Applied Algorithms Assignment 1

Fei Fan (Peter) Chen

January 12, 2021

Exercise 1 (10 pts)

Using the fact that a fair coin can be constructed by assigning TH to be tails and HT to be heads, the expected number of tosses before we generate an unbiased coin flip is:

$$\frac{1}{Prob(toss1 \neq toss2)} \times 2 = \frac{1}{2pq} \times 2 = \frac{1}{pq}$$

Exercise 2 (10 pts)

Each randomized path has probability of $1/n^(n-1)$ since at each iteration, index j is taken from any index from $(0...n)$. The number of permutations is only however $1/n!$. Since n^n is not divisible by $1/n!$ for $n > 2$ for the reason that n is not divisible by all the primes $p < n$, then so is $n^n - 1$. Therefore, there is always some permutation P that is reachable at the end of $n - 1$ iterations by more randomized paths than some other permutation P'. Therefore this algorithm cannot generate an uniform distribution for its possible permutations.

Exercise 3 (10 pts)

Part A

Three bits are needed for 6 numbers. We assign all 3-bit numbers less than 6 to 1..6 and start over for the remaining two 3-bit combinations.

$$E = 3/4 \times 3 + 1/4 \times (3 + E) = 4\ bits \leq 4\ bits$$

Since this is <= 4 bits, the next smallest number of bits that can represent 6 numbers will have expected bits greater than 4 since probability is always less than 1.

Part B

Four bits are needed for 9 numbers, We assign all 4-bit numbers less than 9 to 1...9 and start over for the remaining 7 4-bit combinations.

$$E = 9/16 \times 4 + 7/16 \times (4 + E) = 64/9\ bits \geq 4\ bits$$

There can be room for improvement all the way up to 8 bits...possibly. We try out the next bit size 5 (we map 3 combinations to each number).

$$E = 27/32 \times 5 + 5/32 \times (5 + E) = 160/32\ bits \leq 6\ bits$$

Exercise 4 (20 pts)

N	Median-of-One		Median-of-Three		Median-of-Five	
	Set	Pivot	Set	Pivot	Set	Pivot
10	2.31	2.83	1.98	2.27	1.77	1.88
100	2.97	6.53	2.53	5.81	2.38	5.66
1000	3.38	11.77	2.84	10.12	2.48	9.17
10000	3.25	16.18	2.82	13.61	2.63	13.46
100000	3.37	20.97	2.71	17.41	2.55	16.79
1000000	3.15	24.65	2.72	21.85	2.54	20.51
10000000	3.44	30.48	2.78	25.82	2.57	24.05

Table 1: Average Case Analysis: average of 100 trials. Set Comparisons are normalized by N.

Observation 1: The number of comparisons seem to be around 2N to 3.5N as N increases from 10 to 10,000,000.

Observation 2: The number of pivots increases sub-linearly with N. In fact, the number of pivots increases at an additive constant rate of roughly 4 units for every 10x increase in N.

Observation 3: Both the number of pivots and number of comparisons in total improves with taking the median of more random pivots. As N increases, the improvement in number of total comparisons become more significant.

<u>**Code**</u>

```
import random
import statistics
import math

class QSelect():
  def __init__(self, medianN=1):
    self.medianN = medianN
    self.compSet = 0
    self.compPivot = 0

  def __call__(self, seq, k):
    # Randomly choose N elements from sequence
    medians = []
    for _ in range(self.medianN):
      medians.append(seq[int(random.uniform(0, len(seq)))])
    x = statistics.median(medians)

    # Create separate sets
```

```python
        s1 = [] # seq[i] < x
        s2 = [] # seq[i] == x
        s3 = [] # seq[i] > x
        for value in seq:
            if value < x:
                s1.append(value)
            elif value == x:
                s3.append(value)
            else:
                s2.append(value)
        self.compSet += len(seq)
        if len(s2) >= k:
            self.compPivot += 1
            return self(s2, k)
        elif len(s2) + len(s3) >= k:
            return x
        else:
            self.compPivot += 1
            return self(s1, k - len(s2) - len(s3))

    def reset(self):
        self.compSet = 0
        self.compPivot = 0

if __name__ == "__main__":
    random.Random(None)

    # Get average time complexity with 100 trials
    T = 100 # number of trials
    medianOfMedians = 5
    N = 10000000
    sequence = [i for i in range(N)]

    compSets = []
    compPivots = []

    q = QSelect(medianOfMedians)
    for _ in range(100):
        q.reset()
        q(sequence, int(math.ceil(N/2)))
        compSets.append(q.compSet/N)
        compPivots.append(q.compPivot)

    print("mean compSets:", statistics.mean(compSets))
    print("mean compPivots:", statistics.mean(compPivots))
```