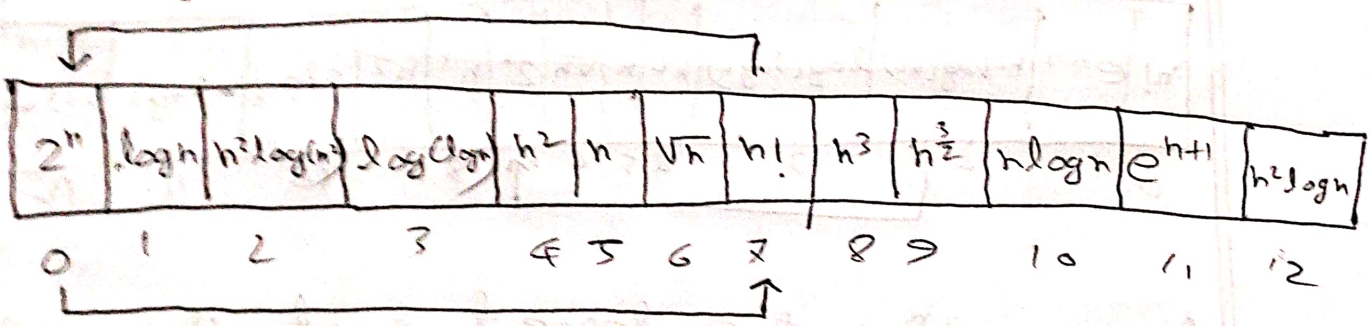
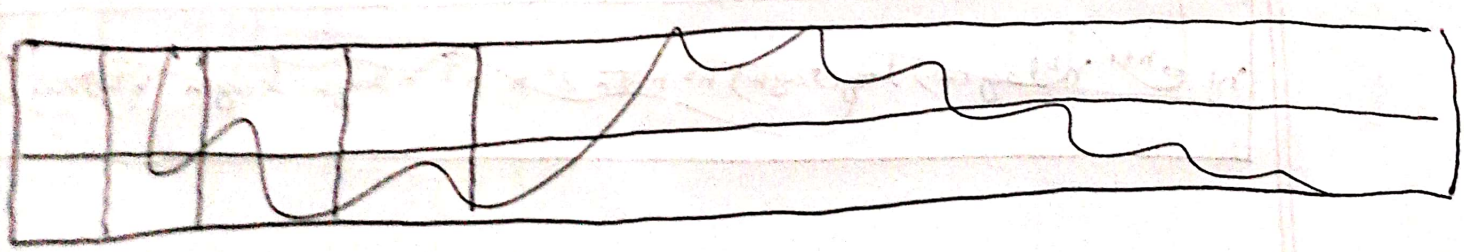
1.(a) Given,

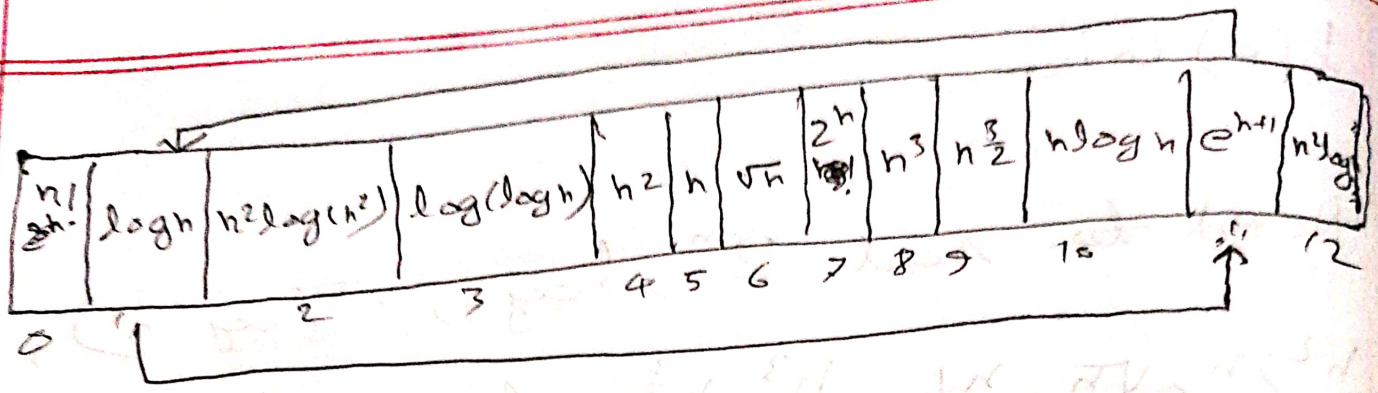
Need to sort 2^n, $\log n$, $n^2 \log(n^2)$, $\log(\log n)$, n^2, n, $\sqrt{n}$, $n!$, n^3, $n^{\frac{3}{2}}$, $n \log n$, e^{n+1}, $n^2 \log n$

For selection sort, lets put them in an array.



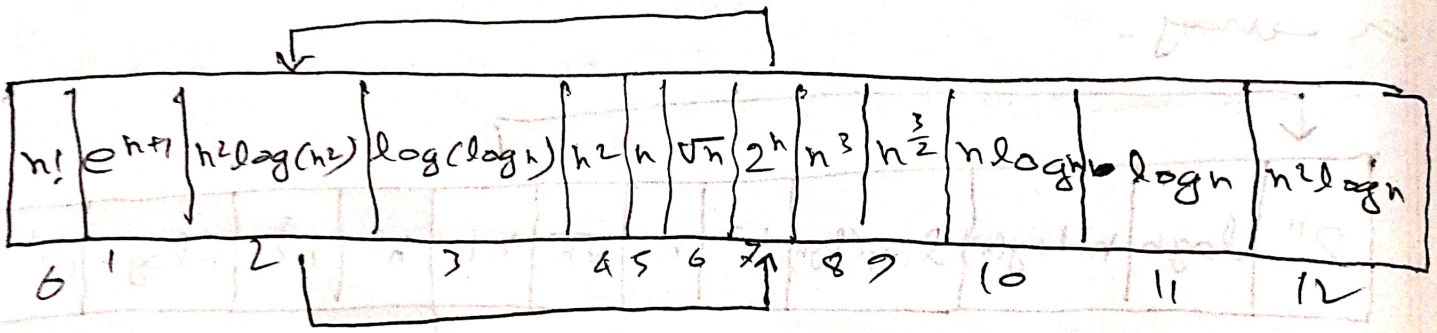
As $n!$ is the largest in array so there will be swapping.



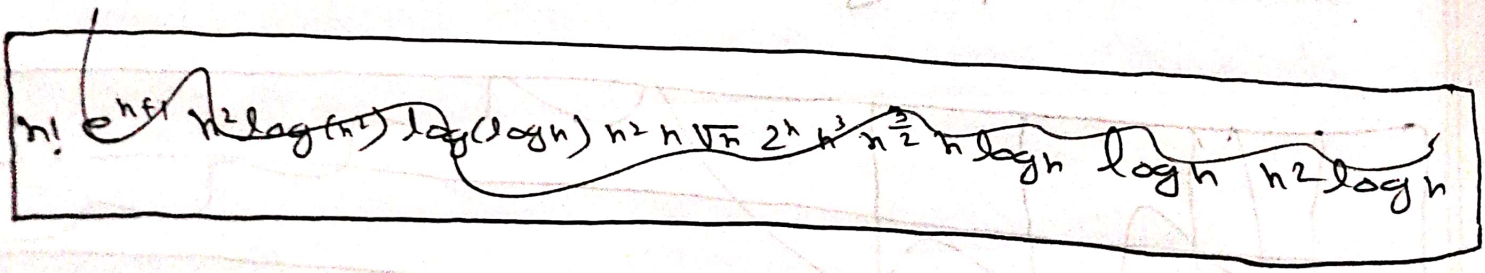


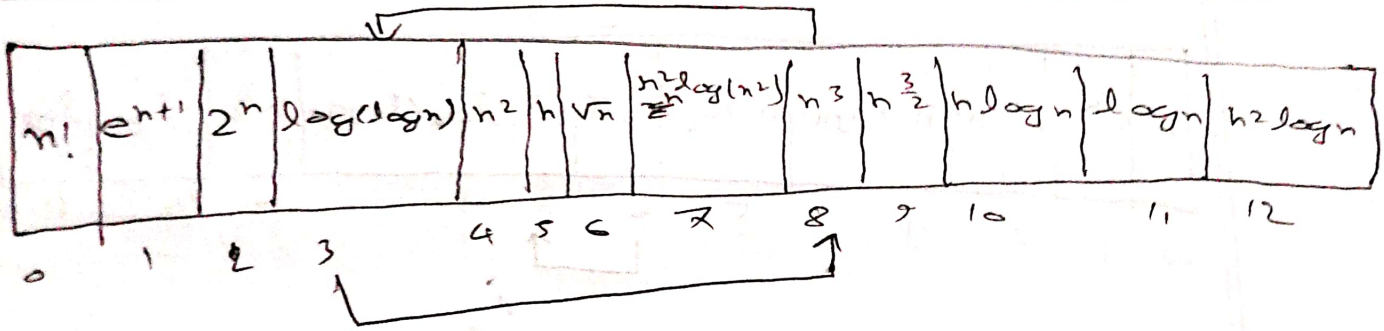
Now from the remaining array

e^{n+1} is the largest so they will swap.

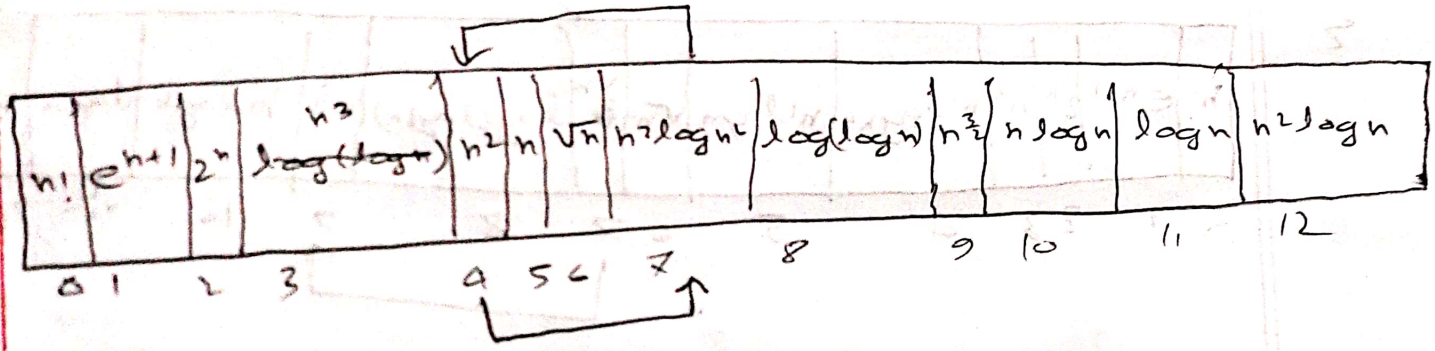


As now 2^n is the largest so it will swap.

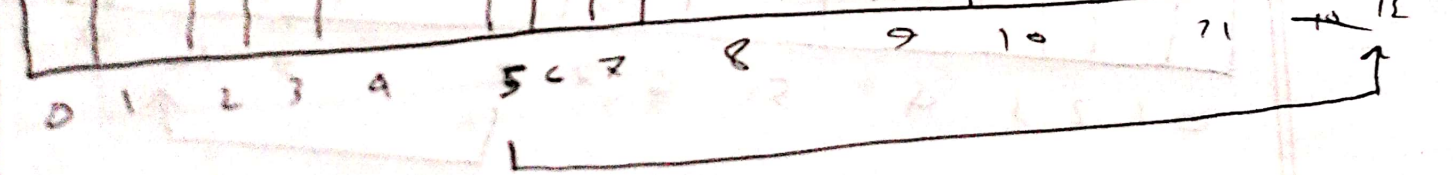




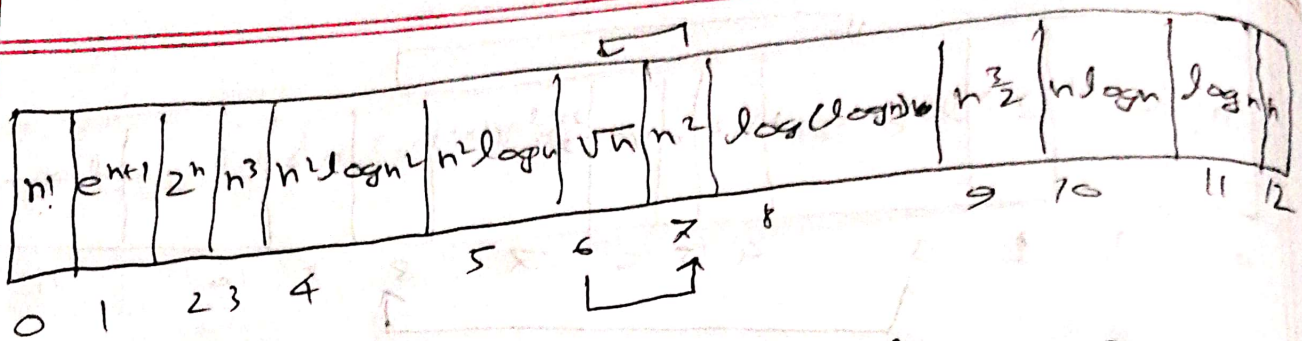
As n^3 is the largest so it will swap.



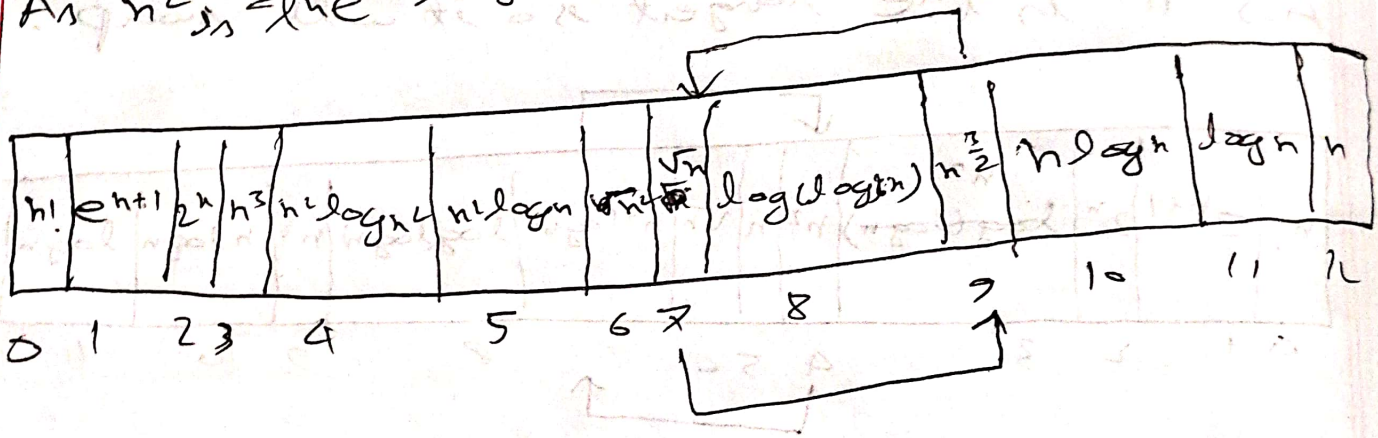
As $n^2\log n^2$ is the largest so it will swap.



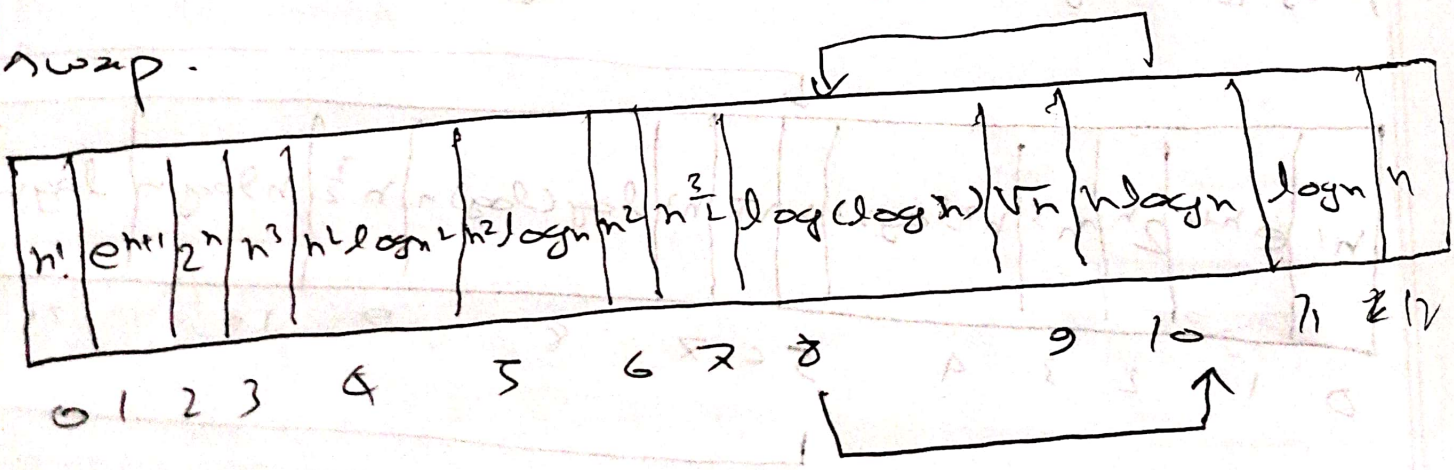
As the $n^2\log n$ is the largest so it will swap.



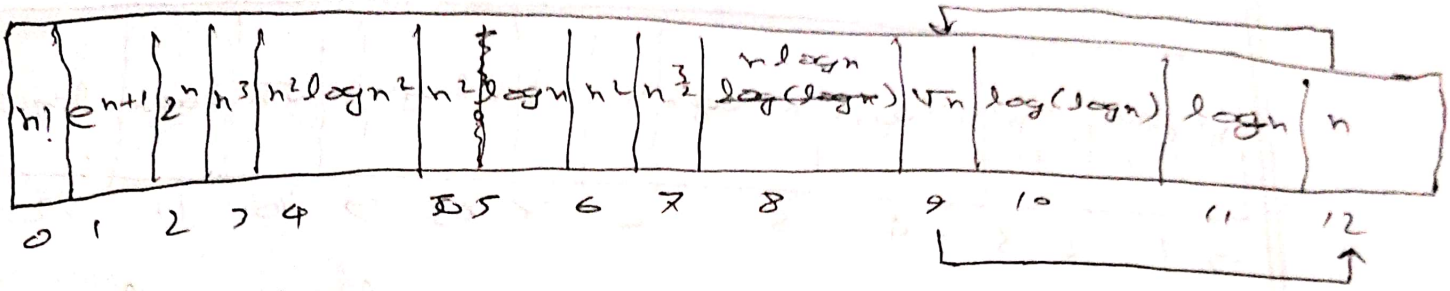
As n^2 is the largest so it will swap.



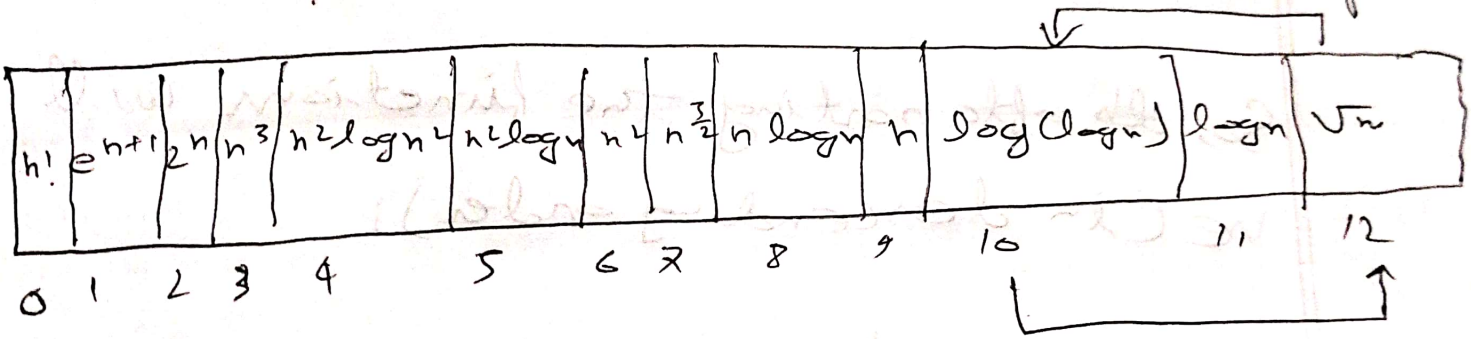
Now $n^{\frac{3}{2}}$ is the largest and it will swap.



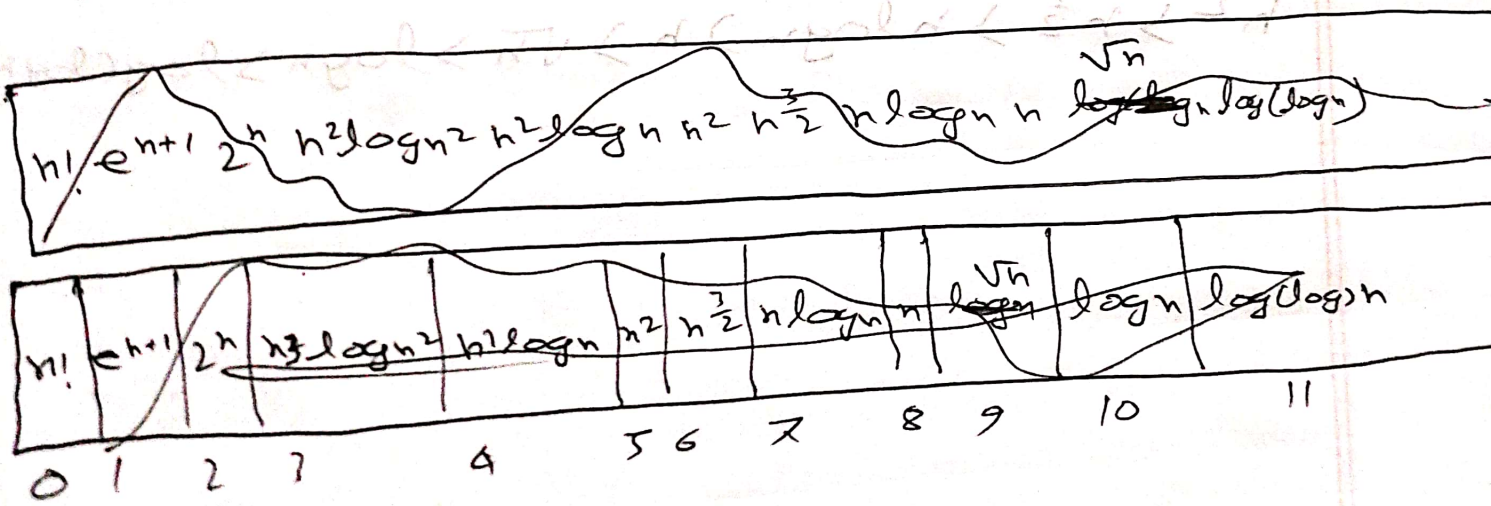
Now as $n \log n$ is the largest so it will swap.

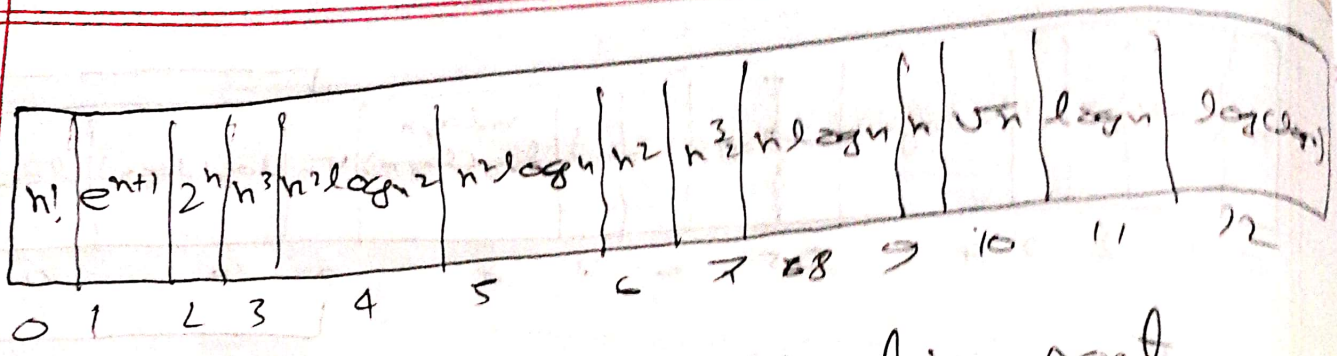


As here n is the largest so it will swap



As √n is the largest so it will swap.





So, Sorted array from selection sort.

So, after sorting the function will be (in descending order):

$$n! > e^{n+1} > 2^n > n^3 > n^2 \log n^2 > n^2 \log n > n^2 > n^{\frac{3}{2}} > n \log n > n > \sqrt{n} > \log n > \log(\log n)$$

1. b) Given,

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

$$\therefore T(n-1) = 2T(n-1-1) + n-1$$
$$= 2T(n-2) + n-1$$

$$\therefore T(n-2) = 2T(n-2-1) + n-2$$
$$= 2T(n-3) + n-2$$

So, $T(n) = 2T(n-1) + n$

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

$$= 2^2 T(n-2) + 2(n-1) + n$$

$$= 2^2\{2T(n-3) + n-2\} + 2(n-1) + n$$

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

$$T(n) = 2^n T(n-n) + 2^{n-1}(n-n+1) + 2^{n-2}(n-n+2) + \ldots$$
$$2(n-1) + n$$

$$= 2^n T(0) + 2^{n-1}(1) + 2^{n-2}(2) + \ldots\ldots\ldots + 2(n-1) + n$$

$$\leq 2^n + 2^n + 2^n + 2^n + \ldots\ldots \text{ [which is for } n \text{ times]}$$

$$\leq n 2^n$$

$$\therefore T(n) = \Theta(n 2^n)$$

1(c)(i) The time complexity recurrence relation for fibonacci number is,

$$T(n) = T(n-1) + T(n-2) + C$$

It can be written as,

$$T(n) \leq T(n-1) + T(n-1)$$

By telescoping ~~2nd method~~ we will get,

$$\Rightarrow T(n) \leq 2T(n-1)$$

As,
$T(n) \leq 2T(n-1)$
$\therefore T(n-1) \leq 2T(n-2)$
$\therefore T(n-2) \leq 2T(n-3)$

By telescopping method we will get,

$$\Rightarrow T(n) \leq 2\{2T(n-1-1)\}$$

$$\Rightarrow T(n) \leq 4T(n-2)$$

$$\Rightarrow T(n) \leq 2^2 T(n-2)$$

$$\Rightarrow T(n) \leq 2^2\{2T(n-3)\}$$

$$\Rightarrow T(n) \leq 2^2 . 2 . T(n-3)$$

$$\Rightarrow T(n) \leq 2^3 T(n-3)$$

For in case of n, :

$$T(n) \leq 2^n T(0)$$

$$\therefore T(n) \leq 2^n$$

So $T(n) = O(2^n)$

ii) Given,

$f(n) = 2n^3 + 5n^2 + 6n + 18$

From the function we can easily say that the higest dignee will be dominating and all other can be easily ignored from the expansion of function of polynomial function.

So, This function can be written as $2n^3$

Also discarding the 2 we will get,

$\theta(n^3)$

$\therefore 2n^3 + 5n^2 + 6n + 18 = \theta(n^3)$

iii) Given,

$$5+2\cos(n) = \theta(1)$$

$\therefore f(n) = 5+2\cos(n)$

As, $\cos(n)$ has range of from -1 to 1.

So we can write,

$$-1 \leq \cos(n) \leq 1$$

$$\Rightarrow -2 \leq 2\cos(x) \leq 2$$

$$\Rightarrow 5-2 \leq 5+2\cos(x) \leq 5+2$$

$$\Rightarrow 3 \leq 5+2\cos(x) \leq 7$$

So, the function is between 3 to 7.

As the range is constant so we can say that $5+2\cos(n) = \theta(1)$.

iv)

Given,

$$\frac{3}{2}n^2 + 2n - 3 = \Omega(n^2)$$

So,

$$f(n) = \frac{3}{2}n^2 + 2n - 3$$

An highest degree is the dominated one is other's can be discarded.

$\therefore \theta = (\frac{3}{2}n^2)$

Ignoring the $\frac{3}{2}$ we can say,

$\therefore \theta = (n^2)$

$\therefore f(n) = \theta(n^2)$ so we can say,

$\theta(n^2) = \Omega(n^2)$ as if tight bound have to true then upa and lower bauld also have to be become true.

$\therefore \frac{3}{2}n^2 + 2n + 3 = \Omega(n^2)$ [showed]

v) Given,

$n^3 + 5n \neq O(n^2)$

Let,

$f(n) = n^3 + 5n$

As the highest degree is the dom. is dominated one and others can be ignored so $f(n) = \Theta(n^3)$

Beider $f(n) = O(n^3)$ or it can be also said that $f(n) = n^3 + 5n = O(n^3)$.

$\therefore f(n) = n^3 + 5n \neq O(n^2)$ [Showed]

vi) Given,

$$T(n) = 2T\left(\frac{n}{2}\right) + n^3$$

~~From the master's~~ Comparing with the master's theorem we can say,

$$a = 2,\ b = 2,\ k = 3$$

As, $T(n) = O(n^k)$ if $b^k > a$

Here, $\because b^k > a$

$= 2^3 > 2$

$\therefore$ Time complexity will be $= O(n^k)$

$= O(n^3)$

$\therefore T(n) = 2T\left(\frac{n}{2}\right) + n^3 = O(n^3)$ [Showed]

vii) Given,

$$T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{3n}{5}\right) + n$$

Comparing this eqn with master's theorem we get,

$a = 1,\ b = \frac{5}{3},\ k = 1$

As we know,

$T(n) = O(n^k)$ if $b^k > a$

So, as,

$\left(\frac{5}{3}\right)^1 > 1$

$\therefore T_1 = O(n^k)$

$= O(n^1)$

Now,

$T(n) = T\left(\frac{n}{4}\right) + n$

Again from the master's theorem we get,

$a = 1,\ b = 4,\ k = 1$

Here, $4^1 > 1$ which is $b^k > a$

∴ Time complexity will be = $O(n)$

$\therefore T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{3n}{5}\right) + n$

$= O(n)$

Q.1) p = 0 ⟶ 1

for (i = 1; i <= n; i++) ⟶ n+1

while (p < i) ⟶ n

p = p + i ⟶ n

So total = 1 + n + 1 + n + n

= 3n + 2

∴ T(n) = 3n + 2 = n

∴ T(n) = θ(n) .

2) for (i = n; i >= 1; i = i/3)

p = 2

while (p < n)

Print ("hello")

p = p * p .

for the above for loop,

loop will go for

i.	
9	3
8	2
7	2
6	2
3	2
2	1

So the outer for loop will go for $(\log_2 n) + 1$

Now for while loop :-

p
2^1
2^2
2^4
2^8
⋮
2^{2^k}

A.S, $p < n$.

So, $n = 2^{2^k}$

A.S $n = 2^2$ the loop will go for $\log_2 2$

$n = 2^4$ the loop will go for $\log_2 4$.

So, while loop executes $\log_2 2^k$ or k times.

Now, $\log_2 n = 2^k \log_2 2$

$\therefore \log_2 n = 2^k$

$\Rightarrow \log_2(\log_2 n) = k \log_2 2 = k$.

So,

for the while total line will be,

$O(\log_2(\log_2 n))$

$\therefore$ Total $= \{(\log_3 n) + 1\}\{\log_2(\log_2 n)\}$.

$\therefore$ Time complexity $O(n) = (\log_3 n)(\log_2^{\log_2 n})$

3) p = 0

```
for i = 1, i < n, i = i * 2
        p++
for j = 1, j < p, j = j * 2
        print ("hello")
```

For the first for loop

1, 1, 2^{0+1}, 2^2, 2^3 so 2^k

Ans, $i = 2^k$

$\therefore 2^k >= n$

$\therefore 2^k = n$

$\therefore k = \log_2 n$

So, first loop will go for $\log_2 n$

The second will go for $\log p$. [As it goes with the same structure with the first loop]

$\log p$ can be also written as $\log_2(\log_2 n)$

$\therefore$ Total $= \log_2 n + \log_2(\log_2 n)$

$= \log_2 n$

$\therefore$ Time complexity $T(n) = \theta(\log_2 n)$

2
b)

```
Input size1
Input size2
Input arr1
Input arr2

function searching(element, array, size):
    start = 0
    stop = size
    point = size + 1
    x = 0
    while stop >= start:
        middle = (start + stop) // 2
        if array[middle] >= elenod:
            point = Take min of (middle, point)
            stop = middle - 1
        end of if
        else:
            start = middle + 1
    end of while
```

```
For i from point to size:
    if prime_check(arr1[i])
        x = x+1
    end of if
end of for
return x

function prime_check(ele):
    c=0
    for i from 1 to ele+1:
        if ele%i==0:
            c=c+1
        end of if
    end of for
    if c==2:
        return True
    else:
        return False

for a from 0 to size2:
    print(arr2[a], arr1, size1)
    print(searching(arr2[a], arr1, size1))
```