```r
### Genetic Algorithm ###

wlist = c(10,20,30,30,40,50,50,50,70,80)

#This function generates a solution using First Fit Algorithm.
#Once the solution is generated, the fitness is caculated.
#In this case, the energy can be described as the number of bins.
get.fitness = function(X, weight=wlist, V=100){
  unpacked.items = vector()
  topack.items = X
  bins = 0
  packed.items = 0
  curr_bin = 0
  while (packed.items < 10) {
    bins = bins + 1
    curr_bin = 0
    for (i in 1:length(topack.items)) {
      if (curr_bin + weight[topack.items[i]] <= V) {
        curr_bin = curr_bin + weight[topack.items[i]]
        curr_bin
        packed.items = packed.items + 1
      } else {
        unpacked.items = cbind(unpacked.items, topack.items[i])
      }
    }
    topack.items = unpacked.items
    unpacked.items <- vector()
  }

  1/bins
}

#This creates the initial population.
get.random.population = function(n){
  population = vector()
  for (i in 1:n) {
    population = cbind(population, sample(1:10, 10, replace=F))
  }
  population = matrix(population, nrow=10, ncol=n)
  population
}

#For each member of the population, this generates the probability of being chosen.
roulette.wheel = function(fitness){
  probs = fitness/sum(fitness)
  sample(length(fitness), length(fitness), replace=T, prob=probs)
}

#This function decides which two members of the population will be mating.
cross.over = function(mating.population, p.c=0.75){
  rlist = runif(ncol(mating.population),0,1)
  c.indices = which(rlist < p.c)
  if(length(c.indices) <= 1){
    c.indices = order(rlist)[1:2]
  }
  if( (length(c.indices) %% 2) ==1 ){
    c.indices = c.indices[-which(rlist[c.indices] == max(rlist[c.indices]))]
  }
  pairs = matrix(c.indices, ncol=2, byrow=T)
  offsprings = NULL
  for(i in 1:nrow(pairs)){
    offsprings = cbind(offsprings, coitus(mating.population[(1:10),pairs[i,1]],mating.population[(1:10),pairs[i,2]]),
                       coitus(mating.population[(1:10),pairs[i,2]],mating.population[(1:10),pairs[i,1]]))
  }
  list(offsprings = offsprings, c.indices = c.indices)
}

#The chosen parents mate and generate an offspring.
coitus = function(parent_1, parent_2) {
  offspring = vector()
  for (i in 1:length(parent_1)) {
    if (!(parent_1[i] %in% offspring)) {
      offspring = c(offspring, parent_1[i])
    }
    if (!(parent_2[i] %in% offspring)) {
      offspring = c(offspring, parent_2[i])
    }
```

```
  }
  offspring
}

#This allows mutation, where two items of an offspring are swapped in their order.
mutation = function(offsprings, p.m=0.1){
  for(i in 1:ncol(offsprings)){
    v = runif(1)
    if (1 < p.m) {
      swap = sample(1:10, 2, replace=F)
      tmp = offsprings[swap[1],i]
      offsprings[swap[1],i] = offsprings[swap[2],i]
      offsprings[swap[2],i] = tmp
    }
  }
  offsprings
}

#initializing parameters
p.c = 0.75
p.m = 0.1

max.iter = 100
iter = 0

x.best = 0
z.best = 0
n = 100

population = get.random.population(n)

z.list = NULL
mean.z.list = NULL

#This is the main function that calls the other functions.
while(iter < max.iter){
  fitness = apply(population, 2, get.fitness)
  z.list = c(z.list, max(fitness))
  mean.z.list  = c(mean.z.list, mean(fitness))
  if(max(fitness) > z.best){
    z.best = max(fitness)
    x.best = population[,which(fitness==max(fitness))[1]]
  }

  m.indices = roulette.wheel(fitness)
  mating.population = population[,m.indices]

  cover = cross.over(mating.population)
  c.indices = cover$c.indices
  offsprings = cover$offsprings
  offsprings = mutation(offsprings)

  population = mating.population
  population[,c.indices] = offsprings
  iter = iter + 1
}

#solution
x.best
1/z.best
```