Nisharg Gosai (002273353)
Algorithms Assignment 1

1) In the following, use a direct proof (by giving values for c and n0 in the definition of big-O/Ω notation) to prove that:

a) n^2+7n+1 is $\Omega(n^2)$

Let above equation be f(n), Using the formal definition of big-Ω,
To prove f(n) = n^2+7n+1 is $\Omega(n^2)$, we need to find values of c and n0 such that n^2+7n+1 >= cn^2

n^2+7n+1 >= cn^2

If we take c=9,

n^2+7n+1 >= 9n^2
0>=8n^2-7n-1

Using formula $\frac{-b\pm\sqrt{b^2-4ac}}{2a}$, to find roots of the equation 8n^2-7n-1,

a=8,b=-7,c=-1,

$\frac{7\pm\sqrt{81}}{16}$

$\frac{7+9}{16}$, $\frac{7-9}{16}$

n=1,n=$\frac{-1}{8}$

For c=9 and n0=1 f(n) belongs to $\Omega(n^2)$

b) 3n2 + n – 10 is O(n2)

Let above equation be f(n), Using the formal definition of big-O,

To prove $3n^2$+n-10 is big-O, we need to find values of c and n0

such that $3n^2$+n-10 <= cn^2

$3n^2$+n-10 <= cn^2

If we take c=4,

$3n^2$+n-10 <= 4n^2

$-n^2 + n <= 10$

n(-n+1) <= 10

n<=10 and n<=-9

For c=4 and n0=10 f(n) belongs to O(n^2)

c) n^2 is $\Omega(nlogn)$

Let above equation be f(n), Using the formal definition of big-Ω,

To prove f(n) = n^2 is $\Omega(nlogn)$, we need to find values of c and n0 such that n^2>= cnlogn

n^2>= cnlogn

If we take c = 2 and assume log base is 2

n^2>= 2nlogn

$n^2 - 2nlogn >= 0$

$n(n - 2logn >= 0$

$n >= 0\ and\ n - 2logn >= 0$

Solving for $n - 2logn >= 0$

$n >= 2logn$

$\frac{n}{2} >= logn$

$2^{\frac{n}{2}} >= n$

The above equation is true for n=2,

Therefore, For c=2 and n0=2 f(n) belongs to $\Omega(n^2)$

2) In the following, use the iteration method to find the asymptotic notation of the order of growth of the recurrences:

a. T(n) = $2T(\frac{n}{2}) + b\ if\ n > 1$

T(n) = $2T(\frac{n}{2}) + b$

$T(\frac{n}{2}) = 2T(\frac{n}{4}) + b$

$T(\frac{n}{4}) = 2T(\frac{n}{8}) + b$

Using above equations,

T(n) = $2T(\frac{n}{2}) + b$ ----- (1)

$= 2\left[2T(\frac{n}{4}) + b\right]$ + b

= $4T\left[\frac{n}{4}\right]$ + 3b ----- (2)

= $4\left[2T(\frac{n}{8}) + b\right]$ + 3b

= $8T(\frac{n}{8}) + 7b$ -------(3)

Our general equation will be,

$2^kT(\frac{n}{2^k}) + (2^k - 1)b$

We need to take a value for k which causes recurrence to reach base case which Is T(1) = 1

$\frac{n}{2^k} = 1$

$n = 2^k$

$k = lg\, n$

Putting k = lg n, in $2^kT(\frac{n}{2^k}) + (2^k - 1)b$

We get,

$= 2^{lgn}T(\frac{n}{2^{lgn}}) + (2^{lgn} - 1)b$

$= n^{lg2}T(\frac{n}{n^{lg2}}) + b.n^{lg2} - b$

= n + b.n - b

Therefore, the time complexity of our recurrence is O(n).

b. $T(n) = T(n - 1) + n + b$ if n>1

$T(n) = T(n - 1) + n + b$
$T(n - 1) = T(n - 2) + (n - 1) + b$
$T(n - 2) = T(n - 3) + (n - 2) + b$

Using above equations,

$T(n - 1) + n + b$ —---------(1)
$[T(n - 2) + (n - 1) + b] + n + b$
$T(n - 2) + 2n + 2b - 1$ —-------------(2)
$[T(n - 3) + (n - 2) + b] + 2n + 2b - 1$
$T(n - 3) + 3n + 3b - 3$ ————————— (3)

Using 1,2 and 3 we can write the general equation as,

$T(n - k) + kn + kb - (2^{k-1} - 1)$

We need to take a value for k which causes recurrence to reach base case which Is T(0) = c

n-k=0
k=n

Putting k=n in $T(n - k) + kn + kb - (2^{k-1} - 1)$

$c + n^2 + bn - (2^{n-1} - 1)$

Therefore, the time complexity of our recurrence is O(2^n)

3) Solve the following recurrences using the substitution method:

a. $T(n) = T(n - 3) + 3lgn$

Our guess is T(n) is O(nlgn)

We need to show that T(n) <= cnlgn for some constant c>0

$T(n) = T(n - 3) + 3lgn$

Substituting cnlgn in above equation,

$= c(n - 3)lg(n - 3) + 3lgn$

Since lgn monotonically increasing for n>0, we can say

cnlgn+3lgn > $c(n - 3)lg(n - 3) + 3lgn$

Since we are finding the upper bound we can ignore 3lgn which is smaller than cnlgn,

Therefore we get,

T(n)<=cnlgn

Therefore, T(n) belongs to O(nlgn)

b. T(n) = 4T(n/3) + n

Our guess is T(n) = O($n^{log_3 4}$)

We need to show that T(n) <= c$n^{log_3 4}$ for some constant c>0

T(n) = 4T(n/3) + n

Substituting $cn^{log_3 4}$ in above equation,

$$=4c(\frac{n}{3})^{log_3 4} + n$$

$$= 4c(\frac{n^{log_3 4}}{3^{log_3 4}}) + n$$

$$= 4c(\frac{n^{log_3 4}}{4^{log_3 3}}) + n$$

$$= cn^{log_3 4} + n$$

Since we are left with n in the equation we have to take a new guess,
Our new guess is $cn^{log_3 4}$ - dn,

Substituting the latest guess,

$$= 4(c(\frac{n}{3})^{log_3 4} - d(n/3) + n$$

$$= 4c(\frac{n^{log_3 4}}{4^{log_3 3}}) - 4d(n/3) + n$$

$$= cn^{log_3 4} - (4/3)dn + n$$

If (4/3)d = 1, then we are left with,

$$= cn^{log_3 4}$$

Therefore, T(n) = O($n^{log_3 4} - n)$

4) Insertion sort as a recursive algorithm,

```
RecursiveInsertion(A,n)

        If n<=1                     //base case (only 1 element to sort)
                return

        RecursiveInsertion(A,n-1)       //recursive call

        key = A[n]
```

```
j=n-1

While j>0 and A[j]>key      //index starting from 1

        A[j+1]=A[j]
        j=j-1

A[j+1] = key
```

The worst case recurrence for algorithm is $T(n)=T(n-1)+O(n)$ if $n>1$
Worst case time complexity is $O(n^2)$

5) Let $f(n)$ and $g(n)$ be asymptotically nonnegative functions. Using the basic definition of Θ-notation, prove that $\max\{f(n), g(n)\} = \Theta(f(n)+g(n))$.

According to the formal definition of Big Theta, a function $f(n)$ belongs to $\Theta(g(n))$,
If there are constants $c1, c2, n0>0$ for all $n>n0$ such that,

$$0 <= c1g(n) <= f(n) <= c2g(n)$$

To prove that $\max\{f(n), g(n)\} = \Theta(f(n)+g(n))$, we need to prove

$\max\{(f(n),g(n)\} = O(f(n)+g(n))$ and $\max\{f(n),g(n)\} = \Omega(f(n)+g(n))$

If $f(n)>g(n)$, then

$$\max\{f(n),g(n)\} >= g(n) \text{ —--(1)}$$

If $g(n)>f(n)$, then

$$\max\{f(n),g(n)\} >= f(n) \text{ —---(2)}$$

Adding (1) and (2), we get

$$2[\max\{f(n),g(n)\}] >= f(n)+g(n)$$
$$\max\{f(n),g(n)\} >= ½[f(n)+g(n)]$$

We can say $c1=½$ and therefore $\max\{f(n),g(n)\} >= c1[f(n)+g(n)]$
Which means $\max\{f(n),g(n)\}$ is $\Omega(f(n)+g(n))$ —---(A)

Also, f(n)<=f(n)+g(n) and g(n)<=f(n)+g(n)

If f(n)>g(n), then

max{f(n),g(n)}<= f(n)+g(n) ———-(3)

If g(n)>f(n), then

max{f(n),g(n)}<= f(n)+g(n) ————-(4)

Therefore using (3) and (4), we get

max{f(n),g(n)}<= 1.[f(n)+g(n)]

We can say c2=1 and therefore max{f(n),g(n)} <= c1[f(n)+g(n)]

Which means max{f(n),g(n)} is O(f(n)+g(n)) ———-(B)

Using (A) and (B), we have

max{f(n), g(n)} is Θ(f(n)+g(n)) which is our answer.

6) Is 2n+1 = O(2n)? Is 22n = O(2n)? Use the formal definition of O-notation to answer these two questions.

a. $2^{n+1} = O(2^n)$?

Using the formal definition of Big O, $2^{n+1} <= c2^n$,

Since $2^{n+1} = 2^n.2^1$,

$2^{n+1} <= c2^n$

$2^n.2^1 <= c2^n$

If we take c=2,

$2^n.2^1 <= 2.2^n$ which is true

The function will never grow faster than c2^n for c>=2 and n>=1

Therefore, for c>=2, $2^{n+1}\ is\ O(2^n)$

b. $2^{2n} = O(2^n)$?

Using the formal definition Big O, $2^{2n} <= c(2^n)$,

$2^{2n} <= c(2^n)$
$2^n . 2^n <= c2^n$
$2^n <= c$

This is not possible since constant c cannot be greater than 2^n for all n
Therefore 2^{2n} $is\ not\ O(2^n)$