Choosing K: Starting with k = 32 and doubling k until it reaches 32768, my function ChoosingK.R, outputs the autocorrelation with lag = 2, and confidence interval for each of the 5 statistics, for both the trace driven and exponentially driven traces, and for the set of k values listed above. From this, I chose the confidence intervals and k pairs that had k that were half the cutoff lag, which forced b to be twice the cutoff lag. Once I chose k, b is naturally set to the number of jobs in total mod k. Unfortunately, the optimal b value, which gives a more valid confidence interval, is very small, so this gives a wide confidence interval for some of the parameters. This makes sense as the data has very high variation.

Confidence Intervals Using Batch Means: Util = 0.1, alpha = 0.05

	Trace Driven	**Exp Driven**	**K (Trace, Exp)**
Mean Delay Time	(7911.73, 9914.77)	(8620.86, 30348.17)	(256,64)
Mean Wait Time	(7920.24, 9923.75)	(8629.51, 30357.01)	(256,64)
Mean Queue Length	(90.43, 113.33)	(98.54, 346.89)	(256,64)
Mean Number in Node	(90.53, 113.43)	(98.64, 346.99)	(256,64)
Server Utilization	(0.0958, 0.104)	(0.0963, 0.104)	(256,64)

Confidence Intervals Using Batch Means: Util = 0.2, alpha = 0.05

	Trace Driven	**Exp Driven**	**K (Trace, Exp)**
Mean Delay Time	(34482.09,87437.23)	(173611.99, 255333.76)	(64,256)
Mean Wait Time	(34496.84, 87457.48)	(173628.55, 255352.20)	(64,256)
Mean Queue Length	(394.14, 999.44)	(1984.44, 2918.55)	(64,256)
Mean Number in Node	(394.31, 999.67)	(1984.63, 2918.76)	(64,256)

Server Utilization	(0.1491, 0.2509)	(0.1961, 0.2039)	(64,256)

Confidence Intervals Using Batch Means: Util = 0.4, alpha = 0.05

	Trace Driven	**Exp Driven**	**K(T, E)**
Mean Delay Time	(398174.6, 802432.9)	(4843900.5, 8329503.5)	(128, 128)
Mean Wait Time	(398205.5, 802472.1)	(4843930.9, 8329543.1)	(128, 128)
Mean Queue Length	(4551.26, 9172.06)	(55367.3, 95208.9)	(128, 128)
Mean Number in Node	(4551.62, 9172.52)	(55367.7, 95209.4)	(128, 128)
Server Utilization	(0.3137, 0.4862)	(0.3323, 0.4673)	(128, 128)

Confidence Intervals Using Batch Means: Util = 0.5, alpha = 0.05

	Trace Driven	**Exp Driven**	**K(T, E)**
Mean Delay Time	(1194577.6, 2284876.9)	(9359360.9, 15222652.9)	(128, 128)
Mean Wait Time	(1194616.4, 2284925.7)	(9359398.2, 15222703.2)	(128, 128)
Mean Queue Length	(13654.4, 26116.9)	(106980.5, 173999.9)	(128, 128)
Mean Number in Node	(13654.8, 26117.4)	(106980.9, 174000.4)	(128, 128)
Server Utilization	(0.3984, 0.6016)	(0.4069, 0.5927)	(128, 128)

Confidence Intervals Using Batch Means: Util = 0.6, alpha = 0.06

	Trace Driven	**Exp Driven**	**K**
Mean Delay Time	(2748126, 5632298)	(13213012, 23938335)	(64, 128)
Mean Wait Time	(2748169, 5632360)	(13213054, 23938399)	(64, 128)
Mean Queue Length	(31411.9, 64379.0)	(151029.0, 273622.9)	(64, 128)
Mean Number in Node	(31412.5, 64379.7)	(151029.5, 273623.7)	(64, 128)
Server Utilization	(0.4698, 0.7302)	(0.4341, 0.7660)	(128, 256)

Homework 5 Alfred J. Williams
Fall, 2020
Appendix:

1. Codes

   (a) Long Run

```
LongRun = function(ArrivalTimes, ServiceTimes){
  ti = 0.0 #initial time
  tf = ComparrivalTimes[length(ComparrivalTimes)]
  Inft = (100*tf) #Some arbitrary large number

  #Initialize job statistics

  tinn = 0  #time integrated number in node
  tinq = 0  #time integrated number in queue
  tins = 0  #time integrated number in service

  #Initialize Time Statistics

  tarr = -1       #next arrival time
  tcomp = -1    #next completion time
  tcurr = -1    #current time
  tnext = -1    #next event time
  tlast = -1    #last arrival time

  #initialize vector of stored departure times

  DepTimes = t = AvgQueue =  AvgNode = AvgServ = numeric(0)

  #Initialize Counters for departed jobs and number of jobs in node
  i = n = 0
  ind = 1
  t[1] = AvgQueue[1] = AvgNode[1] = AvgServ[1] = 0

  #Set the first clock
  tcurr = ti                    #set the clock
  tarr = ArrivalTimes[1]#Schedule the first arrival
  tcomp = Inft                  #The first event can't be a completion



  while(tarr < tf || n > 0){

    tnext = min(tarr, tcomp)  #grab the next event, either arrival or completion
    if(n > 0){
      tinn = tinn + (tnext - tcurr)*n
      tinq = tinq + (tnext - tcurr)*(n-1)
      tins = tins + (tnext - tcurr)
      t[i + n + 1] = tnext
      AvgNode[i + n + 1] = tinn
      AvgQueue[i + n + 1] = tinq
```

```
      AvgServ[i + n + 1] = tins
    }

    tcurr = tnext #advance the clock

    if(tcurr == tarr){  #if we have an arrival,
      n = n + 1         #add one to the node
      tarr = ArrivalTimes[n+i+1] #get the next arrival times

      if(is.na(tarr) || is.na(tf)){break}
      if(tarr > tf){    #do not process jobs after the door has closed
        tlast = tcurr   #end the clock as the last job arrives
        tarr = Inft     #the next event must be a completion (which can happen after doors clos
      }

      if(n == 1){
        tcomp = tcurr + ServiceTimes[i+1]  #get completion time for first person into system
      }

    }else{
      i = i + 1
      if(tcomp < Inft){DepTimes[i] = tcomp} #store the dep
      n = n-1
      if(n > 0){
        tcomp = tcurr + ServiceTimes[i+1]
      }else{
        tcomp = Inft
      }
    }
  }
  R = list(AvgQueue, AvgNode, t, tcurr, i)
  return(R)
}
```

(b) Confidence Interval

```
ConfidenceIntervals = function(AvgQueue, AvgNode, t, tcurr, i, k, alpha){

  b = length(t)%/%k
  s = seq(from = 1, to = length(t), by = b)
  L1 = list(diff(AvgQueue[s])/b, diff(AvgNode[s])/b)
  L2 = list((i/tcurr)*L1[[1]], (i/tcurr)*L1[[2]],(i/tcurr)*(L1[[2]]-L1[[1]]))
  names(L1) = c("AvgDelays", "AvgWaits")
  names(L2) = c("AvgQueueLength", "AvgNodeNum", "AvgUtil")
  L = c(L1, L2)
  names(L) = c(names(L1), names(L2))

  m = s = e = numeric(0)
  Conf = Cut = C = list(0)

  for(i in 1:5){
    Cut[[i]] = c(names(L)[i], cutoff(L[[i]], 2))
  }

  for(i in 1:5){
    m[i] = mean(L[[i]])
    s[i] = sd(L[[i]])
    e[i] = qt(1 - alpha,df = k-1)*s[i]/sqrt(k)
    Conf[[i]] = c(names(L)[i], m[i]-e[i], m[i]+e[i])
  }

  C = list(Cut, Conf)
  names(C) = c("CuttoffLag", "Confidence")
  return(C)
}
```

(c) Choosing K

```
#Choose the utilization
Util = 0.6


mean = TraceSim[[4]]
OGUtil = TraceSim[[2]]
L = length(AdjCompService)
TraceBatch = LongRun(ComparrivalTimes, (Util/OGUtil)*AdjCompService)
ExpBatch = LongRun(ComparrivalTimes, rexp(L, 1/((Util/OGUtil)*mean)))

AvgQueueT = TraceBatch[[1]]
AvgNodeT = TraceBatch[[2]]
tT = TraceBatch[[3]]
iT = TraceBatch[[5]]
tcurrT = TraceBatch[[4]]
alpha = 0.05

AvgQueueE = ExpBatch[[1]]
AvgNodeE = ExpBatch[[2]]
tE = ExpBatch[[3]]
iE = ExpBatch[[5]]
tcurrE = ExpBatch[[4]]



kk = numeric(0)
IntTrace = IntExp = list(0)
for(j in 1:11){
kk[j] = 32*2^(j-1)
IntTrace[[j]] = ConfidenceIntervals(AvgQueueT, AvgNodeT, tT, tcurrT, iT, kk[j], alpha)
IntExp[[j]] = ConfidenceIntervals(AvgQueueE, AvgNodeE, tE, tcurrE, iE, kk[j], alpha)
}
```