CSC 225 Assignment 1,
Julian Rocha, V00870460
A01, B02

1)

- $n^{0.1}$ is $O(n^{0.1})$
- $2^{2^\wedge n}$ is $O(2^{2^\wedge n})$
- $5n$ is $O(n)$
- $(\log n)^5$ is $O[(\log n)^5]$
- n^5 is $O(n^5)$
- 5 is $O(1)$
- 5^n is $O(5^n)$
- $n!$ is $O(n!)$
- $4^{\log n} = (2^{\log n})(2^{\log n}) = n^2$ is $O(n^2)$
- $2n \log\log n$ is $O(n \log\log n)$

Therefore we know:

$5 < n^{0.1} < 2n \log\log n < 4^{\log n} < n^5 < 5^n < n!$

Now we must determine where $(\log n)^5$ and $2^{2^\wedge n}$ rank:

- $\text{Lim}_{n\to\infty}$ $(\log n)^5/n^{0.1} = 0$, so $(\log n)^5 < n^{0.1}$
  (**Using L'Hopitals Rule**)
- $\text{Log}\,(2^{2^\wedge n}) = 2^n$ which is $O(2^n)$ and $\log(n!)$ is $\theta(n \log n)$ (**See work for q. 2**) so since the log of the first function grows faster than the log of the second function, $n! < 2^{2^\wedge n}$

Sol: $5 < (\log n)^5 < n^{0.1} < 2n \log\log n < 4^{\log n} < n^5 < 5^n < n! < 2^{2^\wedge n}$

2)

$S(n) = \sum_{i=1}^{n} \log i = \log 1 + \log 2 + \ldots + \log n$

- $S(n) \leq \log n + \log n + \ldots + \log n = n \log n$

Upper bound: S(n) is O(n log n)

- $S(n) = \log 1 + \log 2 + \ldots + \log (n/2) + \ldots + \log n$
  $S(n) \geq \log (n/2) + \log(n/2 + 1) + \ldots + \log n$
  $S(n) \geq \log (n/2) + \log (n/2) + \ldots + \log (n/2) = (n/2)\log (n/2)$

Lower bound: S(n) is Ω(n log n)

Since upper and lower bound of S(n) is n log n, we can conclude that **S(n) is θ(n log n)**

3)

a.

```
int sum = 0;
for (int n = N; n > 0; n /= 2)
   for (int i = 0; i < n; i++)
      sum++;
```

- Outer loop runs log N times, and inner loop condition is divided by 2 every time, substitute variables:

```
int sum = 0;
for (int n = 0; n < log N; n++)
   for (int i = 0; i < N/2n; i++)
      sum++;
```

- This becomes summation $= \sum_{n=0}^{\log N} N/2\text{^}n$
- $= \sum_{n=0}^{\log N} N(1/2)^n$, geometric series
- $=[N (1 - (½)^{\log N + 1})]/[1 - ½]$
- $=(2N)(1 - 1/2N)$
- $=2N - 1$
- **Therefore, the order of growth is N (linear)**

b.

```
int sum = 0;
for (int i = 1; i < N; i *= 2)
   for(int j = 0; j < i; j++)
      sum++;
```

- Outer loop runs log N times and inner loop condition is multiplied by 2 every time, substitute variables:

```
int sum = 0;
for (int i = 0; i < log N; i++)
   for(int j = 0; j < 2^i; j++)
      sum++;
```

- This becomes summation: $\sum_{n=0}^{\log N} 2\hat{}n$, geometric series
- = [1 – $2^{\log \text{N} + 1}$] / [1 – 2]
- =2N -1
- **Therefore, the order of growth is N (linear)**

c.

```
int sum = 0;
for (int I = 1; I < N; I *= 2)
   for (int j = 0; j < N; j++)
      sum++;
```

- Outer loop runs log(N) times
- Inner loop runs N times each time
- Outer * Inner = N log N
- **Therefore, the order of growth is N log N (linearithmic)**

4)

$\sum_{i=1}^{n}(2i - 1) = n^2$ for all $n \geq 1$

<u>Base Case:</u>

- For $n = 1$, $2(1) - 1 = 1 = 1^2$, therefore RHS = LHS

<u>Inductive Hypothesis:</u>

- Assume $\sum_{i=1}^{k}(2i - 1) = k^2$ for some $k \geq 1$

<u>Inductive Step</u>: Simplify $k + 1$ starting with LHS

- LHS = $[2(1) - 1] + [2(2) - 1] + \ldots + [2(k) - 1] + [2(k + 1) - 1]$
- $= k^2 + 2(k + 1) - 1$, *from I.H.
- $= k^2 + 2k + 1$
- $= (k + 1)^2 =$ RHS

5)

<u>Pseudocode</u>:

```
num ← n – 1
for i ← 0 to (n – 2) do
    num ← num + i – A[i]
end
return num
```

<u>Description</u>:

This Algorithm finds the missing number from the range by taking the sum of all numbers in the range 0 to n – 1 (“i” variable) and subtracting all n – 1 array elements (“A[i]” variable). The missing number from the range is what is leftover in the end.

It is O(n) – time because there is a single for loop which loops n – 1 times. The summation of the range as well as the subtraction of the array elements are both performed inside the same loop. This single loop setup therefore operates in linear time.

It is O(log n) bits space because no additional arrays were used, and only 2 additional variables were used: integer i (4 bytes) and integer num (4 bytes) = 8 bytes = 64 bits. Arrays of type int in Java typically require ~4N bytes + 24 bytes of header information = ~224N bits. Therefore, for arrays of size N = $2^{64}/224$ or larger, the bits used in the algorithm will be less than or equal to the bits in array A.