HW4

John Horton

September 27, 2015

Question 1:

Part a:

#In parts a through d, we use the c() function to create the desired vectors, as specified in the prompts  
#a is a vector containing the integers in the interval [1,20]  
a <- c(1:20)  
print(a)

## [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

Part b:

#b is a vector containing those same integers in a, now in reverse  
b <- c(20:1)  
print(b)

## [1] 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

Part c:

#c is nearly the concatenation of a and b, except the middle 20 appears only once  
c <- c(1:20,19:1)  
print(c)

## [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 19 18 17  
## [24] 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

Part d:

#tmp created, as per the prompt  
tmp <- c(4,6,3)  
print(tmp)

## [1] 4 6 3

Part e:

#In parts e through g, we use the rep() function to create the desired vectors, as specified in the prompts  
#If we specify our desired output length as 30, the vector (4,6,3) will be repeated 10 times, implying there are 10 occurrences of 4  
e <- rep(tmp, length.out = 30)  
print(e)

## [1] 4 6 3 4 6 3 4 6 3 4 6 3 4 6 3 4 6 3 4 6 3 4 6 3 4 6 3 4 6 3

Part f:

#If we specify our desired output length as 31, the vector (4,6,3) will be repeated 10 times as above; the first element of (4,6,3) tacked on the end as the 31'st element, giving us the extra 4 we need  
f <- rep(tmp, length.out = 31)  
print(f)

## [1] 4 6 3 4 6 3 4 6 3 4 6 3 4 6 3 4 6 3 4 6 3 4 6 3 4 6 3 4 6 3 4

Part g:

#Here we simply specify the amount of times we would like to repeat each element in the vector (4,6,3) by passing rep() the vector created by c(10,20,30)  
g <- rep(tmp, c(10,20,30))  
print(g)

## [1] 4 4 4 4 4 4 4 4 4 4 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 3 3 3 3 3  
## [36] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3

Question 2:

#The vector x contains our desired values of x as per the prompt, i.e. x is the vector (3,3.1,...,5.9,6). To create x, we pass seq() the first value of our vector, the final value, and the increment represented as by=0.1  
x <- seq(3.0,6.0,by=0.1)  
#Now that we have our vector x, we perform the desired operation  
y <- (exp(x))\*cos(x)  
print(y)

## [1] -19.884531 -22.178753 -24.490697 -26.773182 -28.969238 -31.011186  
## [7] -32.819775 -34.303360 -35.357194 -35.862834 -35.687732 -34.685042  
## [13] -32.693695 -29.538816 -25.032529 -18.975233 -11.157417 -1.362099  
## [19] 10.632038 25.046705 42.099201 61.996630 84.929067 111.061586  
## [25] 140.525075 173.405776 209.733494 249.468441 292.486707 338.564378  
## [31] 387.360340

Question 3:

Part a:

#We create the vector x, as in Question 2  
x <- seq(3,36,by=3)  
#We create the vector y by subtracting 2 from each value in x  
y <- x-2  
#Now that we have vectors x and y to represent the exponents called for in the prompt, we can perform the desired operation  
z <- (0.1^x)\*(0.2^y)  
print(z)

## [1] 2.000000e-04 1.600000e-09 1.280000e-14 1.024000e-19 8.192000e-25  
## [6] 6.553600e-30 5.242880e-35 4.194304e-40 3.355443e-45 2.684355e-50  
## [11] 2.147484e-55 1.717987e-60

Part b:

#We create the vector x to represent the exponents and denominators called for in the question prompt   
x <- c(1:25)  
#Now that we have a vector to represent the required exponents and denominators, we can perform the desired operation  
y <- (2^x)/x  
print(y)

## [1] 2.000000e+00 2.000000e+00 2.666667e+00 4.000000e+00 6.400000e+00  
## [6] 1.066667e+01 1.828571e+01 3.200000e+01 5.688889e+01 1.024000e+02  
## [11] 1.861818e+02 3.413333e+02 6.301538e+02 1.170286e+03 2.184533e+03  
## [16] 4.096000e+03 7.710118e+03 1.456356e+04 2.759411e+04 5.242880e+04  
## [21] 9.986438e+04 1.906502e+05 3.647221e+05 6.990507e+05 1.342177e+06

Question 4:

Part a:

#We create the vector x to represent the values i=[10,100] over which we take the desired sum  
x <- c(10:100)  
#We create the vector y to represent the inner part of the sum at each i in [10,100]  
y <- (x^3 + 4\*x^2)  
#We now have the desired sum value at each i in [1,100] contained in the vector y; to perform the full sum, we simply take the sum of all values in the vector y  
z <- sum(y)  
print(z)

## [1] 26852735

Part b:

#We create the vector x to represent the values i=[1,25] over which we take the desired sum  
x <- c(1:25)  
#We create the vector y to represent the inner part of the sum at each i in [1,25]  
y <- (2^x)/x + (3^x)/(x^2)  
#We now have the desired sum value at each i in [1,25] contained in the vector y; to perform the full sum. we simply take the sum of all values in the vector y  
z <- sum(y)  
print(z)

## [1] 2129170437

Question 5:

Part a:

#We create the vector x to represent the values 1,2,...,30 as in "label 1","label 2",...,"label 30"  
x <- c(1:30)  
#The paste function here concatenates "label" and x with a space in between (default) for each value of x  
y <- paste("label",x)  
print(y)

## [1] "label 1" "label 2" "label 3" "label 4" "label 5" "label 6"   
## [7] "label 7" "label 8" "label 9" "label 10" "label 11" "label 12"  
## [13] "label 13" "label 14" "label 15" "label 16" "label 17" "label 18"  
## [19] "label 19" "label 20" "label 21" "label 22" "label 23" "label 24"  
## [25] "label 25" "label 26" "label 27" "label 28" "label 29" "label 30"

Part b:

#We create the vector x to represent the values 1,2,...,30 as in "fn1","fn2",...,"fn30"  
x <- c(1:30)  
#The paste function here concatenates "fn" and x for each value of x; as we have specified no separation between characters with sep="", there will not be a space between "fn" and the number following  
y <- paste("fn",x,sep = "")  
print(y)

## [1] "fn1" "fn2" "fn3" "fn4" "fn5" "fn6" "fn7" "fn8" "fn9" "fn10"  
## [11] "fn11" "fn12" "fn13" "fn14" "fn15" "fn16" "fn17" "fn18" "fn19" "fn20"  
## [21] "fn21" "fn22" "fn23" "fn24" "fn25" "fn26" "fn27" "fn28" "fn29" "fn30"

Question 6:

Part a:

#Here we execute the code to generate our two sets of random numbers, as in the prompt  
set.seed(50)  
xVec <- sample(0:999,250,replace=T)  
yVec <- sample(0:999,250,replace=T)  
#We initialize the vector zVec to a simple vector containing 249 values, so we may later perform the desired operations by overwriting each element zVec[i] rather than creating new vectors  
zVec <- c(1:249)  
  
#We loop through each value in zVec so we may overwrite each value with the output of our desired operation  
for(i in 1:249){  
 zVec[i] <- yVec[i+1] - xVec[i]  
}  
  
print(xVec)

## [1] 708 437 200 767 513 44 699 646 42 107 390 269 640 77 277 676 835  
## [18] 364 74 168 616 193 710 842 309 650 577 257 324 368 358 408 437 618  
## [35] 222 627 121 701 373 458 363 836 278 93 55 700 954 458 713 803 996  
## [52] 765 639 299 358 425 715 525 511 266 578 655 197 585 129 38 724 61  
## [69] 136 944 507 995 661 74 967 148 657 956 652 956 543 17 339 469 544  
## [86] 19 1 680 537 645 691 688 828 760 48 294 69 807 311 668 505 964  
## [103] 632 8 24 862 10 614 840 353 878 72 193 113 82 322 91 789 444  
## [120] 986 624 18 537 554 515 460 263 42 76 256 359 189 807 457 99 274  
## [137] 543 324 176 477 541 160 260 174 48 415 707 625 530 407 216 224 395  
## [154] 977 828 461 148 293 660 38 137 224 852 743 683 545 353 371 866 452  
## [171] 811 768 339 203 478 49 20 880 480 996 894 357 900 603 667 787 972  
## [188] 457 467 324 928 109 365 987 572 280 113 702 963 405 63 621 517 446  
## [205] 533 190 638 275 865 435 501 669 124 14 920 308 84 523 5 863 860  
## [222] 120 206 399 29 256 678 59 497 188 127 258 376 171 781 870 110 957  
## [239] 285 382 34 403 631 197 179 545 123 760 238 178

print(yVec)

## [1] 709 871 315 517 621 930 437 948 157 783 878 471 671 91 415 860 273  
## [18] 768 581 381 47 347 229 4 279 411 698 974 554 279 216 855 813 776  
## [35] 218 721 538 332 31 460 532 917 985 95 705 248 247 884 317 840 94  
## [52] 288 43 575 687 174 213 957 955 786 938 428 930 101 330 641 615 988  
## [69] 500 285 28 881 106 329 398 414 542 570 881 997 221 488 117 299 484  
## [86] 823 428 791 133 50 246 72 520 643 779 693 845 296 441 553 815 752  
## [103] 465 18 766 87 635 257 993 368 919 116 224 686 473 151 512 635 613  
## [120] 660 310 419 800 428 743 282 965 44 330 19 743 615 489 615 194 803  
## [137] 948 760 604 193 409 800 772 133 175 593 184 516 287 863 902 195 220  
## [154] 689 309 14 881 941 924 593 693 280 835 632 225 398 872 876 358 187  
## [171] 211 850 961 681 791 947 117 915 222 712 665 921 798 167 421 268 866  
## [188] 503 828 942 589 521 320 424 13 482 498 509 216 0 78 488 841 645  
## [205] 681 827 83 273 421 277 884 67 890 970 400 10 469 290 632 717 529  
## [222] 426 127 846 49 952 609 99 284 824 598 695 63 293 325 295 675 777  
## [239] 813 557 792 580 783 72 611 853 738 345 668 791

print(zVec)

## [1] 163 -122 317 -146 417 393 249 -489 741 771 81 402 -549 338  
## [15] 583 -403 -67 217 307 -121 -269 36 -706 -563 102 48 397 297  
## [29] -45 -152 497 405 339 -400 499 -89 211 -670 87 74 554 149  
## [43] -183 612 193 -453 -70 -141 127 -709 -708 -722 -64 388 -184 -212  
## [57] 242 430 275 672 -150 275 -96 -255 512 577 264 439 149 -916  
## [71] 374 -889 -332 324 -553 394 -87 -75 345 -735 -55 100 -40 15  
## [85] 279 409 790 -547 -487 -399 -619 -168 -185 19 645 551 227 -366  
## [99] 242 147 247 -499 -614 758 63 -227 247 379 -472 566 -762 152  
## [113] 493 360 69 190 544 -176 216 -676 -205 782 -109 189 -233 505  
## [127] -219 288 -57 487 256 300 -192 -263 704 674 217 280 17 -68  
## [141] 259 612 -127 1 545 -231 -191 -338 333 495 -21 -4 294 -668  
## [155] -814 420 793 631 -67 655 143 611 -220 -518 -285 327 523 -13  
## [169] -679 -241 39 193 342 588 469 68 895 -658 232 -331 27 441  
## [183] -733 -182 -399 79 -469 371 475 265 -407 211 59 -974 -90 218  
## [197] 396 -486 -963 -327 425 220 128 235 294 -107 -365 146 -588 449  
## [211] -434 221 846 386 -910 161 206 109 712 -334 -434 7 640 -350  
## [225] 923 353 -579 225 327 410 568 -195 -83 154 -486 -195 667 -144  
## [239] 272 410 546 380 -559 414 674 193 222 -92 553

Part b:

#We loop through each value in zVec so we may overwrite each value with the output of our desired operation  
for(i in 1:249){  
 zVec[i] <- (sin(yVec[i]))/(cos(xVec[i+1]))  
}  
  
print(zVec)

## [1] 0.88603405 -1.44184825 0.82807258 -1.61591717 -0.86017343  
## [6] 20.26356465 -0.79930406 1.72414444 -0.08094240 -0.74895634  
## [11] -2.59866958 -0.37361045 31.11471579 0.12355916 -0.35925226  
## [16] -0.90743608 0.34374436 5.78205917 -2.57418558 -0.78661325  
## [21] -0.59855406 0.98936263 0.33042931 -1.75124647 -0.59435547  
## [26] 1.05374692 0.65497397 -0.11596582 -0.97176537 0.57180267  
## [31] 0.75799030 -0.49259143 -0.99433357 0.05377148 -3.77616264  
## [36] 20.54902944 0.77784817 1.28146891 -0.51650728 6.66902699  
## [41] -0.92970072 -10.93066299 -3.13102962 30.87943423 -1.14281543  
## [46] 0.36757630 1.18479716 0.94594159 0.93339520 0.93632658  
## [51] -11.05384468 2.76893270 0.97488334 -0.08932225 -1.33616578  
## [56] -3.30065552 0.62663162 -1.96486337 0.08653876 0.56695489  
## [61] 44.07630714 -1.11764853 0.11230330 -0.46073106 -0.13860882  
## [66] 0.84026052 2.64708780 -1.63174570 -9.63022830 -2.15553419  
## [71] -0.42770826 3.24955062 -4.23453154 0.93067452 -0.88388390  
## [76] 0.69339350 1.72841015 -8.22082884 1.69276461 1.02074555  
## [81] -3.21968328 -0.90739226 1.11331935 0.59579467 0.19571363  
## [86] -0.17975474 4.38929818 0.64431266 -1.54509170 -0.26536991  
## [91] -0.81679156 1.34164181 -1.03400420 -1.33639979 -0.44444499  
## [96] 0.96777754 -0.09545121 -0.63686070 -2.30844090 -0.11384497  
## [101] 1.08800453 1.06851885 -0.30428029 -1.77044888 -1.45269351  
## [106] 0.97943716 -2.15021752 1.56128032 0.61018741 5.59692239  
## [111] -1.03020002 -1.14632240 -0.81548097 0.95359082 74.12815803  
## [116] -0.20329495 -0.08875385 -0.76023984 -0.42372635 -0.68385723  
## [121] 1.28860542 0.94117702 1.89561343 0.69369539 4.15021756  
## [126] -1.08026240 1.26615554 0.02147428 3.32694398 0.22930300  
## [131] 1.14217476 0.73847767 8.72339712 -17.15727240 0.90435970  
## [136] 1.07791792 0.75391899 -0.26297571 0.83894657 -1.22542984  
## [141] -0.57277292 -1.22429033 2.10719833 -1.35745285 -0.84117115  
## [146] -0.69663176 -0.99207337 -1.17363312 -5.50814669 -1.12309426  
## [151] 0.60767585 0.32903697 -0.08845387 -4.42251048 -1.31360561  
## [156] -1.05268827 -1.45007537 -1.03184453 0.38034305 2.06381128  
## [161] -1.64568068 0.47938401 46.18666528 1.75988821 14.03349520  
## [166] 1.99884446 -1.02170635 1.02445028 -0.15250370 -1.11793279  
## [171] -4.12228606 1.02355677 0.89546497 0.74732250 -2.09533197  
## [176] -2.40630344 -0.73530615 0.90759126 -0.87474163 -4.22536917  
## [181] -2.04450866 -7.41320483 0.03607946 -0.85674969 -0.85648584  
## [186] 2.58973778 8.68248704 -0.74202802 1.07347586 1.37638585  
## [191] 1.73104746 -0.57596355 -0.49915725 0.11786229 -0.45584137  
## [196] -0.97726281 -6.86428063 -0.60929448 -0.72132361 0.00000000  
## [201] 1.00734878 4.20789995 -0.81616263 -1.72455176 10.00784534  
## [206] 0.71310632 8.77005056 -0.64297796 0.24086573 -6.12424634  
## [211] 0.94848253 9.22132979 -5.85933168 -0.77292827 -0.85749485  
## [216] 0.80000340 -10.45187777 2.91489552 0.86914823 0.93956496  
## [221] 1.15020196 -4.25009579 -0.97278301 1.05669698 23.96919924  
## [226] -0.11659711 0.58615433 -1.23512544 1.08111948 3.37846777  
## [231] 0.96204558 -1.18727215 0.77801767 2.39161655 1.01270315  
## [236] 0.30508064 -1.13987140 1.35085069 2.13213714 0.95034702  
## [241] 0.48941676 -1.03804260 1.11768517 -0.25446052 -15.07630921  
## [246] 1.12429826 0.28067653 -0.75125301 -1.91160477

Part c:

#We recreate zVec as we now only have 248 output entries  
zVec <- c(1:248)  
#We loop through each value in zVec so we may overwrite each value with the output of our desired operation  
for(i in 1:248){  
 zVec[i] <- xVec[i] + 2\*xVec[i+1] - xVec[i+2]  
}  
  
print(zVec)

## [1] 1382 70 1221 1749 -98 796 1949 623 -134 618 288 1472 517 -45  
## [15] 794 1982 1489 344 -206 1207 292 771 2085 810 1032 1547 767 537  
## [29] 702 676 737 664 1451 435 1355 168 1150 989 926 348 1757 1299  
## [43] 409 -497 501 2150 1157 1081 1323 2030 1887 1744 879 590 493 1330  
## [57] 1254 1281 465 767 1691 464 1238 805 -519 1425 710 -611 1517 963  
## [71] 1836 2243 -158 1860 606 506 1917 1304 2021 2025 238 226 733 1538  
## [85] 581 -659 824 1109 1136 1339 1239 1584 2300 562 567 -375 1372 761  
## [99] 1142 714 1801 2220 624 -806 1738 268 398 1941 668 2037 829 345  
## [113] 337 -45 635 -285 1225 691 1792 2216 123 538 1130 1124 1172 944  
## [127] 271 -62 229 785 -70 1346 1622 381 104 1036 1015 199 589 1399  
## [141] 601 506 560 -145 171 1204 1427 1278 1128 615 269 37 1521 2172  
## [155] 1602 464 74 1575 599 88 -267 1185 1655 1564 1420 880 229 1651  
## [169] 959 1306 2008 1243 267 1110 556 -791 1300 844 1578 2427 708 1554  
## [183] 1439 1150 1269 2274 1419 1067 187 2071 781 -148 1767 1851 1019 -196  
## [197] 554 2223 1710 -90 788 1209 876 1322 275 1191 323 1570 1234 768  
## [211] 1715 903 -768 1546 1452 -47 1125 -330 871 2463 894 133 975 201  
## [225] -137 1553 299 865 746 184 267 839 -63 863 2411 133 1739 1145  
## [239] 1015 47 209 1468 846 10 1146 31 1405 1058

Part d:

#We loop through each value in zVec so we may overwrite each value with the output of our desired operation  
for(i in 1:249){  
 zVec[i] <- (exp(-xVec[i+1]))/(xVec[i] + 10)  
}  
sum <- sum(zVec)  
print(sum)

## [1] 0.01269872

Question 7:

Part a:

#We sort the vector yVec so we may see which values are larger than 600  
sortY <- sort(yVec,FALSE)  
#We initialize the index variable i to 1  
i <- 1  
#If the i'th value in sortY is <=600, we increase i by 1; when we come out of the loop, i will equal the index of the first element in sortY that is greater than 600  
while(sortY[i]<=600){  
 i <- i+1  
}  
#We create the vector largeY to contain all elements from sortY that are greater than 600; as i is the index of the first element in sortY that is greater than 600, we need to remove all elements in sortY that come before index i; the line below does so by creating largeY out of sortY without those elements sortY at indices 1 through i-1  
largeY <- sortY[-c(1:i-1)]  
print(largeY)

## [1] 604 609 611 613 615 615 615 621 632 632 635 635 641 643 645 660 665  
## [18] 668 671 675 681 681 686 687 689 693 693 695 698 705 709 712 717 721  
## [35] 738 743 743 752 760 766 768 772 776 777 779 783 783 786 791 791 791  
## [52] 792 798 800 800 803 813 813 815 823 824 827 828 835 840 841 845 846  
## [69] 850 853 855 860 863 866 871 872 876 878 881 881 881 884 884 890 902  
## [86] 915 917 919 921 924 930 930 938 941 942 947 948 948 952 955 957 961  
## [103] 965 970 974 985 988 993 997

Part b:

#We create the vector sortIndex to store the yVec indices corresponding to each sorted element in sortY  
sortIndex <- sort(yVec,FALSE,index.return=TRUE)$ix  
#Our index variable i has not changed from above, so we may use it again to isolate the yVec indices corresponding to elements in sortY that are greater than 600  
print(sortIndex[-c(1:i-1)])

## [1] 139 227 245 119 67 132 134 5 164 219 107 118 66 94 204 120 181  
## [18] 249 13 237 174 205 114 55 154 96 161 232 27 45 1 180 220 36  
## [35] 247 125 131 102 138 105 18 143 34 238 95 10 243 60 88 175 250  
## [52] 241 183 123 142 136 33 239 101 86 230 206 189 163 50 203 97 224  
## [69] 172 246 32 16 150 187 2 167 168 11 72 79 157 48 211 213 151  
## [86] 178 42 111 182 159 6 63 61 158 190 176 8 137 226 59 58 173  
## [103] 127 214 28 43 68 109 80

Part c:

#Our vector xIndex contains the yVec indices corresponding to the elements in sortY that are greater than 600  
xIndex <- sortIndex[-c(1:i-1)]  
#We now print the elements in xVec at the indices stored in xIndex, as described above  
print(xVec[xIndex])

## [1] 176 678 179 444 724 189 457 513 743 5 10 789 38 760 446 986 894  
## [18] 238 640 110 203 533 113 358 977 294 137 258 577 55 708 996 863 627  
## [35] 123 515 359 964 324 24 364 260 618 957 48 107 631 266 680 478 178  
## [52] 34 900 537 160 274 437 285 505 19 188 190 467 852 803 517 69 399  
## [69] 768 545 408 676 407 972 437 353 371 390 995 652 148 458 501 124 216  
## [86] 880 836 878 357 660 44 197 578 293 324 49 646 543 256 511 525 339  
## [103] 263 14 257 278 61 840 956

Part d:

#We define the mean of xVec  
xBar <- mean(xVec)  
#We perform the desired operation, as per the prompt  
zVec <- sqrt(abs(xVec - xBar))  
print(zVec)

## [1] 16.0044994 3.8543482 15.8699716 17.7522956 7.8194629 20.1954450  
## [7] 15.7208142 13.9335566 20.2449006 18.5702989 7.8648585 13.5224258  
## [13] 13.7165593 19.3611983 13.2233127 14.9714395 19.5740645 9.3731532  
## [19] 19.4385185 16.8480266 12.8118695 16.0890025 16.0668603 19.7520632  
## [25] 11.9522383 14.0763632 11.1867779 13.9590831 11.3073427 9.1572922  
## [31] 9.6879306 6.6223863 3.8543482 12.8896858 15.1610026 13.2341981  
## [37] 18.1894475 15.7842960 8.8800901 2.4787093 9.4263461 19.5995918  
## [43] 13.1854465 18.9434949 19.9212449 15.7525871 22.4085698 2.4787093  
## [49] 16.1599505 18.7388367 23.3268943 17.6958752 13.6800585 12.3634947  
## [55] 9.6879306 5.1822775 16.2217138 8.5524266 7.6905136 13.6329014  
## [61] 11.2313846 14.2528594 15.9642100 11.5388041 17.9681941 20.3434510  
## [67] 16.4967876 19.7700784 17.7723381 22.1843188 7.4259006 23.3054500  
## [73] 14.4618118 19.4385185 22.6967839 17.4314658 14.3228489 22.4531512  
## [79] 14.1472259 22.4531512 9.5469367 20.8532012 10.6233705 4.1405314  
## [85] 9.5991666 20.8051917 21.2333700 15.1044364 9.2273506 13.8976257  
## [91] 15.4642814 15.3669776 19.3944322 17.5540309 20.0961688 12.5640758  
## [97] 19.5667064 18.8452647 11.8682770 14.7018366 7.2899931 22.6305988  
## [103] 13.4217734 21.0678903 20.6846803 20.2520122 21.0203711 12.7335777  
## [109] 19.7013705 9.9426355 20.6432556 19.4898948 16.0890025 18.4080417  
## [115] 19.2316406 11.3954377 18.9962101 18.3614814 2.8028557 23.1115556  
## [121] 13.1203658 20.8292103 9.2273506 10.1066315 7.9463199 2.8537694  
## [127] 13.7424889 20.2449006 19.3870060 13.9948562 9.6361818 16.2128344  
## [133] 18.8452647 2.2680388 18.7844617 13.3362663 9.5469367 11.3073427  
## [139] 16.6089133 5.0143793 9.4416100 17.0837935 13.8512093 16.6690132  
## [145] 20.0961688 6.0709143 15.9732276 13.1584194 8.8399095 6.6974622  
## [151] 15.3576040 15.0948998 7.5402918 22.9160206 19.3944322 3.0239048  
## [157] 17.4314658 12.6038089 14.4271965 20.3434510 17.7441821 15.0948998  
## [163] 20.0035997 17.0629423 15.2034207 9.6511139 9.9426355 8.9919964  
## [169] 20.3505282 0.3794733 18.9510950 17.7804387 10.6233705 15.7751704  
## [175] 5.1131204 20.0712730 20.7811453 20.6916408 5.3050919 23.3268943  
## [181] 21.0272205 9.7394045 21.1694119 12.2940636 14.6677878 18.3069386  
## [187] 22.8066657 2.2680388 3.8915293 11.3073427 21.8207241 18.5163711  
## [193] 9.3196566 23.1331796 10.9610219 13.1093860 18.4080417 15.8159413  
## [199] 22.6084940 6.8451443 19.7194320 13.0055373 8.0711833 2.4199174  
## [205] 9.0079964 16.1819653 13.6434600 13.2987217 20.3259440 4.1056059  
## [211] 7.0102782 14.7358067 18.1067943 20.9250090 21.6366356 11.9939985  
## [217] 19.1795725 8.4346903 21.1389688 20.2766861 20.2025741 18.2169152  
## [223] 15.6797959 7.2702132 20.5634627 13.9948562 15.0380850 19.8205953  
## [229] 6.7189285 16.2436449 18.0237621 13.9232180 8.7095350 16.7587589  
## [235] 18.1423262 20.4485696 18.4893483 22.4754088 12.9172753 8.3579902  
## [241] 20.4415264 6.9897067 13.3844686 15.9642100 16.5183534 9.6511139  
## [247] 18.1343872 17.5540309 14.6238162 16.5485951

Part e:

#We define the max of yVec  
maxY <- max(yVec)  
#We initialize the count variable k to 0  
k <- 0  
#We loop through all values in yVec, to see if they are within 200 of maxY  
for(j in 1:250){  
 if((maxY-yVec[j])<200){  
 #If yVec[j] is within 200 of maxY, we increase k by 1  
 k <- k+1  
 }  
}  
#When we come out of the loop, k will equal the number of values in yVec that are within 200 of maxY; k is printed below   
print(k)

## [1] 57

Part f:

#We initialize the count variable k to 0  
k <- 0  
#We loop through all values in xVec, to see if they are divisible by 2  
for(j in 1:250){  
 if(xVec[j]%%2 == 0){  
 #If xVec[j] is divisible by 2, we increase k by 1  
 k <- k+1  
 }  
}  
#When we come out of the loop, k will equal the number of values in xVec that are divisible by 2; k is printed below  
print(k)

## [1] 124

Part g:

#We reorder xVec as per the order of increasing values in yVec; recall that sortIndex represents the yVec indices corresponding to each sorted element in sortY; we now reorder xVec using those same indices  
sortX <- xVec[sortIndex]  
print(sortX)

## [1] 405 842 308 572 461 8 256 507 373 639 42 616 29 645 376 669 688  
## [18] 197 63 638 862 77 996 93 59 585 661 72 339 20 206 537 174 322  
## [35] 42 603 425 48 707 452 477 99 224 811 715 358 963 222 395 543 480  
## [52] 193 683 710 691 954 700 614 787 835 275 435 309 368 224 460 497 944  
## [69] 530 765 523 171 870 807 469 828 624 200 713 365 781 74 129 76 701  
## [86] 760 193 866 353 168 967 545 920 541 650 148 277 18 667 865 987 120  
## [103] 655 1 554 699 311 458 632 84 269 82 280 544 17 621 807 113 136  
## [120] 457 702 91 625 767 828 109 860 363 121 657 668 324 382 956 299 403  
## [137] 74 928 415 38 127 176 678 179 444 724 189 457 513 743 5 10 789  
## [154] 38 760 446 986 894 238 640 110 203 533 113 358 977 294 137 258 577  
## [171] 55 708 996 863 627 123 515 359 964 324 24 364 260 618 957 48 107  
## [188] 631 266 680 478 178 34 900 537 160 274 437 285 505 19 188 190 467  
## [205] 852 803 517 69 399 768 545 408 676 407 972 437 353 371 390 995 652  
## [222] 148 458 501 124 216 880 836 878 357 660 44 197 578 293 324 49 646  
## [239] 543 256 511 525 339 263 14 257 278 61 840 956

Part h:

#We define vector l to represent the indices requested in the prompt  
l <- seq(1,250,by=3)  
#We pull the elements in yVec at those desired indices  
yVal <- yVec[l]  
print(yVal)

## [1] 709 517 437 783 671 860 581 347 279 974 216 776 538 460 985 248 317  
## [18] 288 687 957 938 101 615 285 106 414 881 488 484 791 246 643 845 553  
## [35] 465 87 993 116 473 635 310 428 965 19 489 803 604 800 175 516 902  
## [52] 689 881 593 835 398 358 850 791 915 665 167 866 942 320 482 216 488  
## [69] 681 273 884 970 469 717 127 952 284 695 325 777 792 72 738 791

Question 8:

#We create the vector x to represent the numerators in our fractions, as per the prompt  
x <- seq(2,38,by=2)  
#We create the vector y <- x+1 to represent the denominators in our fractions, as per the prompt  
y <- x+1  
#The cumprod function will return a vector of values to be added as per the prompt, i.e. the function returns (2/3, 2/3\*4/5, ...); to achieve the desired sum, we take the sum of that vector and add 1 in front  
z <- 1 + sum(cumprod(x/y))  
print(z)

## [1] 6.976346