Apple Bread Problem 2

Name: Lavanis A
Roll No.: bmat2353
Q1: A_n's length grows by 1 after every step.
Q2:

```
library(ggplot2)
library(gganimate)

A = c(0)
Annihilation_Simulation_3 = function(n){
  for (i in 1:n){
    x = c()
    x[1] = A[i] + 1
    x[2] = A[i] - 1
    x[3] = sample(c(x[1] + 1, x[2] -1), size = 1)
    cat("Iteration number:",i, "\n")
    cat("The 3 Added elements are:",x, "\n")
    A[i+1] = sample(x, size = 1)
    cat("The 2 Deleted elements are:",x[x != A[i+1]], "\n")
    cat("The simulation of A:",A, "\n")
  }
  return(A)
}
#Sample output for Simulation by taking n = 10.
x<- Annihilation_Simulation_3(10)


## Iteration number: 1
## The 3 Added elements are: 1 -1 2
## The 2 Deleted elements are: -1 2
## The simulation of A: 0 1
## Iteration number: 2
## The 3 Added elements are: 2 0 -1
## The 2 Deleted elements are: 2 -1
## The simulation of A: 0 1 0
## Iteration number: 3
## The 3 Added elements are: 1 -1 2
## The 2 Deleted elements are: 1 -1
## The simulation of A: 0 1 0 2
## Iteration number: 4
## The 3 Added elements are: 3 1 0
## The 2 Deleted elements are: 1 0
## The simulation of A: 0 1 0 2 3
## Iteration number: 5
## The 3 Added elements are: 4 2 1
## The 2 Deleted elements are: 2 1
## The simulation of A: 0 1 0 2 3 4
## Iteration number: 6
## The 3 Added elements are: 5 3 2
## The 2 Deleted elements are: 3 2
```

```
## The simulation of A: 0 1 0 2 3 4 5
## Iteration number: 7
## The 3 Added elements are: 6 4 3
## The 2 Deleted elements are: 4 3
## The simulation of A: 0 1 0 2 3 4 5 6
## Iteration number: 8
## The 3 Added elements are: 7 5 4
## The 2 Deleted elements are: 7 4
## The simulation of A: 0 1 0 2 3 4 5 6 5
## Iteration number: 9
## The 3 Added elements are: 6 4 3
## The 2 Deleted elements are: 6 3
## The simulation of A: 0 1 0 2 3 4 5 6 5 4
## Iteration number: 10
## The 3 Added elements are: 5 3 6
## The 2 Deleted elements are: 5 6
## The simulation of A: 0 1 0 2 3 4 5 6 5 4 3
```

Just to represent the simulation, I've taken smaller values for n. For the animation part, I used the n = 1000. Below is the code for the animation, The video is sent as an attachment to the email.

```
x<- Annihilation_Simulation_3(1000)
simulation_data = data.frame(0:1000, x)
names(simulation_data) = c("n", "site")
animation <- ggplot(simulation_data, aes(x = site, fill = as.factor(site))) +
  geom_histogram(stat = "count", binwidth = 1, show.legend = FALSE) +
  ylim(0,10)
  labs(title = "Step: {frame_time}",
       x = "Site", y = " ") +
  theme_minimal() +
  transition_time(n) +
  ease_aes('linear')

# Save animation as video
animate(animation, duration = 120, fps = 20,
        renderer = av_renderer("annihilation_creation3.mp4"))
```

The animation provided represents the best dynamic simulation I could achieve for A_n. That is I could only animate the ending values of A_n for every n. Although I attempted more complex approaches, either the output from R was unsatisfactory or the computation time became unmanageable. I acknowledge that there are likely alternative functions or methods that could have potentially produced better results, but unfortunately, I may not have encountered them yet.

Q3: (a) (Step 1)

```
g2A = function(x){
  c = 0
  l = length(x)
  for (i in 1:l){
    for (j in i:l){
      if (abs(x[i] - x[j]) -
          length(unique(x[((x[i] > x) & (x < x[j])) | ((x[j] < x) & (x > x[j]))]))
 > 2){
        c = c + 1
      }

    }
  }
  return(c)
}
LA = function(x){
  xmin = min(x)
  xmax = max(x)
  return(xmax - xmin - length(unique(x)) + 1)
}

Annihilation_Simulation_3 = function(n){
  A = c(0)
  for (i in 1:n){
    x = c()
    x[1] = A[i] + 1
    x[2] = A[i] - 1
    x[3] = sample(c(x[1] + 1, x[2] -1), size = 1)
    A[i+1] = sample(x, size = 1)
  }
  return(A)
}
g2_function = c()
l_function = c()
for(i in 1:1000){
  x = unique(Annihilation_Simulation_3(i))
  y = g2A(x)
  z = LA(x)
 g2_function[i] = y
 l_function[i] = z
}
print(g2_function[1:10])


##  [1] 0 0 0 0 1 3 6 0 0 0


print(l_function[1:10])


##  [1] 0 0 0 1 0 1 2 0 1 0
```

(b) (Step 2) I'll continue the question with n=100 due to the significantly long computation time. (When I attempted to compute the results for n=1000 in parts (b), (c), and (d), the computation took over 15 minutes without producing any output.)

```
g2n = c()
Lan = c()
for(i in 1:100){
  g2n[i] = sum(replicate(100, g2A(unique(Annihilation_Simulation_3(i)))))/100
  Lan[i] = sum(replicate(100, LA(unique(Annihilation_Simulation_3(i)))))/100
}
Lan
```

```
##   [1] 0.33 0.37 0.48 0.47 0.38 0.43 0.63 0.51 0.52 0.63 0.51 0.56 0.40 0.56 0.57
##  [16] 0.66 0.58 0.50 0.62 0.62 0.67 0.64 0.55 0.51 0.82 0.46 0.65 0.59 0.65 0.46
##  [31] 0.66 0.60 0.53 0.45 0.82 0.49 0.50 0.59 0.77 0.36 0.67 0.70 0.65 0.74 0.59
##  [46] 0.83 0.66 0.56 0.89 0.79 0.68 0.74 0.55 0.84 0.66 0.49 0.63 0.64 0.51 0.63
##  [61] 0.63 0.67 0.75 0.68 0.79 0.59 0.57 0.64 0.62 0.59 0.62 0.65 0.65 0.86 0.73
##  [76] 1.04 0.76 0.67 0.68 0.66 0.62 0.62 0.70 0.59 0.75 0.72 0.55 0.75 0.67 0.78
##  [91] 0.76 0.57 0.69 0.65 0.72 0.73 0.73 0.60 0.70 0.66
```

```
g2n
```

```
##   [1]  0.00  0.10  0.25  0.64  0.61  0.87  1.05  1.35  1.95  1.98  1.97  1.60
##  [13]  2.21  2.39  2.87  2.42  2.77  3.73  4.17  4.93  5.14  4.23  5.63  4.49
##  [25]  6.20  7.33  8.12  5.87  7.35  6.56  6.21  7.00  8.29  9.45  6.60  8.29
##  [37]  9.43  8.32 10.07 10.97 11.02  9.01 12.50 13.62  9.23 12.65 12.51 11.27
##  [49] 12.82 12.21 14.41 10.61 13.28 12.94 15.26 15.03 17.77 17.45 16.79 11.64
##  [61] 15.19 14.21 19.41 14.78 13.79 21.05 16.40 20.70 18.09 16.70 18.60 21.23
##  [73] 29.06 21.89 13.70 21.74 25.91 19.75 24.57 21.37 17.39 21.69 22.05 21.81
##  [85] 19.90 24.63 21.87 27.51 26.67 24.00 26.31 27.36 25.84 25.90 24.76 28.45
##  [97] 24.10 31.05 28.10 24.16
```

```
Lan = Lan[2:100]
```

(c) (Step 3)

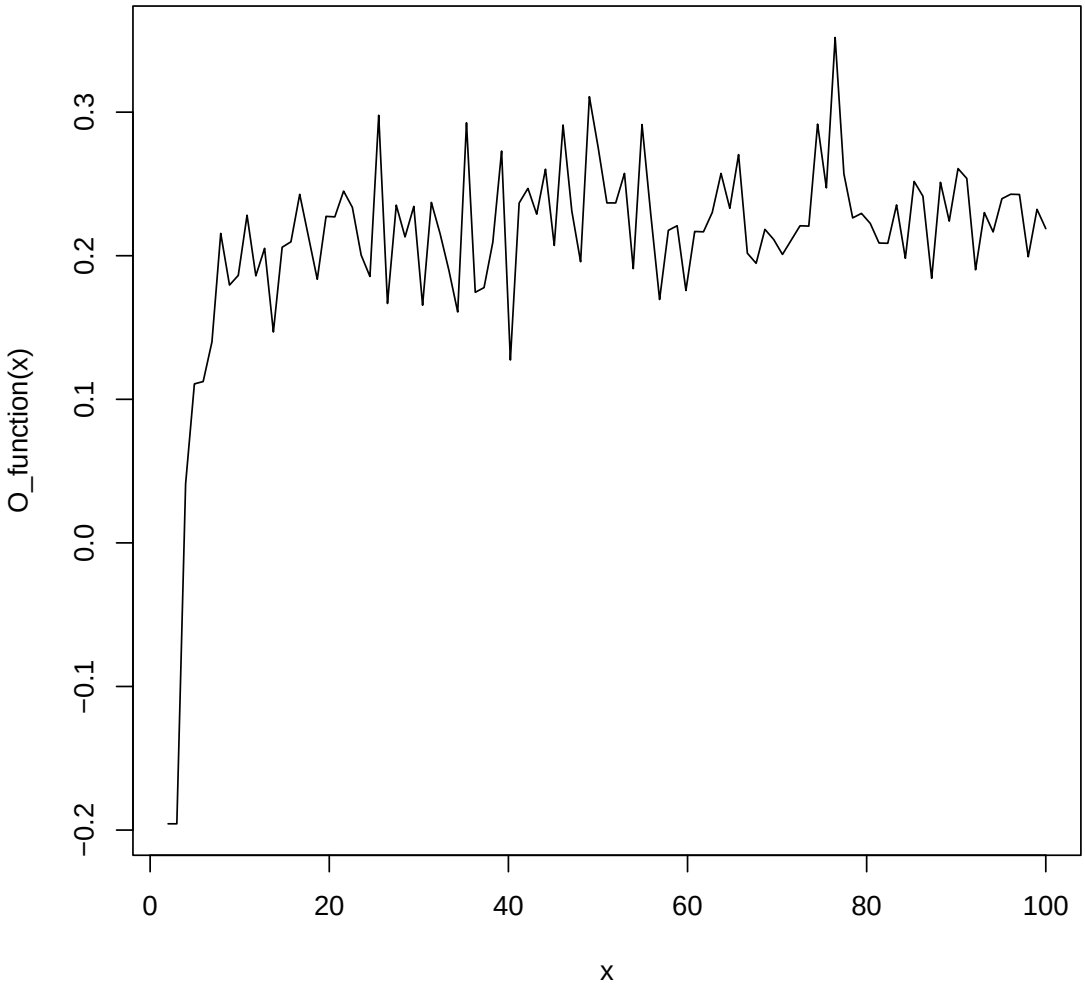
```
log_function = c()
for(i in 2:100){
  log_function[i-1] = log(i)/log(log(i))
}
val = Lan/log_function
val
```

```
##  [1] -0.19564356  0.04109089  0.11073990  0.11235991  0.13996029  0.21553399
##  [7]  0.17955334  0.18629930  0.22819588  0.18601380  0.20513111  0.14689393
## [13]  0.20592057  0.20968979  0.24275355  0.21319209  0.18360703  0.22739454
## [19]  0.22707536  0.24501068  0.23365754  0.20045731  0.18555293  0.29780782
## [25]  0.16676174  0.23521464  0.21311541  0.23436358  0.16555895  0.23711694
```

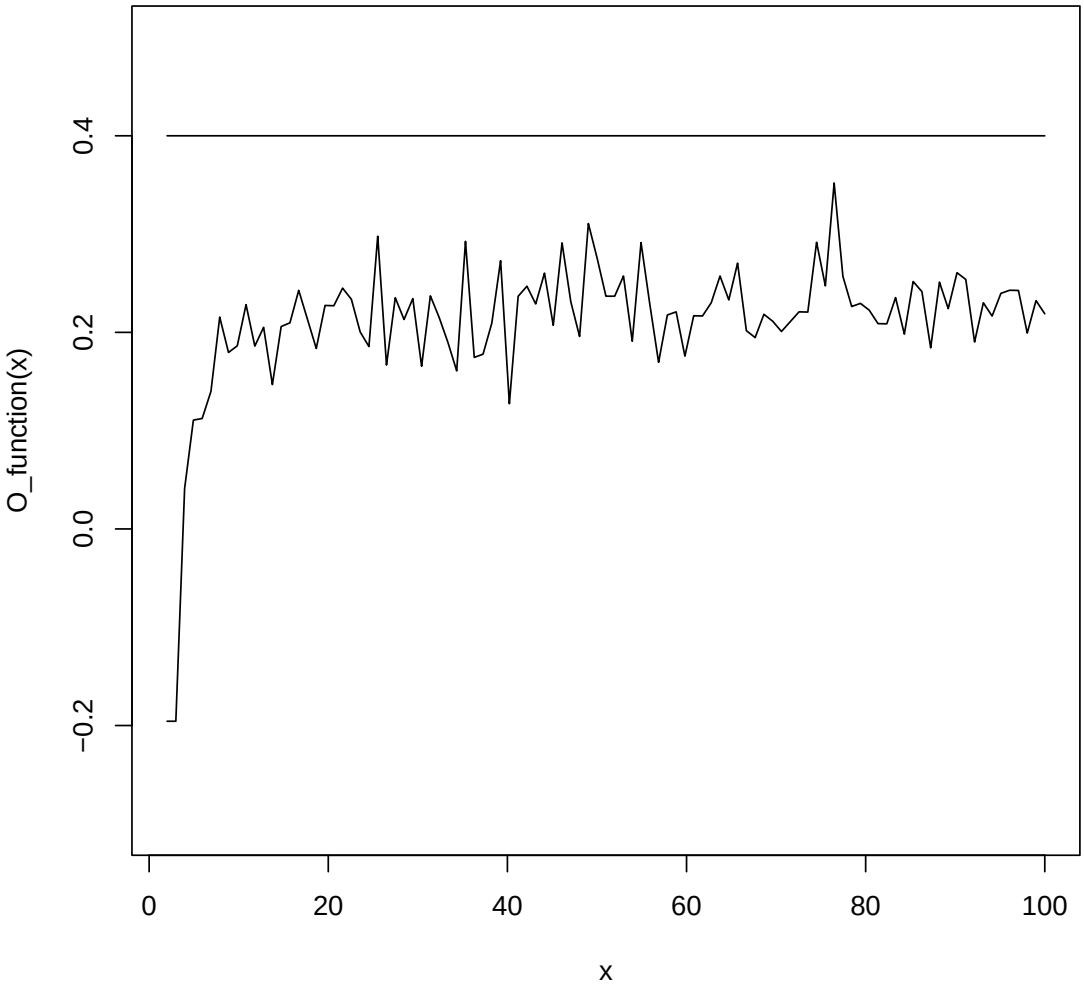
```
## [31]  0.21517941  0.18974226  0.16082299  0.29255404  0.17452375  0.17778887
## [37]  0.20944631  0.27290262  0.12738724  0.23670906  0.24692488  0.22893750
## [43]  0.26024476  0.20718521  0.29103883  0.23109597  0.19580391  0.31075446
## [49]  0.27545931  0.23678329  0.25733226  0.19100908  0.29134504  0.22862166
## [55]  0.16952060  0.21768455  0.22086923  0.17579276  0.21689728  0.21664261
## [61]  0.23013082  0.25731471  0.23303556  0.27043116  0.20174570  0.19469549
## [67]  0.21837140  0.21132365  0.20088837  0.21088536  0.22086432  0.22064210
## [73]  0.29163628  0.24730858  0.35198805  0.25697528  0.22632918  0.22949191
## [79]  0.22253572  0.20885716  0.20866791  0.23538169  0.19821730  0.25175023
## [85]  0.24147062  0.18429844  0.25110269  0.22412994  0.26071044  0.25381652
## [91]  0.19020733  0.23006532  0.21655520  0.23968694  0.24282575  0.24263757
## [97]  0.19927509  0.23231089  0.21887108
```

Clearly if c $= 1$, then $\frac{L_n}{h_n} < 1$ for every $n > 1$, thus $L_n = O(h_n)$.

```
O_function = function(x){
  return(val[x-1])
}
curve(O_function, from = 2, to = 100)
```



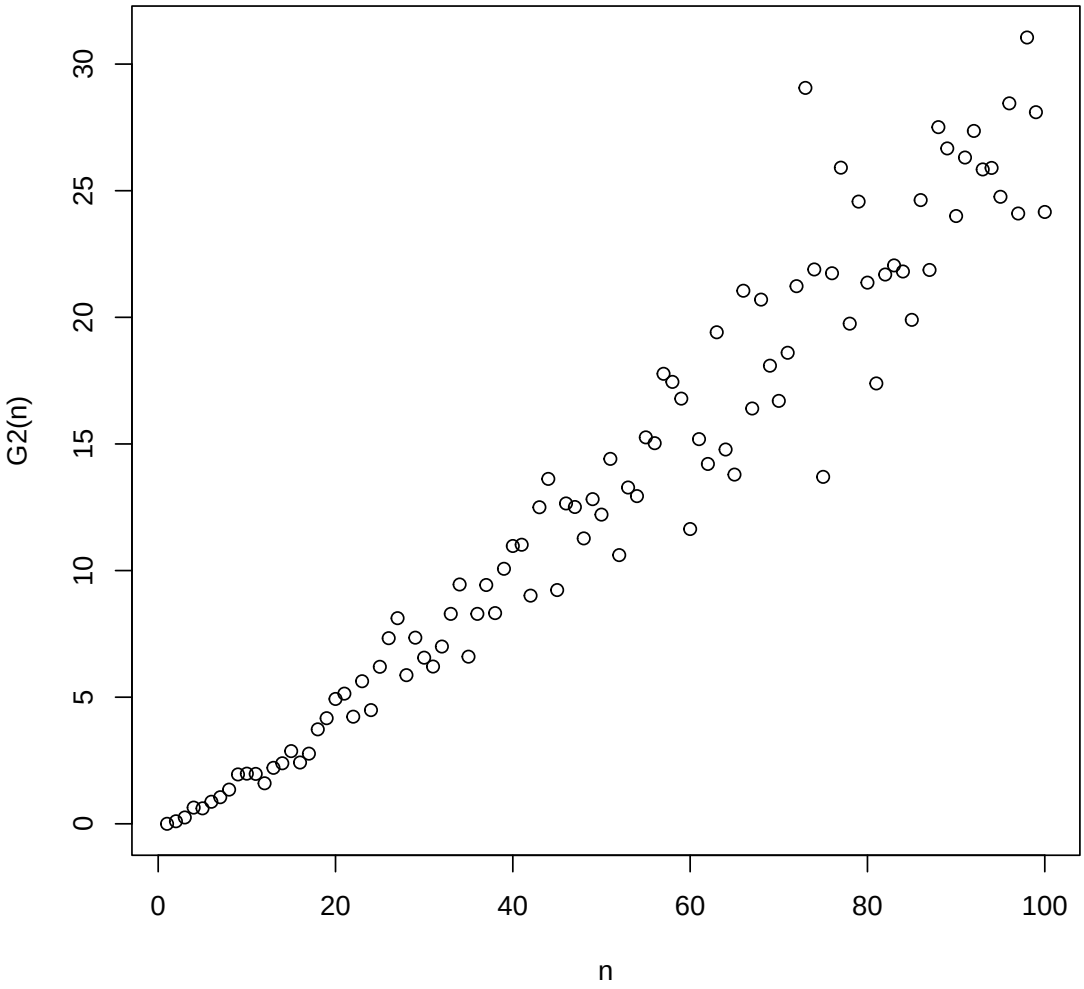

```r
plot.new()
curve(O_function, from = 2, to = 100, ylim = c(-0.3,0.5))
O_function = function(x){
  return(x-x+0.4)
}
curve(O_function, from = 2, to = 100, add = TRUE)
```



Since every point in the curve lies below the line y = 0.4, setting the upper bound for the function. Thus, $\frac{L_n}{h_n} < 0.4 < 1$ for every $n > 1$, thus $L_n = O(h_n)$

(d) (Step 4)

```
plot(1:100,g2n, xlab = "n", ylab = "G2(n)")
```



I'm apologetic that I'm not very sure about my answer for part (d) or Step 4 of the problem. I've tried my best to solve this problem, and I've learned a lot along the way.