---

exercise 1

- chosen invariant - $exp(n) = 3^n$

- initialisation
  when n = 0, $exp(n)$ should equal 3^0 (1), and this is correct according to the function

- step cases:
  n is even, $exp(2n)$
  when n is even, return $exp(2n/2) \cdot exp(2n/2)$
  $= exp(n) \cdot exp(n)$
  using the inductive hypothesis, $exp(n) = 3^n$
  $3^n \cdot 3^n$ = 3^{2n}
  so $exp(2n) = 3^{2n}$ which satisfies our invariant

n is odd, $exp(2n+1)$
when n is odd, return $3 \cdot exp(((2n+1)-1)/2) \cdot exp(((2n+1)-1)/2)$
$= 3 \cdot exp(2n/2) \cdot exp(2n/2)$
$= 3 \cdot exp(n) \cdot exp(n)$
using induction hypothesis $\exp(n) = 3^n$
return $3 \cdot 3^n \cdot 3^n$
$= 3^{2n+1}$
so $exp(2n+1) = 3^{2n+1}$, which satisfies our invariant

base case, and both possible stepcases hold, $\therefore$ proved

---

exercise 2

$T(n) = T(\lceil \frac{n}{2} \rceil) + 1$ is $O(log_2 n)$

Our inductive hypothesis is:
$T(n) \leq clog_2 n$

base case:
$n = 2$

$T(2) = T(\lceil \frac{2}{2} \rceil) + 1$
$T(1) = T(1) + 1$
$T(1) = 1 + 1$ (Using $T(1) = 1$ as stated)
$T(1) = 2$

$clog_2 2 = c$
Therefore $T(2) \leq clog_2 n$ given that c ≥ 2

$n = 3$
$T(3) = T(\lceil \frac{3}{2} \rceil) + 1$
$T(3) = T(2) + 1$
$T(3) = 2 + 1$ (Using $T(2) = 2$ as stated)
$T(3) = 3$

$clog_2 3 = 1.584c$
Therefore $T(3) \leq clog_2 n$ given that $c \geq 2$

step case:
inequality holds when n is even:
$T(n) \leq T(\lceil \frac{2n}{2} \rceil) + 1$
$T(n) \leq T(\frac{2n}{2}) + 1$
$= T(n) + 1$
$= clog_2 n + 1$ (using inductive hyp.)
$\leq clog_2 n$

inequality holds when n is odd:
$T(n) \leq T(\lceil \frac{2n+1}{2} \rceil) + 1$
$T(n) \leq T(\frac{2n+1}{2}) + 1$
$= clog_2 \frac{2n+1}{2} + 1$ (using inductive hyp.)
$= (clog_2(2n+1) - clog_2 2) + 1$
$= (clog_2(2n+1) - 1) + 1$
$= clog_2(2n+1)$
$= clog_2(2(n + \frac{1}{2}))$
$= clog_2 2 + clog_2(n + \frac{1}{2})$
$= 1 + clog_2(n + \frac{1}{2})$
$\leq log_2(n)$ (constants can be disregarded)

Therefore, $T(n) = O(log_2 n)$

---

exercise 3

$T(n) = 2T(n^{\frac{1}{4}}) + 1$, if $n > 1$
$T(n) = 1$ if $n = 1$

$m = logn$ (replacement)
$n = 2^m$
$T(2^m) = 2T(2^{m/4}) + 1$
$S(m) = 2S(m/4) + 1$

Solving with master method
$a = 2, b = 4, f(m) = 1$
$log_b a = log_4 2 = \frac{1}{2}$
$f(n) = O(m^{log_b a - \epsilon})$
$f(m) = O(m^0) = O(1)$, where $\epsilon = \frac{1}{2}$
Therefore $S(m) = \Theta(m^{1/2})$ (As master method gives tight bound)
$T(n) = T(2^m) = S(m)$
$O(m^{1/2}) = O(\sqrt{logn})$

Therefore $T(n) = \Theta(\sqrt(logn))$

---

ex 4

pseudocode for modified binary search

```
binarySearch(l, r, nums, target)
        if l >= r then
                if target = nums[l] then
                        return true
                else then
                        return false

        midpoint = l + ((r − l) div 3)

        if nums[midpoint] < target then
                return binarySearch(midpoint + 1, r, nums, target)
        else if nums[midpoint] > target then
                return binarySearch(l, midpoint − 1, nums, target)
        else
                return true //if we arent > nor <, mid number must == target
```

we can analyse the upper bounds of each line within the code
`midpoint = l + ((r - l) div 3)` - $O(1)$
`return binarySearch(midpoint + 1, r, nums, target)` - $T(2n/3)$
`return binarySearch(l, midpoint - 1, nums, target)` - $T(n/3)$
we only ever take one of the two `return binarySearch(...)` calls, since they are within the same if-else block. In the worst case scenario, we will be calling $T(2n/3)$ each time, since that leaves us with the list of the bigger size. Therefore, considering the constants too, we can represent the recurrence relation of the binary search as:

- $T(n) = T(2n/3) + \Theta(1)$ if n > 1
- $T(n) = 1$ if n = 1
  Due to no recursive call being made in the case that the current list/sublist is of size 1

substitution method
Guess: $T(n) = O(log_{3/2}n)$

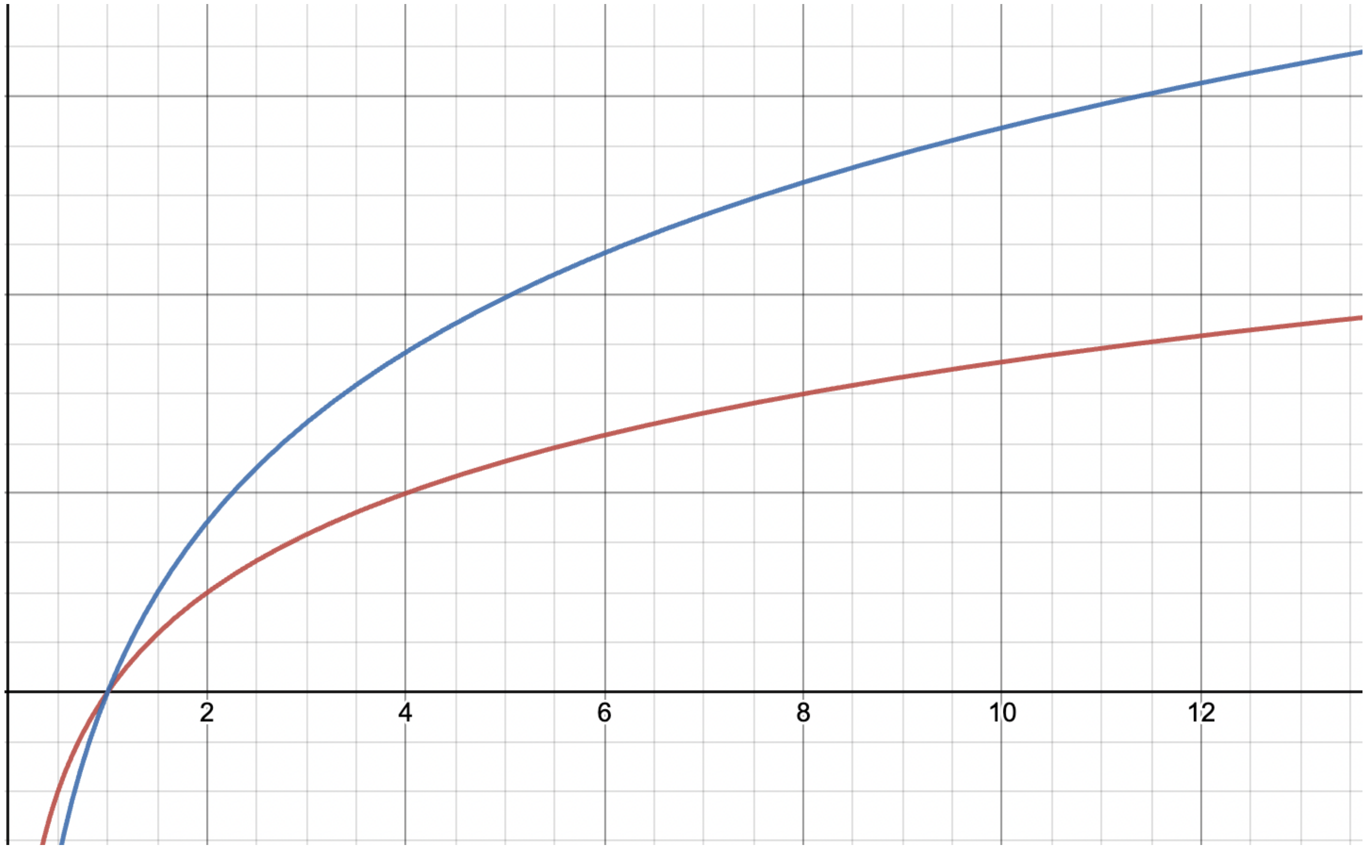
Therefore, the inductive hypothesis is:
$T(n) \leq clog_{3/2}n$

base case: $n = 2$
$T(n) = \ clog_{3/2}2 + 1$
$= clog_{3/2}2 + 1$
$= c \cdot 1.709... + 1$
$1.709c \leq clog_{3/2}2$ for all constants $c \geq 1$, therefore the base case holds

step case: $3n/2$
$T(n) \leq clog_{3/2}3n/3 + 1$
$= c(log_{3/2}3n - log_{3/2}2) + 1$
$= c((log_{3/2}3 + log_{3/2}n) - log_{3/2}2) + 1$
$= c((2.709... + log_{3/2}n) - 1.709...) + 1$
$= c(log_{3/2}n + 1) + 1$
$= clog_{3/2}n + c + 1$
$\leq clog_{3/2}n$ (disregard constants)

Therefore $T(n) = O(log_{3/2}n)$

- Comparison
  We can graph both upper bounds of the binary searches - Given that the traditional binary search is = $O(log_2n)$, when plotting the two



where blue is modified binary search, and red is regular binary search
as can be seen in the graph, modified binary search dominates regular binary search when $n \geq 1$, therefore has a higher upper bound from this point. This means the running time may be worse than the regular binary search

---

ex 5

Master Method is used for A, B and C

(A)
$T(n) = 2T(n/2) + n^4$
$a = 2, b = 2, f(n) = n^4$
$n^{log_b a} = n^{log_2 2} = n^1 = n$
$f(n) = n^4$
$f(n) = \Omega(n^{log_2(2+\epsilon)})$ *(case 3)*
$f(n) = \Omega(n^4)$
$\epsilon = 14$
$af(n/b) \leq cf(n)$ where $c < 1$
$2((n/2)^4) \leq cn^4$
$2((n^4/16)) \leq cn^4$
$n^4/8 \leq cn^4$

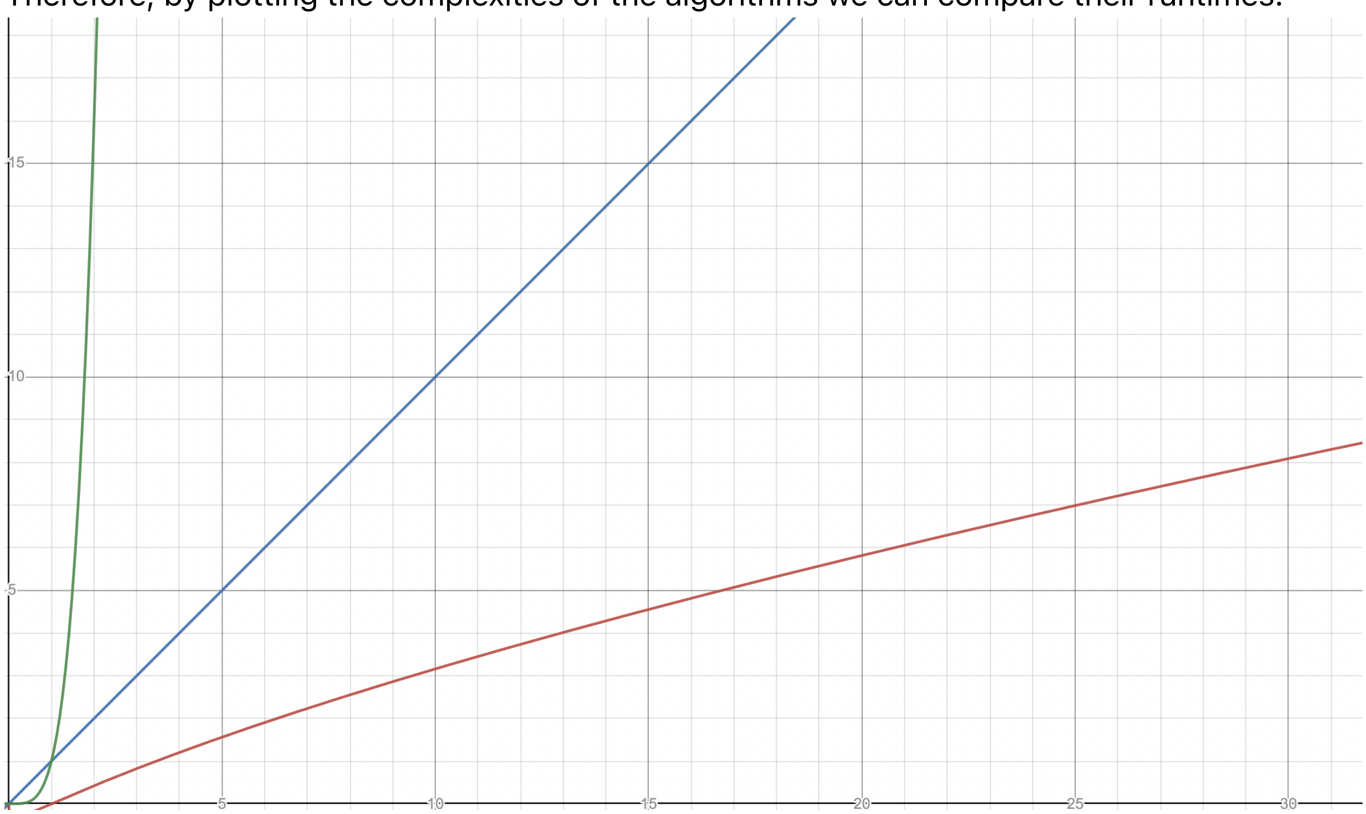
$c = 1/2, n^4/8 \leq n^4/2$ holds
therefore $T(n) = \Theta(n^4)$

(B)
$T(n) = T(10n/7) + n$
$a = 1, b = 10/7$
$f(n) = n$
$n^{log_b a} = n^{log_{10/7} 1}$
$log_b a = 0$
$f(n) = \Theta(n)$
$f(n) = \Omega(n^{0+1})$ (case 3)

therefore $T(n) = \Theta(n)$

(C)
$T(n) = 2T(n/4) + \sqrt{n}$
$a = 2, b = 4, f(n) = \sqrt{n}$
$log_b a = log_4 2$
$log_b a = \frac{1}{2}$
$f(n) = \Theta(n^{\frac{1}{2}})$ (case 2)
therefore $T(n) = \Theta(\sqrt{n} logn)$

Therefore, by plotting the complexities of the algorithms we can compare their runtimes:



Given that C is red, B is blue, A is green so from fastest to slowest, the algorithms are ordered: C, B, A