Homework 4

ECE 590

Zhe Fan || zf70@duke.edu

October 30, 2022

Question 1:

Code and O anaysis:

```
def rmdup(data):              # How many times? | How long?
  ans ={}                     # O(1)            | O(1)
  data = data[::-1]           # O(1)            | O(N)
  ans = ans.fromkeys(data)    # O(1)            | O(N)
  ans = list(ans)             # O(1)            | O(N)
  ans = ans[::-1]             # O(1)            | O(N)
  return ans                  # O(1)            | O(1)
```

Runtime is $O(3N + 3) = O(N)$

Table 1: Data size and runtime for different types of random data.

Data Size	Many duplicates (ms)	Moderate duplicates (ms)	Rare duplicates (ms)
4K	0.0933	0.0825	0.166
8K	0.155	0.275	0.333
16K	0.306	0.538	0.748
32K	0.572	1.350	1.866
64K	1.088	2.395	3.692
128K	2.454	5.096	7.536
256K	4.955	11.198	16.746
512K	10.064	25.581	50.017
1M	46.561	52.859	117.419
2M	91.762	135.416	279.038
4M	202.012	422.907	649.166

In Figure.1, runtime of different data size for 3 types of random data (Many duplicates, Moderate duplicates, Rare duplicates) has been recorded. Y axis is set with unit ms and X axis with data size.

For many duplicates, elements range from 0 to size/2048, thus there would be at most size/2048 different elements.

Many duplicates' runtime is $O(1+size+size+size/2048+size/2048+1) = O((2+2/2048)size+2) = O(n)$

For mordate duplicates, runtime is $O((2+2/16)size+2) = O(n)$

For rare duplicates, elements range from 0 to 4*size, thus there would be at most size different elements.

For rare duplicates, runtime is $O(4size+2) = O(n)$

Although they are all the same O, but their runtime line will be different in 1.

For example, size is 4M, then Many duplicates' elements will range from 0 to 4M/2K = 2K, which means it will at most has 2K different elements, so after my code(line 4), it will only put 2K elements in list and revese it.

But for rare dulicates, whose elements range from 0 to 4M*4 = 16M, which means it will at most has 4M different elements, so after my code(line 4), it will put 4M elements in list and revese it.

2K vs 4M lead to the difference between two lines.

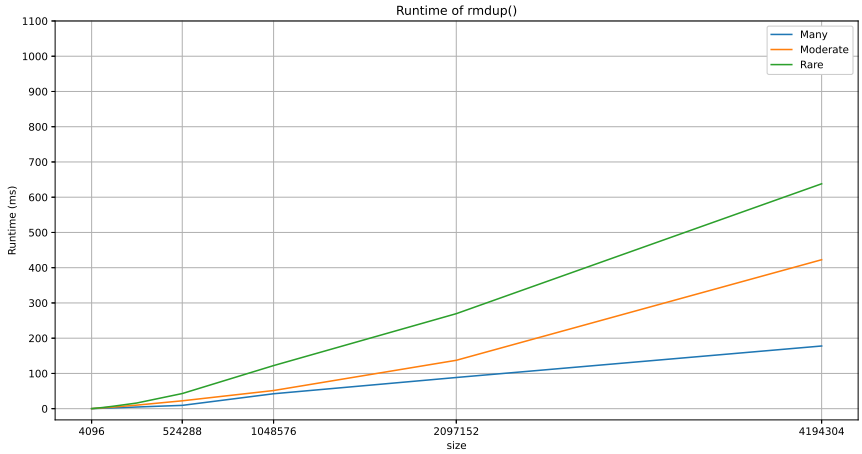

Figure 1: Runtime of different data size for 3 types of random data.

Question 2:

Code and O anaysis: Assume a is NxM, b is MxP

```
def matrix_mul(a, b):                  #How many times? | How long?
    ans = []                                    # O(1)     | O(1)
    for i in range(0, len(a)):                  # O(N)     | O(1)
      ans.append([])                            # O(N)     | O(1)
      for j in range(0, len(b[0])):             # O(N*P)   | O(1)
        mutiplyEle = 0                          # O(N*P)   | O(1)
          for k in range(0, len(a[0])):         # O(N*P*M) | O(1)
            mutiplyEle += a[i][k]*b[k][j]       # O(N*P*M) | O(1)
          ans[i].append(mutiplyEle)             # O(N*P)   | O(1)
    return ans                                  # O(1)     | O(1)

```

Runtime is $O(2N * P * M + 3N * P + 2N + 2) = O(N * P * M)$

Table 2: Data size and runtime for different types of random data.

N	Many rows by few columns (s)	Square(s)	Few rows by many columns(s)
4	1.1541e-05	1.00419e-05	8.542e-06
8	5.0708e-05	5.212e-05	4.999e-05
16	0.000321	0.000339	0.000338
32	0.00232	0.00247	0.00247
64	0.017	0.0176	0.0182
128	0.131	0.1401	0.139
256	1.0457	1.108	1.1015
512	8.854	9.411	9.657

In Figure.2, runtime with different N size for 3 types of random data (Many rows by few columns, Square, Few rows by many columns) has been recorded. Y axis is set with unit second and X axis with N size.

For 1st type: $O(N * P * M) = O(4 * N * N * N/4) = O(N^3)$

For 2nt type: $O(N * P * M) = O(N * N * N) = O(N^3)$

For 3nd type: $O(N * P * M) = O(N/4 * N * N * 4) = O(N^3)$

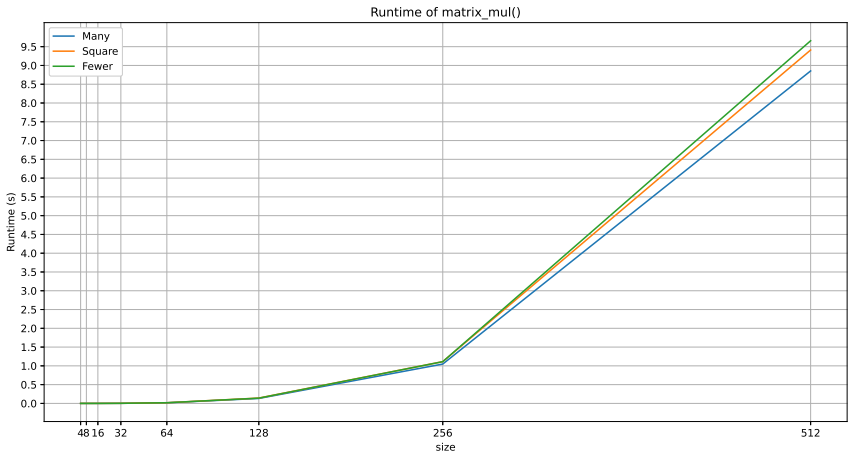

Figure 2: Runtime of different data size for 3 types of random data.

Question 3:

Code and O anaysis:

```
def matching_length_sub_strs(s,c1,c2):#How many times?|How long?
    c1Tuple = []                                    # O(1)   | O(1)
    c2Tuple = []                                    # O(1)   | O(1)
    length1 = 0                                     # O(1)   | O(1)
    index1 = 0                                      # O(1)   | O(1)
    length2 = 0                                     # O(1)   | O(1)
    index2 = 0                                      # O(1)   | O(1)
    # make 2 set of 2-tuple (index, length)
    for i in range(0, len(s)):                      # O(N)   | O(1)
      if s[i] != c1 or i == len(s)-1:               # O(N)   | O(1)
        if length1 > 0:                             # O(N/2) | O(1)
          c1Tuple.append((index1, length1))         # O(N/2) | O(1)
          length1 = 0                               # O(N/2) | O(1)
      else:                                         # O(N/2) | O(1)
        length1 = length1 +1                        # O(N/2) | O(1)
        if length1 == 1:                            # O(N/2) | O(1)
          index1 = i                                # O(N/2) | O(1)
      if s[i] != c2 or i == len(s)-1:               # O(N)   | O(1)
        if length2 > 0:                             # O(N/2) | O(1)
          c2Tuple.append((index2, length2))         # O(N/2) | O(1)
          length2 = 0                               # O(N/2) | O(1)
      else:                                         # O(N/2) | O(1)
        length2 = length2 + 1                       # O(N/2) | O(1)
        if length2 == 1:                            # O(N/2) | O(1)
          index2 = i                                # O(N/2) | O(1)
    # append a set of 3-tuple from the 2 set
    ans = set()                                     # O(1)   | O(1)
    if len(c1Tuple) == 0 or len(c2Tuple) == 0:      # O(1)   | O(1)
      return ans                                    # O(1)   | O(1)
    for i in range(0, len(c1Tuple)):                # O(N/2) | O(1)
      for j in range(0, len(c2Tuple)):              # O(N^2/4) | O(1)
        if c1Tuple[i][1] == c2Tuple[j][1]:          # O(N^2/4) | O(1)
          ans.add((c1Tuple[i][0],                   # O(N^2/4) | O(1)
                   c2Tuple[j][0], c1Tuple[i][1]))
    return ans                                      # O(1)   | O(1)

```

Runtime is $O(5N^2/4 + 10N + 8) = O(N^2)$
Best regular expression: $a^*|b^*$
Worst regular expression: $(ab)^*\,(aba)^*$

Table 3: Data size and runtime for different types of random data.

N	Best call (s)	Worst call(s)	Rndstr(s)
512	0.000145	0.0223	0.00314
1K	0.000289	0.09097	0.01109
2K	0.000616	0.438	0.0423
4K	0.001207	1.605	0.173
8K	0.00211	6.646	0.715
16K	0.00419	28.275	3.208

In Figure.3, runtime with different N size for 3 types of data (Best call, Worst call and random string) has been recorded. Y axis is set with unit second and X axis with N size.

For best case: O of runtime is $O(6+N+N+3+N+N+2+N+N+3) = O(6N+11) = O(N)$;

For worst case: O of runtime is $O(5N^2/4 + 10N + 8) = O(N^2)$

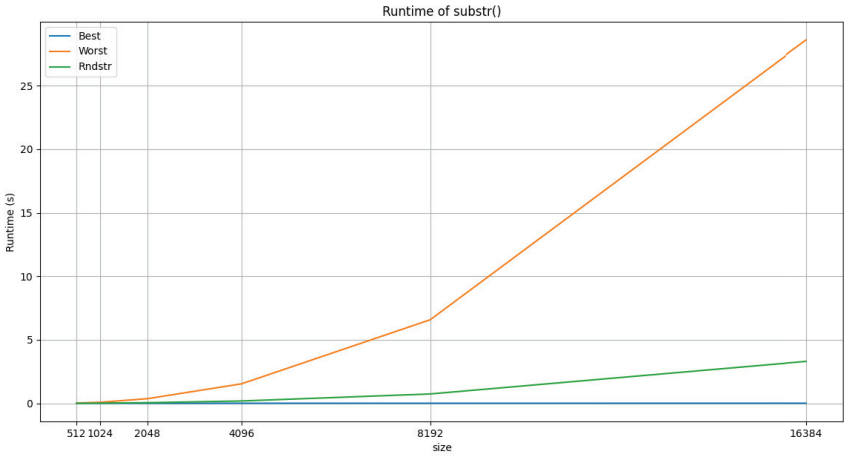

Figure 3: Runtime of different data size for 3 types of data..

Question 4:

```
fun reverse(data : array of int) {
  n : int := data.length / 2; # rounds down
  i : int := 0;
  while (i < n)
    do
      j : int := data.length - i - 1;
      temp : int := data[i];
      data[i] := data[j];
      data[j] := temp;
      i := i + 1;
    done
}
```

(a) reversed(data) $\leftrightarrow$

$$\forall i \in \mathbb{N}.0 \leq i < data.length \rightarrow data[i] = data_{orig}[data.length - i - 1]$$

(b) revregion(data, i) $\leftrightarrow$

$$\forall k \in \mathbb{N}.((0 \leq k < i) \rightarrow (data[k] = data_{orig}[data.length - k - 1])) \wedge ((i \leq k < data.length - i) \rightarrow (data[k] = data_{orig}[k])) \wedge ((data.length - i \leq k < data.length) \rightarrow (data[k] = data_{orig}[data.length - k - 1])))$$

(c) Prove : $\forall$data. revregion(data,$\lfloor$data.length/2$\rfloor$) $\rightarrow$reversed(data)

Defination :

Def_{odd} : $odd(n)$ means: $\exists x \in \mathbb{Z}.n = 2 * x + 1$

Def_{even} : $even(n)$ means: $\exists x \in \mathbb{Z}.n = 2 * x$

Lemma : $\forall n \in \mathbb{N}.odd(n) \vee even(n)$

1. $\forall$data. revregion(data,$\lfloor$data.length/2$\rfloor$) $\rightarrow$reversed(data)
   Pick any data, and...

   1.1 reversed(data)
   By case on $odd(data.length) \vee even(data.length)$.

   case: $odd(data.length)$:

1.1.1 $\exists x \in \mathbb{Z}.n = 2 * x + 1$
Use Def_{odd} and current case.

1.1.2 $date.length = 2 * x + 1$
By exist-elim on 1.1.1.

1.1.3 $data.length - x - 1 = x$
Math on 1.1.2.

1.1.4 $data.length - x = x + 1$
Math on 1.1.2.

1.1.5 revregion(data, x)
Based on 1.

1.1.6 $\forall i \in \mathbb{N}.0 \leq i < data.length \rightarrow data[i] = data_{orig}[data.length - i - 1]$
Based on defination in (a).
Pick any i, and...

1.1.6.1 $\forall k \in \mathbb{N}.((0 \leq k < x) \rightarrow (data[k] = data_{orig}[data.length - k - 1]))$
Based on defination in (b).

1.1.6.2 $\forall k \in \mathbb{N}.((x \leq k < data.length - x) \rightarrow (data[k] = data_{orig}[data.length - k - 1])))$
Pick any k.

1.1.6.2.1 $((x \leq k < data.length - x) \rightarrow (data[k] = data_{orig}[k])))$
Based on defination in (b)

1.1.6.2.2 $x \leq k < x + 1$
Math based on 1.1.4.

1.1.6.2.3 $k = x$
Math based on 1.1.6.2.2.

1.1.6.2.4 $data[k] = data_{orig}[x]$
Math based on 1.1.6.2.3.

1.1.6.2.5 $data[k] = data_{orig}[data.length - x - 1]$
Math based on 1.1.3.

1.1.6.2.6 $data[k] = data_{orig}[data.length - k - 1]$
Math based on 1.1.6.2.3.

1.1.6.3 $\forall k \in \mathbb{N}.((data.length - x \leq k < data.length) \rightarrow (data[k] = data_{orig}[data.length - k - 1])))$
Based on defination in (b).

1.1.6.4 $\forall k \in \mathbb{N}.0 \leq k < data.length \rightarrow data[k] = data_{orig}[data.length - k - 1]$
Based on 1.1.6.1, 1.1.6.2 and 1.1.6.3.

case: $even(data.length)$:

1.2.1 $\exists x \in \mathbb{Z}.n = 2 * x$
Use Def_{even} and current case.

1.2.2 $date.length = 2 * x$
By exist-elim on 1.2.1.

1.2.3 $data.length - x = x$
Math on 1.2.2.

1.2.4 revregion(data, x)
Based on 1.

1.2.5 $\forall i \in \mathbb{N}.0 \leq i < data.length \rightarrow data[i] = data_{orig}[data.length - i - 1]$
Based on defination in (a).
Pick any i, and...

1.2.5.1 $\forall k \in \mathbb{N}.((0 \leq k < x) \rightarrow (data[k] = data_{orig}[data.length - k - 1]))$
Based on defination in (b).

1.2.5.2 $\forall k \in \mathbb{N}.((data.length - x \leq k < data.length) \rightarrow (data[k] = data_{orig}[data.length - k - 1]))$
Based on defination in (b).

1.2.5.3 $\forall k \in \mathbb{N}.((x \leq k < data.length) \rightarrow (data[k] = data_{orig}[data.length - k - 1]))$
Math based on 1.2.3

1.2.5.4 $\forall k \in \mathbb{N}.0 \leq k < data.length \rightarrow data[k] = data_{orig}[data.length - k - 1]$
Based on 1.2.5.1 and 1.2.5.3.

(d) :

line 3 : i=0, n=data.length/2, revregion(data, 0)

line 5 : i<n, n=data.length/2, revregion(data, i)

line 6 : i<n, n=data.length/2, revregion(data, i), j=data.length-i-1

line 7 : i<n, n=data.length/2, revregion(data, i), j=data.length-i-1

line 8 : i<n, n=data.length/2, revregion(data, i), j=data.length-i-1,
$\forall k \in \mathbb{N}.((i{+}1 \leq k < data.length{-}i) \rightarrow (data[k] = data_{orig}[k])) \wedge$
$((0 \leq k < i+1) \rightarrow (data[k] = data_{orig}[data.length - k - 1]))$

line 9 : i<n, n=data.length/2, j=data.length-i-1
$\forall k \in \mathbb{N}.((i{+}1 \leq k < data.length{-}i) \rightarrow (data[k] = data_{orig}[k])) \wedge$
$((0 \leq k < i+1) \rightarrow (data[k] = data_{orig}[data.length - k - 1])) \wedge$
$((data.length{-}(i{+}1) \leq k < data.length) \rightarrow (data[k] = data_{orig}[data.length{-}$
$k - 1]))$, revregion(data, i+1)

line 10 : i≤n, n=data.length/2, j=data.length-i-2, revregion(data, i),

line 11 : i=n, n=data.length/2, revregion(data, data.length/2),reversed(data)