C S 509 HW4

Author: Indronil Bhattacharjee

Submitted on: September 26, 2023

==========

::: {.cell _cell_guid='b1076dfc-b9ad-4769-8c92-a6c4dae69d19' _uuid='8f2839f25d086af736a60e9eeb907d3b93b6
execution='{"iopub.execute_input":"2023-09-26T05:36:09.564310Z","iopub.status.busy":"2023-09-26T05:36:09.529504Z","iopub.status.idle":"2023-09-26T05:37:26.040367Z"}' trusted='true' vscode='{"languageId":"r"}'}

```r
# Load required libraries
library(dplyr)
library(ggplot2)

# Load the GFF3 file (replace 'path_to_gff3_file' with the actual file path)
gff3_file <- read.table("/kaggle/input/gencode-human-genome-annotation/gencode.v44.primary
```

:::

Task 1: Linear interval search

```r
# Linear search to count overlapping features
count.features.linear <- function(chr, x, y, GFF) {
  count <- 0
  for (i in 1:nrow(GFF)) {
    if (GFF$chromosome[i] == chr && GFF$end[i] >= x && GFF$start[i] <= y) {
      count <- count + 1
    }
  }
  return(count)
}
```

Task 2: Vectorized interval search

```
# Vectorized search to count overlapping features
count.features.vectorized <- function(chr, x, y, GFF) {
  count <- sum(GFF$chromosome == chr & GFF$end >= x & GFF$start <= y)
  return(count)
}
```

Task 3: Binary interval search

```
count.features.binary <- function(chr, x, y, sorted.coordinates) {
  count <- 0
  left <- 1
  right <- length(sorted.coordinates)

  while (left <= right) {
    mid <- left + floor((right - left) / 2)
    if (sorted.coordinates[mid] >= x && sorted.coordinates[mid] <= y) {
      count <- count + 1
    }
    if (sorted.coordinates[mid] < x) {
      left <- mid + 1
    } else {
      right <- mid - 1
    }
  }
  return(count)
}
```

Task 4: Reporting the runtime

```
# Example usage and runtime reporting
chr <- "chr1"
x <- 10000
y <- 12000
# Extract relevant data for the chromosome
# Assuming the GFF3 format columns: chromosome, start, end
GFF <- gff3_file %>%
  filter(V1 == chr) %>%
```

```
  select(chromosome = V1, start = V4, end = V5) %>%
  mutate(start = as.numeric(start), end = as.numeric(end))  # Convert start and end to num

# Sort the coordinates for binary search
sorted.coordinates <- sort(c(GFF$start, GFF$end))

# Runtime for linear search
linear_time <- system.time(count.features.linear(chr, x, y, GFF))[3]
linear_result <- count.features.linear(chr, x, y, GFF)
cat("Linear Search:", linear_result, "times\n")

# Runtime for vectorized search
vectorized_time <- system.time(count.features.vectorized(chr, x, y, GFF))[3]
vectorized_result <- count.features.vectorized(chr, x, y, GFF)
cat("Vectorized Search:", vectorized_result, "times\n")

# Runtime for binary search
binary_time <- system.time(count.features.binary(chr, x, y, sorted.coordinates))[3]
binary_result <- count.features.binary(chr, x, y, sorted.coordinates)
cat("Binary Search:", binary_result, "times\n")

# Report runtimes
cat("Linear Search Runtime:", linear_time, "seconds\n")
cat("Vectorized Search Runtime:", vectorized_time, "seconds\n")
cat("Binary Search Runtime:", binary_time, "seconds\n")
```

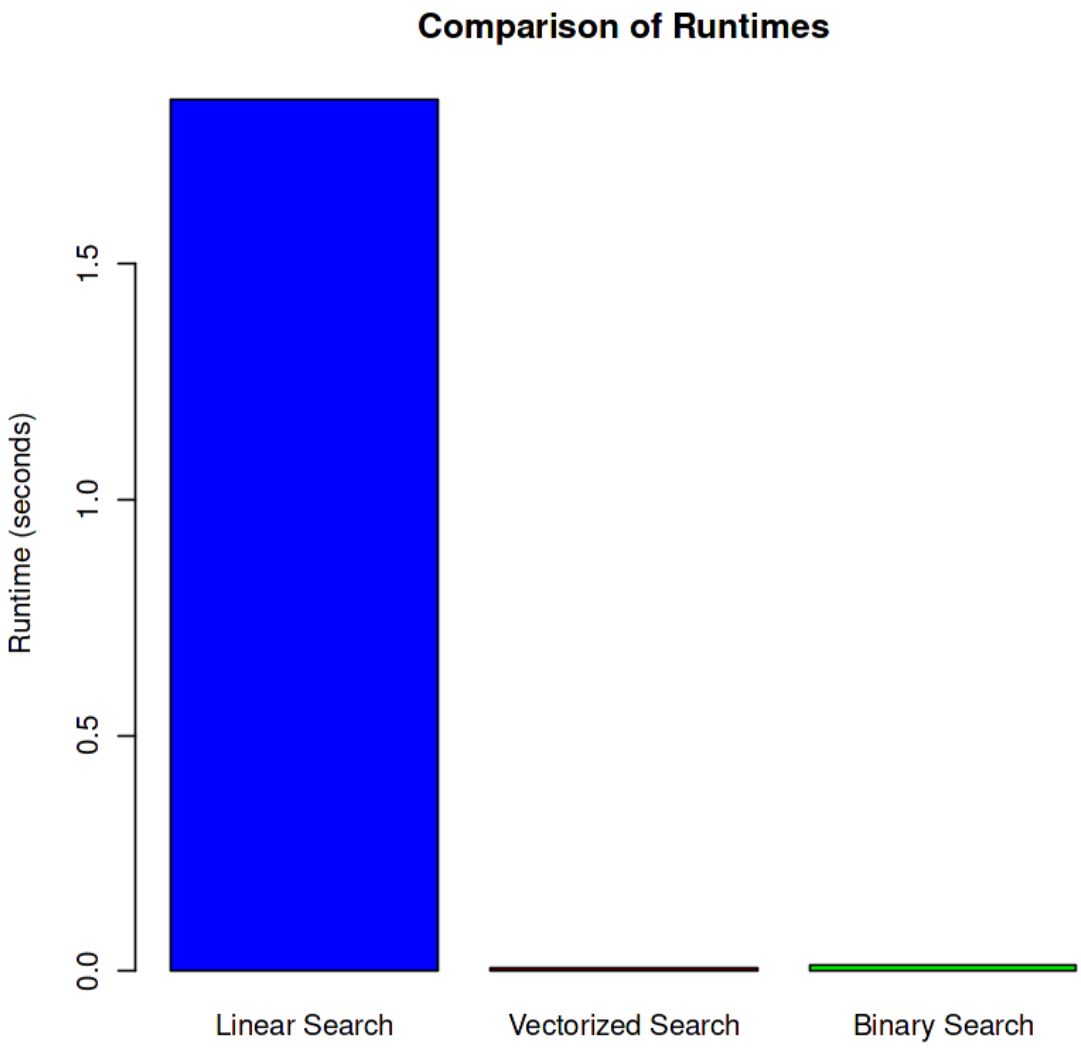
```
Linear Search: 3 times
Vectorized Search: 3 times
Binary Search: 3 times
Linear Search Runtime: 1.849 seconds
Vectorized Search Runtime: 0.006 seconds
Binary Search Runtime: 0.013 seconds
```

4.1 Visualization of runtimes

```
# Create a vector of runtimes
runtimes <- c(linear_time, vectorized_time, binary_time)

# Create a bar chart
barplot(runtimes,
        names.arg = c("Linear Search", "Vectorized Search", "Binary Search"),
```

```
    col = c("blue", "red", "green"),  # Colors for the bars
    ylab = "Runtime (seconds)",  # Label for the y-axis
    main = "Comparison of Runtimes",  # Title of the plot
    ylim = c(0, max(runtimes) + 0.01))  # Adjust the y-axis limits
```



Extra Tasks

Task E1: Query interval spanning only known coordinates

```
annotation_intervals <- list()
for (i in 1:nrow(GFF)){
    pair <- c(GFF$start[i], GFF$end[i])
    annotation_intervals <- c(annotation_intervals, list(pair))
}
```

```
# Initialize a list of pairs to store intervals like dictionary
interval_map <- list()

# Populate the hashmap with interval coordinates
for (idx in seq_along(annotation_intervals)) {
  interval <- annotation_intervals[[idx]]
  start <- interval[1]
  end <- interval[2]
  for (coord in seq(start, end)) {
    interval_map[[as.character(coord)]] <- idx
  }
}
```

```
# Function to find overlapping intervals for [x, y]
find_overlapping_intervals <- function(x, y, interval_map) {
  count <- 0
  for (coord in seq(x, y)) {
    if (as.character(coord) %in% names(interval_map)) {
      count <- count + 1
    }
  }
  return(count)
}

# Query interval
x <- 5788
y <- 5824

# Find overlapping intervals
```

```r
count <- find_overlapping_intervals(x, y, interval_map)
runtime_imp <- system.time(find_overlapping_intervals(x, y, interval_map))[3]

# Print the result
cat(count, "times\n")
cat("Improved runtime: ", runtime_imp, "seconds")
```

```
37 times
Improved runtime:  0.002 seconds
```

Task E2: Overlapping intervals for every interval

E2.1 Function Definition

```
from intervaltree import IntervalTree

# Find overlapping intervals for each intervals
def findOverlaps(interval_tree, input_intervals, overlapping_intervals):

    for idx, (start, end) in enumerate(input_intervals):
        overlaps = interval_tree[start:end]
        overlapping_intervals[idx] = [overlap.data for overlap in overlaps if overlap.data
    return overlapping_intervals
```

E2.2 Read Dataset

```
gff_file=open('/kaggle/input/gencode-human-genome-annotation/gencode.v44.primary_assembly.
```

```
# Initialize an empty list to store intervals
intervals = []

# with open('path_to_gff3_file.gff', 'r') as gff_file:
for line in gff_file:
    # Skip comment lines starting with '#'
    if not line.startswith('#'):
        fields = line.strip().split('\t')
        if len(fields) >= 5:
            if fields[0]=="chrY" and fields[2]=="exon":
                # Extract start and end positions from columns 4 and 5 (0-based index)
                start = int(fields[3])
                end = int(fields[4])
```

```
                intervals.append((start, end))
```

```
# Print the extracted intervals
print(len(intervals))
```

```
5301
```

E2.3 Apply function with Interval Tree

```
import time

# Create an IntervalTree and populate it with the genomic regions
interval_tree = IntervalTree()
for idx, (start, end) in enumerate(intervals):
    interval_tree[start:end] = idx

# Initialize a dictionary to store the overlapping intervals for each intervals
overlapping_intervals = {}
for idx in range(len(intervals)):
    overlapping_intervals[idx] = []

start_time = time.time()
overlappings = findOverlaps(interval_tree, intervals, overlapping_intervals)
end_time = time.time()
runtime_nlogn = end_time - start_time

# Print the result
for idx, overlaps in overlappings.items():
    print(f"Interval {idx}: {overlaps}")

    if idx>50:  # Printing only 50 since the output is too large
        break

print("Runtime O(n logn):", "{:.5f}".format(runtime_nlogn), "seconds")
```

```
Interval 0: []
Interval 1: []
Interval 2: [15, 10]
Interval 3: [21, 25, 28]
Interval 4: [16, 22, 36, 39, 44, 26, 11, 29, 48]
```

```
Interval 5: [37, 12, 23, 40, 45, 49, 17, 27, 30]
Interval 6: [50, 13, 18, 31, 41, 46]
Interval 7: [14, 42, 47, 19, 32]
Interval 8: [33]
Interval 9: [34]
Interval 10: [15, 2]
Interval 11: [16, 22, 36, 39, 44, 4, 26, 29, 48]
Interval 12: [37, 23, 40, 45, 49, 17, 5, 27, 30]
Interval 13: [50, 18, 31, 6, 41, 46]
Interval 14: [42, 47, 19, 32, 7]
Interval 15: [2, 10]
Interval 16: [22, 36, 39, 44, 4, 26, 11, 29, 48]
Interval 17: [37, 12, 23, 40, 45, 49, 5, 27, 30]
Interval 18: [50, 13, 31, 6, 41, 46]
Interval 19: [14, 42, 47, 32, 7]
Interval 20: []
Interval 21: [3, 25, 28]
Interval 22: [16, 36, 39, 44, 4, 26, 11, 29, 48]
Interval 23: [37, 12, 40, 45, 49, 17, 5, 27, 30]
Interval 24: []
Interval 25: [3, 21, 28]
Interval 26: [16, 22, 36, 39, 44, 4, 11, 29, 48]
Interval 27: [37, 12, 23, 40, 45, 49, 17, 5, 30]
Interval 28: [3, 21, 25]
Interval 29: [16, 22, 36, 39, 44, 4, 26, 11, 48]
Interval 30: [37, 12, 23, 40, 45, 49, 17, 5, 27]
Interval 31: [50, 13, 18, 6, 41, 46]
Interval 32: [14, 42, 47, 19, 7]
Interval 33: [8]
Interval 34: [9]
Interval 35: []
Interval 36: [16, 22, 39, 44, 4, 26, 11, 29, 48]
Interval 37: [12, 23, 40, 45, 49, 17, 5, 27, 30]
Interval 38: [43]
Interval 39: [16, 22, 36, 44, 4, 26, 11, 29, 48]
Interval 40: [37, 12, 23, 45, 49, 17, 5, 27, 30]
Interval 41: [50, 13, 18, 31, 6, 46]
Interval 42: [14, 47, 19, 32, 7]
Interval 43: [38]
Interval 44: [16, 22, 36, 39, 4, 26, 11, 29, 48]
Interval 45: [37, 12, 23, 40, 49, 17, 5, 27, 30]
Interval 46: [50, 13, 18, 31, 6, 41]
Interval 47: [14, 42, 19, 32, 7]
```

```
Interval 48: [36, 39, 44, 4, 26, 11, 29, 16, 22]
Interval 49: [37, 12, 23, 40, 45, 17, 5, 27, 30]
Interval 50: [13, 18, 31, 6, 41, 46]
Interval 51: [61]
Runtime O(n logn): 0.34230 seconds
```

E2.4 Apply function with quadratic time

```
def findOverlaps_n_sq(intervals):
    n = len(intervals)
    overlaps = [[] for _ in range(n)]  # Initialize an empty list for each interval

    for i in range(n):
        for j in range(n):
            if i != j:
                x1, y1 = intervals[i]
                x2, y2 = intervals[j]
                if x1 <= y2 and x2 <= y1:
                    overlaps[i].append(j)

    return overlaps

# Find overlapping intervals for each interval
start_time = time.time()
overlapping_intervals = findOverlaps_n_sq(intervals)
end_time = time.time()
runtime_n_sq = end_time - start_time


# Print the result
for i, overlaps in enumerate(overlapping_intervals):
    print(f"Interval {i}: {overlaps}")

    if i>50: # Printing only 50 since the output is too large
        break
print("Runtime O(n^2):", "{:.5f}".format(runtime_n_sq), "seconds")
```

```
Interval 0: []
Interval 1: []
Interval 2: [10, 15]
Interval 3: [21, 25, 28]
Interval 4: [11, 16, 22, 26, 29, 36, 39, 44, 48]
```

```
Interval 5: [12, 17, 23, 27, 30, 37, 40, 45, 49]
Interval 6: [13, 18, 31, 41, 46, 50]
Interval 7: [14, 19, 32, 42, 47]
Interval 8: [33]
Interval 9: [34]
Interval 10: [2, 15]
Interval 11: [4, 16, 22, 26, 29, 36, 39, 44, 48]
Interval 12: [5, 17, 23, 27, 30, 37, 40, 45, 49]
Interval 13: [6, 18, 31, 41, 46, 50]
Interval 14: [7, 19, 32, 42, 47]
Interval 15: [2, 10]
Interval 16: [4, 11, 22, 26, 29, 36, 39, 44, 48]
Interval 17: [5, 12, 23, 27, 30, 37, 40, 45, 49]
Interval 18: [6, 13, 31, 41, 46, 50]
Interval 19: [7, 14, 32, 42, 47]
Interval 20: []
Interval 21: [3, 25, 28]
Interval 22: [4, 11, 16, 26, 29, 36, 39, 44, 48]
Interval 23: [5, 12, 17, 27, 30, 37, 40, 45, 49]
Interval 24: []
Interval 25: [3, 21, 28]
Interval 26: [4, 11, 16, 22, 29, 36, 39, 44, 48]
Interval 27: [5, 12, 17, 23, 30, 37, 40, 45, 49]
Interval 28: [3, 21, 25]
Interval 29: [4, 11, 16, 22, 26, 36, 39, 44, 48]
Interval 30: [5, 12, 17, 23, 27, 37, 40, 45, 49]
Interval 31: [6, 13, 18, 41, 46, 50]
Interval 32: [7, 14, 19, 42, 47]
Interval 33: [8]
Interval 34: [9]
Interval 35: []
Interval 36: [4, 11, 16, 22, 26, 29, 39, 44, 48]
Interval 37: [5, 12, 17, 23, 27, 30, 40, 45, 49]
Interval 38: [43]
Interval 39: [4, 11, 16, 22, 26, 29, 36, 44, 48]
Interval 40: [5, 12, 17, 23, 27, 30, 37, 45, 49]
Interval 41: [6, 13, 18, 31, 46, 50]
Interval 42: [7, 14, 19, 32, 47]
Interval 43: [38]
Interval 44: [4, 11, 16, 22, 26, 29, 36, 39, 48]
Interval 45: [5, 12, 17, 23, 27, 30, 37, 40, 49]
Interval 46: [6, 13, 18, 31, 41, 50]
Interval 47: [7, 14, 19, 32, 42]
```

```
Interval 48: [4, 11, 16, 22, 26, 29, 36, 39, 44]
Interval 49: [5, 12, 17, 23, 27, 30, 37, 40, 45]
Interval 50: [6, 13, 18, 31, 41, 46]
Interval 51: [61]
Runtime O(n^2): 4.81346 seconds
```

E2.5 Runtime comparison

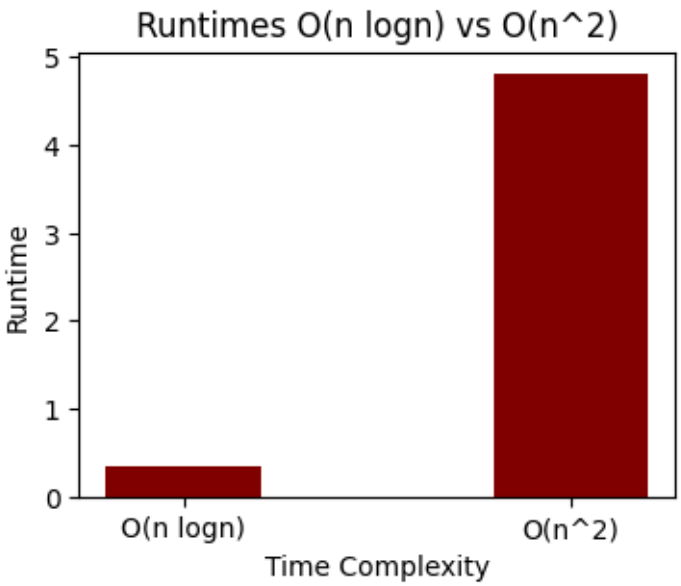
```
import numpy as np
import matplotlib.pyplot as plt


data = {'O(n logn)':runtime_nlogn, 'O(n^2)':runtime_n_sq}
courses = list(data.keys())
values = list(data.values())

fig = plt.figure(figsize = (4, 3))

# creating the bar plot
plt.bar(courses, values, color ='maroon',width = 0.4)

plt.xlabel("Time Complexity")
plt.ylabel("Runtime")
plt.title("Runtimes O(n logn) vs O(n^2)")
plt.show()
```



Task E3: Biological application

E3.1 Biological Motivation:

In genomics, it is essential to identify overlapping genomic regions, such as exons or binding sites of transcription factors, to understand their functional relationships and potential regulatory mechanisms. Overlapping regions can indicate shared regulatory elements, protein binding sites, or regions of interest.

E3.2 Dataset:

I have considered GENCODE Human Genome annotation dataset containing genomic coordinates of annotations from different genes. Exons are coding regions in genes, and we want to identify exons that overlap with each other.

E3.3 Interpretation:

The function, findOverlaps, takes the interval_tree and an empty overlapping_intervals dictionary as input. It loops through the intervals in the intervals list, retrieves overlapping intervals from the interval_tree, and stores their corresponding indices in the overlapping_intervals dictionary. Overlaps with the same index as the current interval are excluded.

Here, an IntervalTree data structure is used. Searching in an IntervalTree take O(log n) time and for n intervals, the total time complexity for the function stands out as O(n logn) which is way more efficient that quardatric time.

For the exons of Chromosome Y, here we have found runtime=0.34230 seconds for the function with an interval tree. On the other hand, runtime=4.81346 seconds for the function with quadratic time complexity.