Homework Assignment

Computational Gene Finding

02-17-2023

Task A

```
# Accession number for SARS Coronavirus MA15 ExoN1
accession_number <- "FJ882953"

# Query
q <- query("q", paste("AC=", accession_number, sep = ""))

# Print the details
print(paste("Retrieved", q$nelem, "sequence(s) for Accession Number:",
    accession_number))
```

```
## [1] "Retrieved 1 sequence(s) for Accession Number: FJ882953"
```

```
print(paste("Length of the sequence:", getLength(q)))
```

```
## [1] "Length of the sequence: 29648"
```

```
seq_vector <- getSequence(q$req)[[1]]
seq_str <- DNAString(c2s(seq_vector))
print(seq_str)
```

```
## 29648-letter DNAString object
## seq: CTCGATCTCTTGTAGATCTGTTCTCTAAACGAACTT...GGAAGAGCCCTAATGTGTAAAATTAATTTTAGTAGT
```

Task B

```
# Reverse complement of the sequence
reverse_comp_seq <- reverseComplement(seq_str)
print(reverse_comp_seq)
```

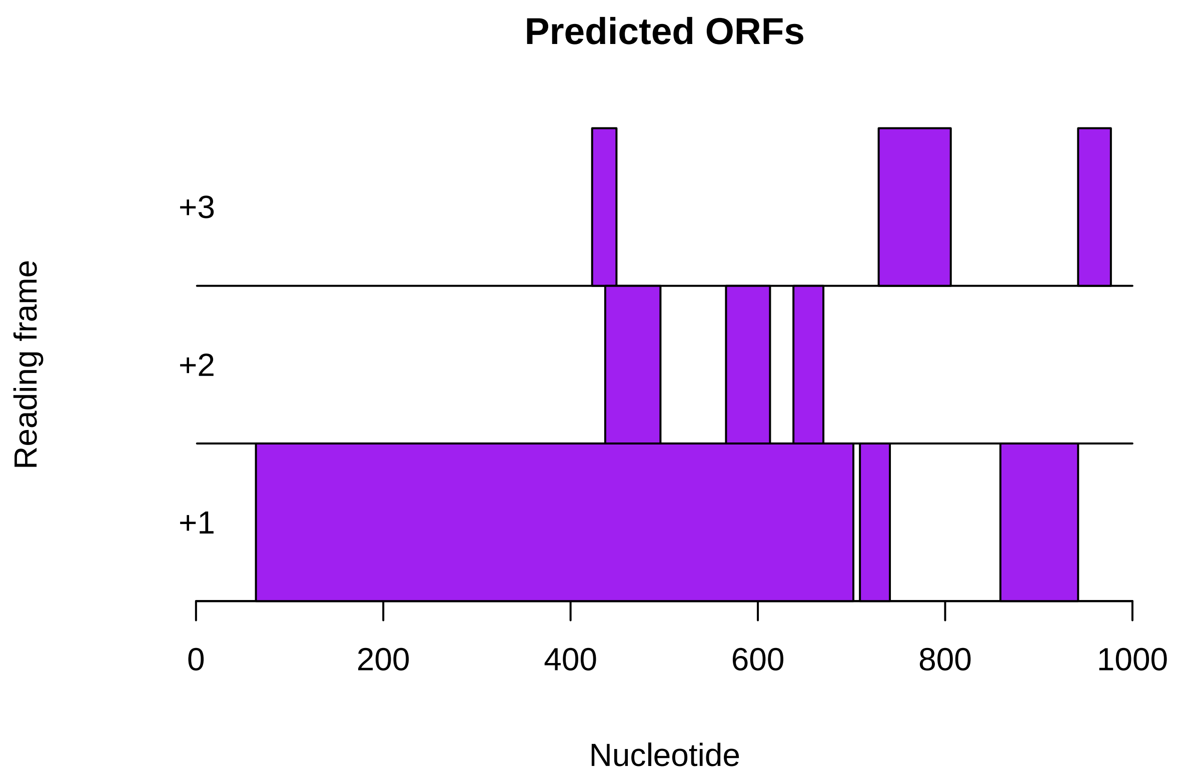
```
## 29648-letter DNAString object
## seq: ACTACTAAAATTAATTTTACACATTAGGGCTCTTCC...AAGTTCGTTTAGAGAACAGATCTACAAGAGATCGAG
```

```
# Print all the potential ORFs in the reverse complement of
# the sequence
orfs_rev_comp <- findORFs(as.character(reverse_comp_seq))
print(orfs_rev_comp[, c("start", "end", "length")])
```

```
##        start   end     length
##   [1,] "98"    "145"   "48"
##   [2,] "132"   "149"   "18"
##   [3,] "158"   "163"   "6"
##   [4,] "231"   "308"   "78"
##   [5,] "301"   "462"   "162"
##   [6,] "333"   "344"   "12"
##  [7,] "366"   "428"   "63" 
##  [8,] "478"   "516"   "39" 
##  [9,] "495"   "566"   "72" 
## [10,] "563"   "613"   "51" 
## [11,] "577"   "594"   "18" 
## [12,] "600"   "674"   "75" 
## [13,] "625"   "690"   "66" 
## [14,] "665"   "694"   "30" 
## [15,] "745"   "756"   "12" 
## [16,] "802"   "825"   "24" 
## [17,] "1003"  "1113"  "111"
## [18,] "1128"  "1199"  "72" 
## [19,] "1388"  "1432"  "45" 
## [20,] "1516"  "1521"  "6"  
## [21,] "1521"  "1589"  "69" 
## [22,] "1610"  "1651"  "42" 
## [23,] "1632"  "1742"  "111"
## [24,] "1706"  "1720"  "15" 
## [25,] "1822"  "1875"  "54" 
## [26,] "1836"  "1898"  "63" 
## [27,] "1862"  "1918"  "57" 
## [28,] "2065"  "2124"  "60" 
## [29,] "2152"  "2190"  "39" 
## [30,] "2196"  "2249"  "54" 
## [31,] "2279"  "2344"  "66" 
## [32,] "2375"  "2380"  "6"  
## [33,] "2390"  "2407"  "18" 
## [34,] "2413"  "2436"  "24" 
## [35,] "2424"  "2444"  "21" 
## [36,] "2561"  "2608"  "48" 
## [37,] "2605"  "2781"  "177"
## [38,] "2811"  "2852"  "42" 
## [39,] "2859"  "3011"  "153"
## [40,] "2959"  "2994"  "36" 
## [41,] "3150"  "3179"  "30" 
## [42,] "3288"  "3386"  "99" 
## [43,] "3298"  "3321"  "24" 
## [44,] "3473"  "3634"  "162"
## [45,] "3561"  "3572"  "12" 
## [46,] "3725"  "3787"  "63" 
## [47,] "3732"  "3737"  "6"  
## [48,] "3808"  "3837"  "30" 
## [49,] "3896"  "3955"  "60" 
## [50,] "3942"  "4049"  "108"
## [51,] "3964"  "4005"  "42" 
## [52,] "4059"  "4067"  "9"  
## [53,] "4067"  "4096"  "30" 
## [54,] "4141"  "4146"  "6"  
## [55,] "4151"  "4201"  "51" 
## [56,] "4266"  "4280"  "15" 
## [57,] "4330"  "4386"  "57" 
## [58,] "4430"  "4447"  "18" 
## [59,] "4437"  "4442"  "6"  
## [60,] "4508"  "4519"  "12" 
##  [61,] "4540"  "4650"  "111"
##  [62,] "4657"  "4713"  "57"
##  [63,] "4714"  "4776"  "63"
##  [64,] "4769"  "4783"  "15"
##  [65,] "4773"  "4811"  "39"
##  [66,] "4808"  "4852"  "45"
##  [67,] "4852"  "5004"  "153"
##  [68,] "4889"  "4903"  "15"
##  [69,] "4988"  "5026"  "39"
##  [70,] "5001"  "5063"  "63"
##  [71,] "5042"  "5056"  "15"
##  [72,] "5076"  "5141"  "66"
##  [73,] "5375"  "5395"  "21"
##  [74,] "5417"  "5437"  "21"
##  [75,] "5524"  "5568"  "45"
##  [76,] "5540"  "5554"  "15"
##  [77,] "5588"  "5617"  "30"
##  [78,] "5701"  "5901"  "201"
##  [79,] "6025"  "6132"  "108"
##  [80,] "6213"  "6407"  "195"
##  [81,] "6229"  "6261"  "33"
##  [82,] "6334"  "6357"  "24"
##  [83,] "6422"  "6445"  "24"
##  [84,] "6478"  "6552"  "75"
##  [85,] "6494"  "6502"  "9"
##  [86,] "6554"  "6601"  "48"
##  [87,] "6730"  "6747"  "18"
##  [88,] "6764"  "6781"  "18"
##  [89,] "6832"  "6885"  "54"
##  [90,] "6861"  "6902"  "42"
##  [91,] "6913"  "6969"  "57"
##  [92,] "7118"  "7159"  "42"
##  [93,] "7178"  "7201"  "24"
##  [94,] "7412"  "7417"  "6"
##  [95,] "7465"  "7473"  "9"
##  [96,] "7495"  "7596"  "102"
##  [97,] "7705"  "7935"  "231"
##  [98,] "7727"  "7747"  "21"
##  [99,] "7749"  "7865"  "117"
## [100,] "7814"  "7825"  "12"
## [101,] "7954"  "7977"  "24"
## [102,] "7974"  "8024"  "51"
## [103,] "8012"  "8056"  "45"
## [104,] "8090"  "8113"  "24"
## [105,] "8097"  "8180"  "84"
## [106,] "8194"  "8205"  "12"
## [107,] "8254"  "8286"  "33"
## [108,] "8359"  "8376"  "18"
## [109,] "8413"  "8418"  "6"
## [110,] "8424"  "8483"  "60"
## [111,] "8428"  "8436"  "9"
## [112,] "8486"  "8518"  "33"
## [113,] "8534"  "8551"  "18"
## [114,] "8541"  "8618"  "78"
## [115,] "8685"  "8768"  "84"
## [116,] "8905"  "8913"  "9"
## [117,] "9126"  "9155"  "30"
## [118,] "9148"  "9168"  "21"
## [119,] "9152"  "9178"  "27"
## [120,] "9275"  "9301"  "27"
## [121,] "9298"  "9306"  "9"
## [122,] "9390"  "9518"  "129"
## [123,] "9433"  "9450"  "18"
## [124,] "9472"  "9552"  "81"
## [125,] "9583"  "9606"  "24"
## [126,] "9638"  "9643"  "6"
## [127,] "9817"  "9876"  "60"
## [128,] "9852"  "10133" "282"
## [129,] "9949"  "10044" "96"
## [130,] "9974"  "9985"  "12"
## [131,] "10057" "10119" "63"
## [132,] "10151" "10201" "51"
## [133,] "10226" "10297" "72"
## [134,] "10257" "10283" "27"
## [135,] "10353" "10403" "51"
## [136,] "10412" "10588" "177"
## [137,] "10446" "10487" "42"
## [138,] "10599" "10880" "282"
## [139,] "10652" "10705" "54"
## [140,] "10723" "10731" "9"
## [141,] "10774" "10959" "186"
## [142,] "10914" "11057" "144"
## [143,] "11000" "11065" "66"
## [144,] "11138" "11143" "6"
## [145,] "11154" "11186" "33"
## [146,] "11260" "11349" "90"
## [147,] "11274" "11369" "96"
## [148,] "11433" "11606" "174"
## [149,] "11554" "11568" "15"
## [150,] "11572" "11664" "93"
## [151,] "11661" "11723" "63"
## [152,] "11674" "11778" "105"
## [153,] "11723" "11728" "6"
## [154,] "11797" "11895" "99"
## [155,] "11915" "11935" "21"
## [156,] "12065" "12082" "18"
## [157,] "12075" "12119" "45"
## [158,] "12214" "12312" "99"
## [159,] "12290" "12355" "66"
## [160,] "12412" "12591" "180"
## [161,] "12434" "12484" "51"
## [162,] "12557" "12688" "132"
## [163,] "12588" "12746" "159"
## [164,] "12610" "12627" "18"
## [165,] "12643" "12654" "12"
## [166,] "12700" "12714" "15"
## [167,] "12748" "12786" "39"
## [168,] "12836" "12880" "45"
## [169,] "12995" "13006" "12"
## [170,] "13022" "13066" "45"
## [171,] "13029" "13079" "51"
## [172,] "13076" "13087" "12"
## [173,] "13088" "13108" "21"
## [174,] "13159" "13164" "6"
## [175,] "13187" "13204" "18"
## [176,] "13232" "13267" "36"
## [177,] "13274" "13285" "12"
## [178,] "13285" "13311" "27"
## [179,] "13296" "13301" "6"
## [180,] "13409" "13420" "12"
## [181,] "13417" "13431" "15"
## [182,] "13422" "13448" "27"
## [183,] "13451" "13462" "12"
## [184,] "13533" "13580" "48"
## [185,] "13612" "13674" "63"
## [186,] "13641" "13646" "6"
## [187,] "13701" "13736" "36"
## [188,] "13814" "13858" "45"
## [189,] "13869" "13877" "9"
## [190,] "13936" "13953" "18"
## [191,] "13981" "14001" "21"
## [192,] "14053" "14076" "24"
## [193,] "14061" "14150" "90"
## [194,] "14230" "14253" "24"
## [195,] "14273" "14293" "21"
## [196,] "14314" "14361" "48"
## [197,] "14391" "14513" "123"
## [198,] "14431" "14463" "33"
## [199,] "14520" "14585" "66"
## [200,] "14601" "14765" "165"
## [201,] "14674" "14697" "24"
## [202,] "14710" "14748" "39"
## [203,] "14801" "14833" "33"
## [204,] "15134" "15139" "6"
## [205,] "15174" "15203" "30"
## [206,] "15231" "15236" "6"
## [207,] "15252" "15266" "15"
## [208,] "15276" "15290" "15"
## [209,] "15390" "15602" "213"
## [210,] "15437" "15457" "21"
## [211,] "15566" "15574" "9"
## [212,] "15586" "15606" "21"
## [213,] "15599" "15622" "24"
## [214,] "15698" "15733" "36"
## [215,] "15715" "15852" "138"
## [216,] "15761" "15790" "30"
## [217,] "15872" "15877" "6"
## [218,] "15893" "15958" "66"
## [219,] "15918" "15926" "9"
## [220,] "15978" "16121" "144"
## [221,] "15988" "16059" "72"
## [222,] "16303" "16398" "96"
## [223,] "16484" "16519" "36"
## [224,] "16491" "16646" "156"
## [225,] "16516" "16551" "36"
## [226,] "16654" "16662" "9"
## [227,] "16800" "16877" "78"
## [228,] "16942" "17082" "141"
## [229,] "17145" "17165" "21"
## [230,] "17215" "17226" "12"
## [231,] "17280" "17474" "195"
## [232,] "17302" "17310" "9"
## [233,] "17502" "17603" "102"
## [234,] "17632" "17649" "18"
## [235,] "17712" "17864" "153"
## [236,] "17842" "17856" "15"
## [237,] "17887" "17913" "27"
## [238,] "17938" "17967" "30"
## [239,] "17967" "18056" "90"
## [240,] "18153" "18197" "45"
## [241,] "18190" "18222" "33"
## [242,] "18237" "18263" "27"
## [243,] "18294" "18308" "15"
## [244,] "18348" "18392" "45"
## [245,] "18408" "18464" "57"
## [246,] "18496" "18504" "9"
## [247,] "18501" "18533" "33"
## [248,] "18589" "18606" "18"
## [249,] "18599" "18691" "93"
## [250,] "18622" "18639" "18"
## [251,] "18652" "18765" "114"
## [252,] "18749" "18844" "96"
## [253,] "18823" "18831" "9"
## [254,] "18957" "18968" "12"
## [255,] "18965" "19099" "135"
## [256,] "19084" "19254" "171"
## [257,] "19104" "19160" "57"
## [258,] "19211" "19273" "63"
## [259,] "19248" "19304" "57"
## [260,] "19270" "19281" "12"
## [261,] "19314" "19325" "12"
## [262,] "19336" "19416" "81"
## [263,] "19463" "19675" "213"
## [264,] "19527" "19544" "18"
## [265,] "19557" "19745" "189"
## [266,] "19681" "19695" "15"
## [267,] "19726" "19770" "45"
## [268,] "19978" "19983" "6"
## [269,] "19995" "20015" "21"
## [270,] "20030" "20131" "102"
## [271,] "20100" "20114" "15"
## [272,] "20128" "20145" "18"
## [273,] "20152" "20223" "72"
## [274,] "20264" "20356" "93"
## [275,] "20328" "20462" "135"
## [276,] "20462" "20500" "39"
## [277,] "20527" "20574" "48"
## [278,] "20534" "20677" "144"
## [279,] "20607" "20633" "27"
## [280,] "20698" "20718" "21"
## [281,] "20784" "20804" "21"
## [282,] "20808" "21056" "249"
## [283,] "20843" "20887" "45"
## [284,] "20959" "20979" "21"
## [285,] "20983" "21027" "45"
## [286,] "20999" "21103" "105"
## [287,] "21057" "21068" "12"
## [288,] "21072" "21089" "18"
## [289,] "21106" "21123" "18"
## [290,] "21110" "21160" "51"
## [291,] "21139" "21171" "33"
## [292,] "21231" "21341" "111"
## [293,] "21357" "21374" "18"
## [294,] "21398" "21505" "108"
## [295,] "21421" "21468" "48"
## [296,] "21444" "21464" "21"
## [297,] "21471" "21515" "45"
## [298,] "21515" "21520" "6"
## [299,] "21595" "21600" "6"
## [300,] "21870" "21956" "87"
## [301,] "21932" "22033" "102"
## [302,] "22030" "22044" "15"
## [303,] "22098" "22151" "54"
## [304,] "22102" "22134" "33"
## [305,] "22203" "22256" "54"
## [306,] "22263" "22298" "36"
## [307,] "22295" "22462" "168"
## [308,] "22314" "22319" "6"
## [309,] "22338" "22424" "87"
## [310,] "22501" "22659" "159"
## [311,] "22572" "22592" "21"
## [312,] "22623" "22709" "87"
## [313,] "22848" "22982" "135"
## [314,] "22969" "22974" "6"
## [315,] "23032" "23040" "9"
## [316,] "23059" "23067" "9"
## [317,] "23117" "23137" "21"
## [318,] "23172" "23198" "27"
## [319,] "23216" "23224" "9"
## [320,] "23254" "23262" "9"
## [321,] "23265" "23363" "99"
## [322,] "23419" "23484" "66"
## [323,] "23511" "23570" "60"
## [324,] "23531" "23539" "9"
## [325,] "23617" "23622" "6"
## [326,] "23698" "23703" "6"
## [327,] "23734" "23760" "27"
## [328,] "23862" "23876" "15"
## [329,] "23878" "23934" "57"
## [330,] "23978" "24028" "51"
## [331,] "24031" "24078" "48"
## [332,] "24133" "24156" "24"
## [333,] "24138" "24167" "30"
## [334,] "24230" "24286" "57"
## [335,] "24290" "24451" "162"
## [336,] "24351" "24395" "45"
## [337,] "24552" "24590" "39"
## [338,] "24584" "24613" "30"
## [339,] "24653" "24814" "162"
## [340,] "24700" "24732" "33"
## [341,] "24857" "24928" "72"
## [342,] "25006" "25026" "21"
## [343,] "25039" "25044" "6"
## [344,] "25060" "25089" "30"
## [345,] "25130" "25174" "45"
## [346,] "25251" "25262" "12"
## [347,] "25290" "25355" "66"
## [348,] "25352" "25393" "42"
## [349,] "25438" "25458" "21"
## [350,] "25638" "25643" "6"
## [351,] "25686" "25694" "9"
## [352,] "25700" "25738" "39"
## [353,] "25788" "25922" "135"
## [354,] "25926" "25943" "18"
## [355,] "26031" "26222" "192"
## [356,] "26047" "26076" "30"
## [357,] "26086" "26097" "12"
## [358,] "26141" "26212" "72"
## [359,] "26241" "26399" "159"
## [360,] "26288" "26395" "108"
## [361,] "26576" "26668" "93"
## [362,] "26668" "26712" "45"
## [363,] "26742" "26843" "102"
## [364,] "26803" "26817" "15"
## [365,] "26893" "26898" "6"
## [366,] "27019" "27045" "27"
## [367,] "27057" "27269" "213"
## [368,] "27182" "27187" "6"
## [369,] "27294" "27350" "57"
## [370,] "27354" "27437" "84"
## [371,] "27447" "27593" "147"
## [372,] "27625" "27633" "9"
## [373,] "27636" "27668" "33"
## [374,] "27672" "27899" "228"
## [375,] "27691" "27723" "33"
## [376,] "27918" "27938" "21"
## [377,] "27988" "27999" "12"
## [378,] "27996" "28022" "27"
## [379,] "28019" "28126" "108"
## [380,] "28035" "28160" "126"
## [381,] "28245" "28262" "18"
## [382,] "28277" "28345" "69"
## [383,] "28315" "28338" "24"
## [384,] "28354" "28359" "6"
## [385,] "28394" "28435" "42"
## [386,] "28432" "28470" "39"
## [387,] "28448" "28588" "141"
## [388,] "28501" "28536" "36"
## [389,] "28533" "28571" "39"
## [390,] "28572" "28679" "108"
## [391,] "28715" "28792" "78"
## [392,] "28758" "28763" "6"
## [393,] "28821" "28871" "51"
## [394,] "28928" "28990" "63"
## [395,] "28962" "28997" "36"
## [396,] "29010" "29018" "9"
## [397,] "29021" "29194" "174"
## [398,] "29071" "29211" "141"
## [399,] "29139" "29183" "45"
## [400,] "29211" "29222" "12"
## [401,] "29458" "29544" "87"
```

Task C

```
# Plot potential ORFs in the last 1000 bases
plotORFsinSeq(c2s(tail(seq_vector, 1000)))
```



Task D

```r
# Extract, translate and print the longest potential gene
potential_orfs <- findORFs(as.character(seq_str))
longest_gene <- potential_orfs[as.numeric(potential_orfs[, "length"]) ==
    max(as.numeric(potential_orfs[, "length"]))]
print(paste("The longest gene starts at index", longest_gene[1],
    "and ends at index", longest_gene[2]))
```

```
## [1] "The longest gene starts at index 227 and ends at index 13375"
```

```r
print(paste("Length of the longest gene:", longest_gene[3]))
```

```
## [1] "Length of the longest gene: 13149"
```

```r
print("The longest gene:")
```

```
## [1] "The longest gene:"
```

```r
print(DNAString(longest_gene[4]))
```

```
## 13149-letter DNAString object
## seq: ATGGAGAGCCTTGTTCTTGGTGTCAACGAGAAAACA...GATGCATCAACGTTTTTAAACGGGTTTGCGGTGTAA
```

```r
protein <- translate(DNAString(longest_gene[4]))
print(paste("The resulting protein sequence of length:", length(protein)))
```

```
## [1] "The resulting protein sequence of length: 4383"
```

```r
print(protein)
```

```
## 4383-letter AAString object
## seq: MESLVLGVNEKTHVQLSLPVLQVRDVLVRGFGDSVE...TVCGMWKGYGCSCDQLREPLMQSADASTFLNGFAV*
```

Task E and F

```r
# Identify the significant ORFs using the 95th percentile
# as the threshold value.
random_seqs <- generateSeqsWithMultinomialModel(c2s(seq_vector),
    30)
random_orfs <- lapply(random_seqs, findORFs)

length_vector <- sapply(random_orfs, get_max_seq_vector)

threshold <- quantile(length_vector, 0.95)
print(threshold)
```

```
##    95%
## 385.65
```

```r
significant_orfs <- potential_orfs[as.numeric(potential_orfs[,
    "length"]) > threshold, ]

print("Significant ORFs:")
```

```
## [1] "Significant ORFs:"
```

```r
print(significant_orfs[, c("start", "end", "length")])
```

```
##      start   end     length 
## [1,] "227"   "13375" "13149"
## [2,] "696"   "1187"  "492"  
## [3,] "13561" "21447" "7887" 
## [4,] "21454" "25221" "3768" 
## [5,] "25230" "26054" "825"  
## [6,] "25651" "26115" "465"  
## [7,] "26360" "27025" "666"  
## [8,] "28082" "29350" "1269" 
```

Not all of the ORFs found in a DNA sequence correspond to real genes. Some of them occur by chance. To extract the actual genes, lab experimentation is necessary along with bioinformatics. As computer scientists, we can make predictions to extract genes from a sequence. The length of the ORF can be used as a measure, as long ORFs cannot happen by chance. Small ORFs have a high probability of occurring by chance, so we can eliminate those using a threshold. We could have used the 100th percentile (or the largest value) of the ORF length as the threshold, but by reducing the threshold slightly (5% in this case), we improve our chances of finding the actual genes in the sequence.

Methods used in this assignment

```
findPotentialStartsAndStops <- function(sequence)
{
  # Define a vector with the sequences of potential start and stop codons
  codons <- c("atg", "taa", "tag", "tga")
  # Find the number of occurrences of each type of potential start or stop codon
  for (i in 1:4)
  {
    codon <- codons[i]
    # Find all occurrences of codon "codon" in sequence "sequence"
    occurrences <- matchPattern(codon, sequence)
    # Find the start positions of all occurrences of "codon" in sequence "sequence"
    codonpositions <- start(occurrences)
    # Find the total number of potential start and stop codons in sequence "sequence"
    numoccurrences <- length(codonpositions)
    if (i == 1)
    {
      # Make a copy of vector "codonpositions" called "positions"
      positions <- codonpositions
      # Make a vector "types" containing "numoccurrences" copies of "codon"
      types <- rep(codon, numoccurrences)
    }
    else
    {
      # Add the vector "codonpositions" to the end of vector "positions":
      positions <- append(positions, codonpositions, after=length(positions))
      # Add the vector "rep(codon, numoccurrences)" to the end of vector "types":
      types <- append(types, rep(codon, numoccurrences), after=length(types))
    }
  }
  # print(positions)
  # Sort the vectors "positions" and "types" in order of position along the input sequence:
  indices <- order((positions))
  positions <- positions[indices]
  types <- types[indices]
  # Return a list variable including vectors "positions" and "types":
  mylist <- list(positions,types)
  return(mylist)
}


findORFsinSeq <- function(sequence)
{
  require(Biostrings)
  # Make vectors "positions" and "types" containing information on the positions of ATGs in the sequenc
  mylist <- findPotentialStartsAndStops(sequence)
  positions <- mylist[[1]]
  types <- mylist[[2]]
  # Make vectors "orfstarts" and "orfstops" to store the predicted start and stop codons of ORFs
  orfstarts <- numeric()
  orfstops <- numeric()
  # Make a vector "orflengths" to store the lengths of the ORFs
  orflengths <- numeric()
  # Print out the positions of ORFs in the sequence:
  # Find the length of vector "positions"
  numpositions <- length(positions)
  # There must be at least one start codon and one stop codon to have an ORF.
  if (numpositions >= 2)
  {
    for (i in 1:(numpositions-1))
    {
      posi <- positions[i]
      typei <- types[i]
      found <- 0
      while (found == 0)
      {
        for (j in (i+1):numpositions)
        {
          posj <- positions[j]
          typej <- types[j]
          posdiff <- posj - posi
          posdiffmod3 <- posdiff %% 3
          # Add in the length of the stop codon
          orflength <- posj - posi + 3
          if (typei == "atg" && (typej == "taa" || typej == "tag" || typej == "tga") && posdiffmod3 == 
          {
            # Check if we have already used the stop codon at posj+2 in an ORF
            numorfs <- length(orfstops)
            usedstop <- -1
            if (numorfs > 0)
            {
              for (k in 1:numorfs)
              {
                orfstopk <- orfstops[k]
                if (orfstopk == (posj + 2)) { usedstop <- 1 }
              }
            }
            if (usedstop == -1)
```

```r
            {
              orfstarts <- append(orfstarts, posi, after=length(orfstarts))
              orfstops <- append(orfstops, posj+2, after=length(orfstops)) # Including the stop codon.
              orflengths <- append(orflengths, orflength, after=length(orflengths))
            }
            found <- 1
            break
          }
          if (j == numpositions) { found <- 1 }
        }
      }
    }
  }
  # Sort the final ORFs by start position:
  indices <- order(orfstarts)
  orfstarts <- orfstarts[indices]
  orfstops <- orfstops[indices]
  # Find the lengths of the ORFs that we have
  orflengths <- numeric()
  numorfs <- length(orfstarts)
  for (i in 1:numorfs)
  {
    orfstart <- orfstarts[i]
    orfstop <- orfstops[i]
    orflength <- orfstop - orfstart + 1
    orflengths <- append(orflengths,orflength,after=length(orflengths))
  }
  mylist <- list(orfstarts, orfstops, orflengths)
  return(mylist)
}


plotORFsinSeq <- function(sequence)
{
  # Make vectors "positions" and "types" containing information on the positions of ATGs in the sequenc
  mylist <- findPotentialStartsAndStops(sequence)
  positions <- mylist[[1]]
  types <- mylist[[2]]
  # Make vectors "orfstarts" and "orfstops" to store the predicted start and stop codons of ORFs
  orfstarts <- numeric()
  orfstops <- numeric()
  # Make a vector "orflengths" to store the lengths of the ORFs
  orflengths <- numeric()
  # Print out the positions of ORFs in the sequence:
  numpositions <- length(positions) # Find the length of vector "positions"
  # There must be at least one start codon and one stop codon to have an ORF.
  if (numpositions >= 2)
  {
    for (i in 1:(numpositions-1))
    {
      posi <- positions[i]
      typei <- types[i]
      found <- 0
      while (found == 0)
      {
        for (j in (i+1):numpositions)
        {
          posj <- positions[j]
          typej <- types[j]
          posdiff <- posj - posi
          posdiffmod3 <- posdiff %% 3
          orflength <- posj - posi + 3 # Add in the length of the stop codon
          if (typei == "atg" && (typej == "taa" || typej == "tag" || typej == "tga") && posdiffmod3 ==
          {
            # Check if we have already used the stop codon at posj+2 in an ORF
            numorfs <- length(orfstops)
            usedstop <- -1
            if (numorfs > 0)
            {
              for (k in 1:numorfs)
              {
                orfstopk <- orfstops[k]
                if (orfstopk == (posj + 2)) { usedstop <- 1 }
              }
            }
            if (usedstop == -1)
            {
              orfstarts <- append(orfstarts, posi, after=length(orfstarts))
              orfstops <- append(orfstops, posj+2, after=length(orfstops)) # Including the stop codon.
              orflengths <- append(orflengths, orflength, after=length(orflengths))
            }
            found <- 1
            break
          }
          if (j == numpositions) { found <- 1 }
        }
      }
    }
  }
  # Sort the final ORFs by start position:
  indices <- order(orfstarts)
  orfstarts <- orfstarts[indices]
  orfstops <- orfstops[indices]
  # Make a plot showing the positions of ORFs in the input sequence:
  # Draw a line at y=0 from 1 to the length of the sequence:
  x <- c(1,nchar(sequence))
  y <- c(0,0)
  plot(x, y, ylim=c(0,3), type="l", axes=FALSE, xlab="Nucleotide", ylab="Reading frame", main="Predicte
  segments(1,1,nchar(sequence),1)
  segments(1,2,nchar(sequence),2)
  # Add the x-axis at y=0:
  axis(1, pos=0)
  # Add the y-axis labels:
  text(0.9,0.5,"+1")
  text(0.9,1.5,"+2")
  text(0.9,2.5,"+3")
```

```
  # Make a plot of the ORFs in the sequence:
  numorfs <- length(orfstarts)
  for (i in 1:numorfs)
  {
    orfstart <- orfstarts[i]
    orfstop <- orfstops[i]
    remainder <- (orfstart-1) %% 3
    if (remainder == 0) # +1 reading frame
    {
      rect(orfstart,0,orfstop,1,col="purple",border="black")
    }
    else if (remainder == 1)
    {
      rect(orfstart,1,orfstop,2,col="purple",border="black")
    }
    else if (remainder == 2)
    {
      rect(orfstart,2,orfstop,3,col="purple",border="black")
    }
  }
}

generateSeqsWithMultinomialModel <- function(inputsequence, X)
{
  # Change the input sequence into a vector of letters
  require("seqinr") # This function requires the SeqinR package.
  inputsequencevector <- s2c(inputsequence)
  # Find the frequencies of the letters in the input sequence "inputsequencevector":
  mylength <- length(inputsequencevector)
  mytable <- table(inputsequencevector)
  # Find the names of the letters in the sequence
  letters <- rownames(mytable)
  numletters <- length(letters)
  probabilities <- numeric() # Make a vector to store the probabilities of letters
  for (i in 1:numletters)
  {
    letter <- letters[i]
    count <- mytable[[i]]
    probabilities[i] <- count/mylength
  }
  # Make X random sequences using the multinomial model with probabilities "probabilities"
  seqs <- numeric(X)
  for (j in 1:X)
  {
    seq <- sample(letters, mylength, rep=TRUE, prob=probabilities) # Sample with replacement
    seq <- c2s(seq)
    seqs[j] <- seq
  }
  # Return the vector of random sequences
  return (as.list(seqs))
}

get_max_seq_vector <- function (x) {
```

```r
  return (max(as.numeric(x[,"length"])))
}
```