# SIT221 Practical task 2.1

In this task, answer all the following questions and complement each answer with a detailed explanation.

1. Review the following algorithms (assume any undeclared variables are declared earlier)

i.

```
int a = 0;
a += rand();
if( a < 0.5 ) a += rand();
if( a < 1.0 ) a += rand();
if( a < 1.5 ) a += rand();
if( a < 2.0 ) a += rand();
if( a < 2.5 ) a += rand();
if( a < 3.0 ) a += rand();
```

ii.

```
int count = 0;
for (int i = 0; i < N; i++)
{
    int num = rand();
    if( num < 0.5 ){
        count += 1;
    }
}
```

iii.

```
int count = 0;
for (int i = 0; i < N; i++)
{
    if (unlucky)
    {
        for (j = N; j > i; j--)
        {
            count = count + i + j;
        }
    }
}
```

iv.

```
int count = 0,
int i = N;

if (unlucky)
{
    while (i > 0)
    {
        count += i;
        i /= 2;
    }
}
```

v.

```
int count = 0;

for (int i = 0; i < N; i++)
{
    int num = rand();
    if( num < 0.5 )
    {
        count += 1;
    }
}
int num = count;
for (int j = 0; j < num; j++)
{
     count = count + j;
}
```

vi.

```
for (int i = 0; i < N - 1; i++)
{
    for (int j = 0; j < N-i-1; j++)
    {
        if (a[j] > a[j+1])
        {
            Swap(a[j], a[j + 1]);
        }
    }
}
```

## Labelling of the number of operations in worst case

i.

a)

Worst case – 14 operations
Best case- 10 operations (we assume that none of the if condition is true and therefore a+=rand() is not performed for all of them).
Average case- 8 operations (only half of the if statement conditions are met and therefore a+=rand() is performed only on 4 occasions).

b)

Worst- Θ(1)
Best- Θ(1)
Average- Θ(1)

There are no loops in this program so time will be constant for all three of them.

c) O(1) because this is the closest possible upper bound to Θ(1).

d) Ω(1) because the algorithm is going to take at least 8 steps regardless of the value of a.
e) Best and worst case is the same. Hence, the overall performance using Big-theta is Θ(1).
f) The best way is to use Big-theta because best and worst case is the same.

ii.
a)
Worst case – 1+1+(n+1)+n+n+n+n = 5n+3
Best case – 4n+3 (assuming that num is greater than 0.5 for all values less than N, so count+=1 is never ran)
Average case- 4.5n+3 (assuming that count += 1 operation is ran only half of the time. Hence, adding n/2 not n.

b)
Worst- Θ(n)
Best- Θ(n)
Average- Θ(n)

c) O(n) because this is the closest possible upper bound to Θ(n).
d) Ω(l) because the algorithm cannot be faster than n.
e) Best and worst case is the same. Hence, the overall performance using Big-theta is Θ(n).
f) The best way is to use Big-theta because best and worst case is the same.

iii.
a)
Worst Case- 1+1+(n+1)+n+n+n+ $n\{\left(\frac{n+1}{2}\right)+1\}+n\left(\frac{n+1}{2}\right)+n\left(\frac{n+1}{2}\right)$ = $\frac{3n\text{^}2}{2}+\frac{13n}{2}+3$
Best case- 3n+3 (assuming that if statement is false for all the values of i less than N in the array).
Average Case- Assuming that the if statement runs for half of the time in the iteration.
1+1+(n+1)+n+n/2+n+ $n\{\left(\frac{n+1}{2}\right)+1\}+n\left(\frac{n+1}{2}\right)+n\left(\frac{n+1}{2}\right)$ = $\frac{3n\text{^}2}{2}+6n+3$

b)
Worst- Θ(n^2)
Best- Θ(n)
Average- Θ(n^2)

c) O(n^2) because this is the closest possible upper bound to Θ(n^2).
d) Ω(n) because this algorithm cannot be faster than Ω(n).
e) Worst and best case is in different classes. Hence Big-theta cannot be defined.
f) Since Big-O and Big- Ω is different, Big-O is the best way to describe the overall performance here.

iv.
a)
Worst case- 1+1+1+( $\log_2 n$)+1+1 = 5+ $\log_2 n$
Best case- 3 operations (assuming that if statement is false and nothing inside it is ran)
Average case- The average case is hard to determine because the while loop run only when the if-statement is false at the first check.

b)
Worst- Θ(logn)
Best- Θ(1)
Average- Θ(logn)

c) O(logn) because this is the closest possible upper bound to Θ(logn).
d) Ω(1) because this algorithm cannot be faster than Ω(1).
e) Worst and best case is in different classes. Hence Big-theta cannot be defined.
f) Since Big-O and Big- Ω is different, Big-O is the best way to describe the overall performance here.

v.
a)
Worst case- 1+1+(n+1)+n+n+n+n+1+1+(n+1)+n+n =6+8n
Best case- 6+4n (assuming that the if(num < 0.5) condition is false for every iteration. Therefore, count += 1 will never run and as a result the second for loop will never run because j will not be less than num).
Average case- 6+6n (assuming that the if(num < 0.5) condition is true for the half of the time and false for the other half. As a result, count += 1 will run half the amount of time and this will cause the iteration of the next for-loop to be half the amount of time. Overall, count = count + j runs half the amount of time.

b)
Worst- Θ(n)
Best- Θ(n)
Average- Θ(n)

c) O(n) because this is the closest possible upper bound to Θ(n).
d) Ω(n) because this algorithm cannot be faster than Ω(n).
e) Best and worst case is the same. Hence, the overall performance using Big-theta is Θ(n).
f) The best way is to use Big-theta because best and worst case is the same.

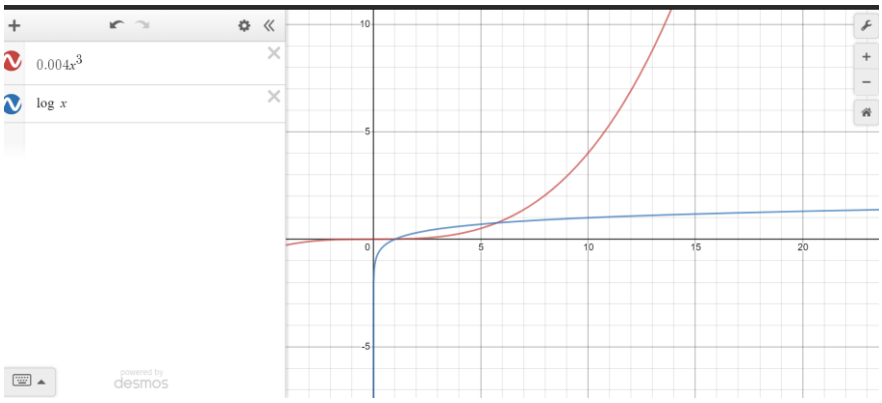
vi.
a) Worst - 1+n+n+1+ $n\left(\frac{n+1}{2}\right)$+ $n\left(\frac{n+1}{2}\right)$+ $n\left(\frac{n+1}{4}\right)$+ $n\left(\frac{n+1}{2}\right)$ = $2n^2+4n+2$
Best- $1.5n^2+3.5n+2$ (assuming that if (a[j] > a[j+1]) is false for every iteration as array is already sorted. Therefore, swap() is never ran.
Average- $1.75n^2+3.75n+2$ (assuming that if (a[j] > a[j+1]) is false for half the amount of time and true for the other half.
b)
Worst- $\Theta(n^2)$
Best- $\Theta(n^2)$

Average- $\Theta(n^2)$

c) O(n^2) because this is the closest possible upper bound to Θ(n^2).
d) Ω(n^2) because this algorithm cannot be faster than Ω(n^2).
e) Best and worst case is the same. Hence, the overall performance using Big-theta is $\Theta(n^2)$.
f) The best way is to use Big-theta because best and worst case is the same.

2. Big-O is mostly used because it is used to describe the worst case scenario when using an algorithm. While using algorithms, people are more interested in the worst case than the best case because that prepares us for the worst cases where the performance of our algorithm will be the worst.

3. $\Theta(n^3)$ algorithm does not always takes longer to run than an Θ(logn) algorithm. If we multiply either of the functions with a number/constant, their graphs might intersect at one point. For example, if we we have 0.004n^3 and log n, initially the log n is climbing at a higher rate than n^3. But after some time they intersect and n^3 starts increasing at a higher rate.




4.
i) In both side of the equation we have n^2 as the highest power. Ignoring the constant values that we have on the left side, we can say for large values of n, O(n^2) will be right. This statement is **correct**.

ii) This is **incorrect** because the Big-O determines the upper bound of an equation and upper bound of n log n is O(n^2) not O(n).

iii) Here the highest order of power is n^3. For a function to be big-theta, it has to be tightly bounded by big-O and big-omega, which means have the same order of power in the best and worst case. Here $\Theta(n^4)$ is incorrect as highest power is n^3.

iv) This is correct because big-omega refers to the lower bound of an equation and lower bound of nlogn is Ω(n).