In particular, we are often interested in the variability of various parameters, and this is neatly captured in the standard deviation.

The Multiserver Queue

Figure 3a shows a generalization of the simple model we have been discussing for multiple servers, all sharing a common queue. If an item arrives and at least one server is available, then the item is immediately dispatched to that server. It is assumed that all servers are identical; thus, if more than one server is available, it makes no difference which server is chosen for the item. If all servers are busy, a queue begins to form. As soon as one server becomes free, an item is dispatched from the queue using the dispatching discipline in force.

With the exception of utilization, all of the parameters illustrated in Figure 2 carry over to the multiserver case with the same interpretation. If we have N identical servers, then ρ is the utilization of each server, and we can consider $N\rho$ to be the utilization of the entire system; this latter term is often referred to as the traffic intensity, u. Thus, the theoretical maximum utilization is $N \times 100\%$, and the theoretical maximum input rate is:

$$\lambda_{max} = \frac{N}{T_s}$$

The key characteristics typically chosen for the multiserver queue correspond to those for the single-server queue. That is, we assume an infinite population and an infinite queue size, with a single infinite queue shared among all servers. Unless otherwise stated, the dispatching discipline is FIFO. For the multiserver case, if all servers are assumed identical, the selection of a particular server for a waiting item has no effect on service time.

By way of contrast, Figure 3b shows the structure of multiple single-server queues. As we shall see, this apparently minor change in structure has a significant impact on performance.

Basic Queuing Relationships

To proceed much further, we are going to have to make some simplifying assumptions. These assumptions risk making the models less valid for various real-world situations. Fortunately, in most cases, the results will be sufficiently accurate for planning and design purposes.

There are, however, some relationships that are true in the general case, and these are illustrated in Table 2. By themselves, these relationships are not particularly helpful.

Assumptions

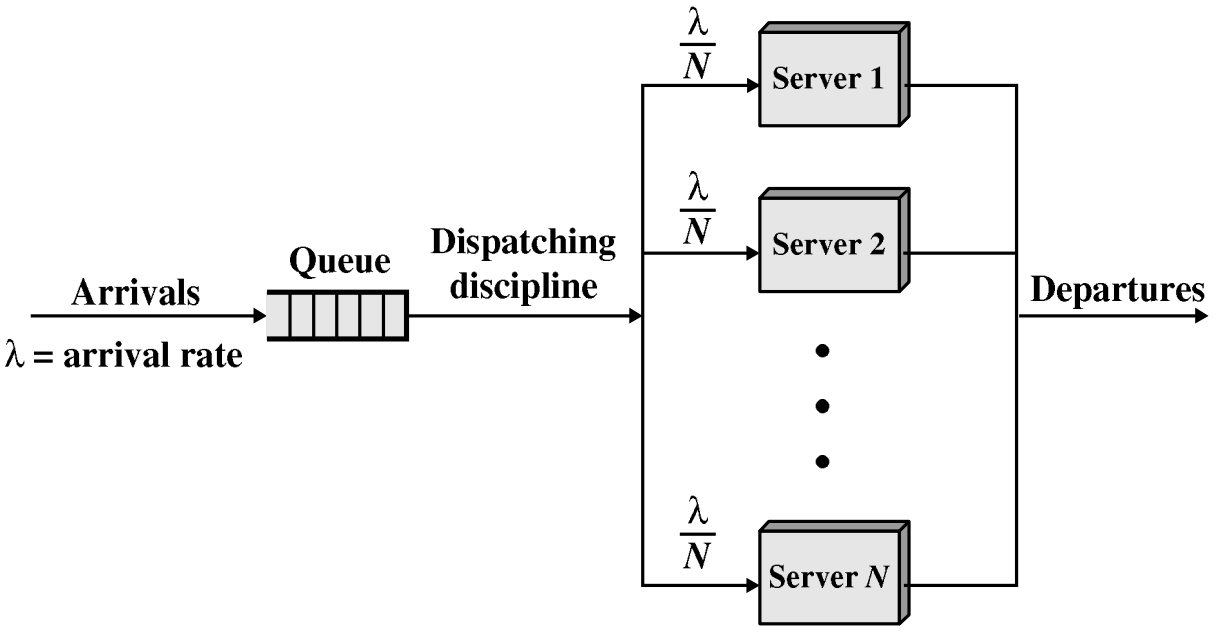
The fundamental task of a queuing analysis is as follows: Given the following information as input:

- Arrival rate
- Service time

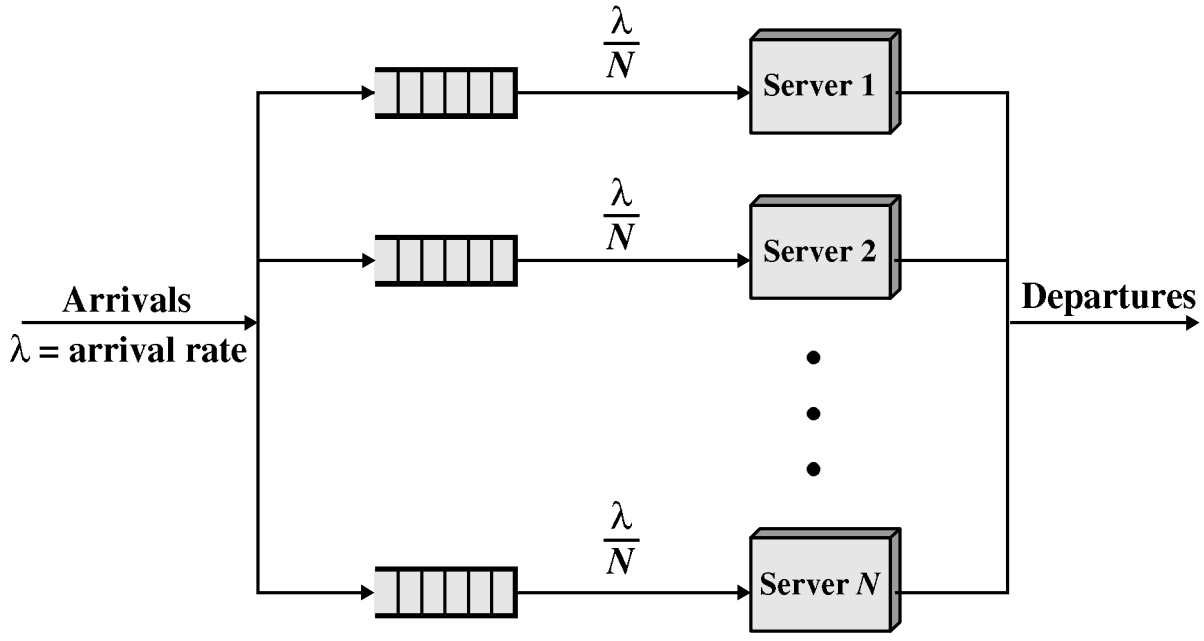
Provide as output information concerning:

- Items waiting
- Waiting time
- Items in residence
- Residence time.

What specifically would we like to know about these outputs? Certainly we would like to know their average values (w, T_w, r, T_r). In addition, it would be useful to know something about



(a) Multiserver queue



(b) Multiple Single-server queues

Figure 3 Multiserver Versus Multiple Single-Server Queues

Table 1 Notation for Queuing Systems

λ	=	arrival rate; mean number of arrivals per second
T_s	=	mean service time for each arrival; amount of time being served, not counting time waiting in the queue
σ_{Ts}	=	standard deviation of service time
ρ	=	utilization; fraction of time facility (server or servers) is busy
u	=	traffic intensity
r	=	mean number of items in system, waiting and being served (residence time)
R	=	number of items in system, waiting and being served
T_r	=	mean time an item spends in system (residence time)
T_R	=	time an item spends in system (residence time)
σ_r	=	standard deviation of r
σ_{Tr}	=	standard deviation of T_r
w	=	mean number of items waiting to be served
σ_w	=	standard deviation of w
T_w	=	mean waiting time (including items that have to wait and items with waiting time = 0)
T_d	=	mean waiting time for items that have to wait
N	=	number of servers
$m_X(y)$	=	the yth percentile; that value of y below which x occurs y percent of the time