QUANTUM ALGORITHMS
HOMEWORK 4 SELECTED SOLUTIONS

PROF. MATTHEW MOORE

1. In the associated python file, look at the function `primes(n)`.

(i) The Prime Number Theorem states that `len(P)` $\in \Theta(n/\ln(n))$, where `P` $=$ `primes(n)`. Using this fact, give a good upper bound on the complexity of the function `primes(n)` in terms of n. To simplify your analysis, I suggest disregarding the `break` statements.

Solution: Ignoring the break statements, at outer loop iteration k, the Prime Number Theorem tells us that the inner loop iterates $\Theta(k/\ln(k))$ times. The total number of iterations is therefore

$$T(n) = \sum_{k=2}^{n+1} \Theta\left(\frac{k}{\ln(k)}\right).$$

The function $x/\ln(x)$ is decreasing on the interval $(2, e)$, has a local minimum at $x = e$, and is then increasing on (e, ∞). Using this, we can easily calculate an upper bound:

$$\begin{aligned} T(n) &= \sum_{k=2}^{n+1} \Theta\left(\frac{k}{\ln(k)}\right) = \frac{2}{\ln(2)} + \sum_{k=3}^{n+1} \Theta\left(\frac{k}{\ln(k)}\right) \le \frac{2}{\ln(2)} + \sum_{k=3}^{n+1} \Theta\left(\frac{n+1}{\ln(n+1)}\right) \\ &= \frac{2}{\ln(2)} + (n-1)\Theta\left(\frac{n+1}{\ln(n+1)}\right) \in \Theta\left(\frac{n^2-1}{\ln(n+1)}\right). \end{aligned}$$

Therefore $T(n) \in \mathcal{O}((n^2-1)/\ln(n+1)) = \mathcal{O}(n^2/\ln(n))$.

(ii) The function `primes(n)` has exponential complexity. Explain why this is true. [*Hint: what is the complexity in terms of the number of bits of n?*]

Solution: The complexity of an algorithm is the runtime in terms of the input size *in bits*. If n is an ℓ bit number, then

$$2^{\ell-1} < n \le 2^{\ell}.$$

It follows that $n \in \Theta(2^{\ell})$. Applying this to the previous part yields

$$T(n) \in \mathcal{O}\left(\frac{n^2}{\ln(n)}\right) = \mathcal{O}\left(\frac{(2^{\ell})^2}{\ln(2^{\ell})}\right) = \mathcal{O}\left(\frac{4^{\ell}}{\ell}\right) \subseteq \mathcal{O}(4^{\ell}).$$

If we use $|\cdot|$ to represent bit size, then this can be rewritten $T(|n|) \in \mathcal{O}(4^{|n|})$.

Date: February 17, 2020.

2. Implement the Miller-Rabin probabilistic primality testing algorithm as presented in class (or in the textbook). Fill in the function `is_prime_MR(q)` in the python source file. You need only submit your function with the homework, not the entire source file.

Solution:

```
def is_prime_MR(q):
  if q <= 1:
    return False

  # Step 1
  if q % 2 == 0:
    return ( q == 2 )

  # Step 2
  k = 0
  l = q - 1
  while l % 2 == 0:
    k += 1
    l = l // 2

  # Step 3
  a = randrange(2, q-1)

  # Step 4
  a_powers = [ (a**l) % q ]
  for _ in range(k):
    a_powers.append( ( a_powers[-1]**2 ) % q )

  # Test 1
  if a_powers[-1] != 1:
    return False

  # Test 2
  for j in range(1, len(a_powers)):
    if a_powers[j] == 1 and a_powers[j-1] not in [1, q-1]:
      return False

  return True
```

3. Find five pairs of numbers $q \in \mathbb{Z}$ and $a \in \{1, \dots, q-1\}$ such that q is composite but passes the Miller-Rabin test with the given choice of a.

Solution: The smallest such pairs of numbers are given in the table below, along with the sequence of powers a^ℓ, $a^{2\ell}$, ..., $a^{2^k \ell}$.

q	a	k	ℓ	$a^\ell, a^{2\ell}, \dots, a^{2^k \ell}$
25	7	3	3	$18, 24, 1, 1$
25	18	3	3	$7, 24, 1, 1$
49	18	4	3	$1, 1, 1, 1, 1$
49	19	4	3	$48, 1, 1, 1, 1$
49	30	4	3	$1, 1, 1, 1, 1$