Computational Statistics Homework 1

Liam Dillingham

February 13, 2019

1 Question 1

Create an R program which does the following in sequence:

1.1 Create a vector x which contains the numbers $[9, 10, 11, 12, 13, 14, 15, 16]$.

```
> x = c(9,10,11,12,13,14,15,16)
```

1.2 Print the last 3 elements of x.

```
> x[(length(x)-2):length(x)]
[1] 14 15 16
```

1.3 Print out all the even numbers in x.

```
> x[lapply(x, "%%", 2) == 0]
[1] 10 12 14 16
```

1.4 Delete all the even numbers form x and print the resulting list

```
> x = x[!x %in% x[lapply(x, "%%", 2) == 0]]
> x
[1]  9 11 13 15
```

2 Question 2

Write an R program which takes as input an integer n and computes $\sum_{i=1}^{n}(1/2)^i$ in three ways: using a *for* loop, using a *while* loop, and without a loop (i.e. an analytic function). Print out the results for all three. What happens when n is very large? do you have any suggerstions on how to make your program more robust to errors when n is large?

Here is the code:

```
func = function(n, mode) {

  result = 0
  if(mode == 1) {
    for(i in 1:n) {
      result = result + 0.5 ^ i
    }
    return(result)
  } else if(mode == 2) {
    i = 1
    while(i <= n) {
      result = result + 0.5 ^ i
      i = i + 1
    }
    return(result)
  } else {
    result = 1 - (0.5 ^ n)
    return(result)
  }
}
```

3 Question 3

Show true the following:

3.1 x^3 is $\mathcal{O}(x^3)$ and $\Theta(x^3)$ but not $\Theta(x^4)$.

x^3 is $\mathcal{O}(x^3)$ as there exists a constant c (i.e. 1) such that $x^3 \leq cx^3\ \forall x \geq x_0$, where x_0 is some arbitrary but fixed x. Also, x^3 is $\Theta(x^3)$ since there exists constants c_1 and c_2 (i.e. 1 and 1) such that $c_1x^3 \leq x^3 \leq c_2x^3\ \forall x \geq x_0$. However, x^3 is not $\Theta(x^4)$ as the distance between x^3 and x^4 differs by a factor of x for any given value of x. Therefore, there is no constants c_1 and c_2 such that $c_1x^4 \leq x^3 \leq c_2x^4$.

3.2 For any real constants a and $b > 0$, we have $(n + a)^b = \Theta(n^b)$.

notice that $(n+a)^b$ follows the rule of binomial expansion. That is, the largest power of n in the expansion will be n^b. as $n \to +\infty$, The equation become dominated by the largest power of n, n^b. Thus, $(n + a)^b = \Theta(n^b)$.

3.3 $(\log(n))^k = \mathcal{O}(n)$ for any k.

Note that $log_b(x)$ is the inverse operation for x^b. Although we do not know the base for the $log(n)$ function in this problem, if we compose the inverse of two functions (logarithmic and exponential) we will essentially get the input n as the result, giving us $\mathcal{O}(n)$.

3.4 $n/(n+1) = 1 + \mathcal{O}(1/n)$

note that $\mathcal{O}(1/n)$ implies the asymtotic behavior of $1/n$. Since $1/n \to 0$ as $n \to \infty$, then the equation becomes $n/(n+1) = 1 + 0$. Then if we apply a limit as $n \to \infty$ to both sides, we get $1 = 1$ which is true.

3.5 $\sum_{i=0}^{\lceil \log_2(n) \rceil} 2^i$ is $\Theta(n)$.

Since ith iteration of the summation will be twice as much as the $i - 1$ iteration, and will be greater than the total sum of the $0 \to i - 1$ sum. Since the function is doubling for each increase in n, then we can say the function is roughly growing at a rate of $2n$, which is $\Theta(n)$.

4 Question 4

4.1 Implement the bubble sort algorithm taught in the course

```
bubblesort = function(x) {
  n = length(x)
  swapped = TRUE

  while(swapped) {
    swapped = FALSE
    for(i in 2:n) {
      if(x[i-1] > x[i]) {
        # swap
        temp = x[i-1]
        x[i-1] = x[i]
        x[i] = temp
        swapped = TRUE
      }
    }
  }
  return(x)
}
```

4.2 Compare the CPU times for mergesort and bubble sort for a sample size of 1000 normally-distributed numbers

Bubblesort time: $1.3s$
Mergesort time: $0.07s$

4.3 Calculate the sample mean and variance for the cpu times after 100 iterations

Mean Mergesort time: $0.054s$
Mergesort time variance: $0.000230s$
Mean Bubblesort time: $1.252s$
Bubblesort time: $0.0028s$

This appears to be consistent with my predictions. $\mathcal{O}(n\log(n))$ is much more efficient than bubblesort's $\mathcal{O}(n^2)$ time.