1) a) $T(n) = 16T\left(\frac{n}{4}\right) + n!$ → we can convert $n! \to \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$

For the regularity condition $16f(n/4) <= cf(n)$

$16(n/4)! <= cn!$ $c = 0.5$ satisfy the condition so according to the (case III) $T(n) = \Theta(n!)$

b) $T(n) = \sqrt{2}\,T\left(\frac{n}{4}\right) + \log n$

$T(n) = S(e^n) = \sqrt{2}S\left(\frac{e^n}{4}\right) + \log(e^n) = \sqrt{2}T\left(\frac{n}{4}\right) + n$

so $a = \sqrt{2}$ $b = 4$ $d = 1$

$T(n) = \Theta(n) \Leftarrow a < b^d$

c) $T(n) = 8T\left(\frac{n}{2}\right) + 4n^3$ $a = 8$ $b = 2$ $d = 3$

$a = b^d$ (case II)

⇓

$\Theta(n^3 \cdot \log n)$

d) $T(n) = 64T\left(\frac{n}{8}\right) - n^2/\log n$ → There is minus start of the f(n) function and this function is decreasing because of that we can not apply the master theorem there.

e) $T(n) = 3T\left(\frac{n}{3}\right) + \sqrt{n}$ $a = 3$ $b = 3$ $d = \frac{1}{2}$

$a > b^d$ (case III) → $\Theta(n^{\log_3 3}) = \Theta(n)$

f) $T(n) = 2^n T\left(\frac{n}{2}\right) - n^n$ we don't apply the master teorem because a is not constant.

g) $T(n) = 3T\left(\frac{n}{3}\right) + \frac{n}{\log n}$ $a = 3$ $b = 3$ Does not apply master teorem because non-polynomial difference between f(n) and $n^{\log_b a}$. $\frac{n}{\log n}$ is not polynomial.

CamScanner ile tarandı

2)

a) This is a case of Master theorem. so we can say $a = 9$ (subproblems) $b=3$ and $d=2$. As $a > b^d$ the running time is $a = b^d \Rightarrow T(n) = \theta(n^2 \log n)$

b) $T(n) = 8T(n/2) + n^3$ so we can solve this using master theorem

8 → subproblems, n^3 → $O(n^3)$

$a = 8 \quad b = 2 \quad d = 3$

$a = b^d \Rightarrow \theta(n^3 \log n) = T(n)$

c) $T(n) = \underbrace{2}_{\text{subproblem}} T\left(\underbrace{\frac{n}{4}}_{\text{quarter of the size}}\right) + \underbrace{\sqrt{n}}_{f(n)}$

so we can solve this using master theorem

$a = 2 \quad b = 4 \quad d = \frac{1}{2}$

$a = b^d \Rightarrow T(n) = \theta(n^{\frac{1}{2}} \log n)$

I would choose algorith C because it has the lowest order exponent so it should be fastest algorithm.

3) a)

i) [1, 9, 5, 13, 3, 11, 7, 15]

There is 2n-1 comparisons it is still $\theta(n)$ comparisons.

$C_{worst}(n) = 2C_{worst}(n/2) + n - 1 = \theta(n)$ Second array element is greater than is comparing First array element when we inserting last two array [1 5 9 13] | [3 7 11 15] so this is worst case for comparison.

ii) [3, 4, 5, 6, 7, 8, 9, 10]

If all the elements in the array is sorted the number of comparisons is n.

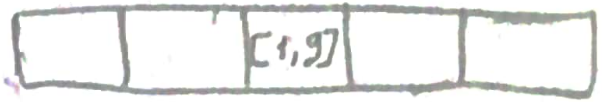
b) i) [1, 2, 3, 4, 5, 6, 7, 8] if array is sorted and pivot element is first element then there is maximum number of swap operations. because is swapping with the same number and it goes end of the array.

ii) [1, 2, 3, 4, 5, 6, 7, 8] if array is sorted there is no swap operation. because it was already sorted. and we choose the pivot last element because of this selecting worst case is happen but there is no swap operation.

CamScanner ile tarandı

1) algorithm (left, right)

```
mid = (left + right)/2 → Θ(1)
if A[mid] == 0 → Θ(1)
    return mid → Θ(1)
else
    if A[mid] > 0
        right = mid
        algorithm(right, left) → T(1)
    else
        left = mid
        algorithm(right, left) → T(2)
```



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

$\downarrow \quad \downarrow \quad \downarrow$

$T(n/2) + T(n/2) + 1 \Rightarrow T(n) = T(n/2) + 1$

we can solve this using master theorem

$a=1 \quad b=2 \quad d=0$ because $f(n) = +1 \cdot n^0 \Rightarrow d=0$

$a = b^d$ so $T(n) = \Theta(n^0 \log n) = \Theta(\log n) \Rightarrow$ Time complexity

5) Psudo-Code

```
def main():
    Box = [5,2,6,7,1] -> O(1)
    Gifts = [6,7,2,5,1] -> O(1)
    call quick-sort(0, len(Box), Box., Gifts) -> O(1)
    call matchBoxandGift(Box, gifts) -> O(1)
Function Quick-sort(x, len1, Box, Gifts) do
    if (x < len1) then -> O(1)
        int1 = Box[x]  end of if -> O(1)
    call index = partition(x, len1, Gifts, int1) -> O(1)
        int1 = Gifts[index] -> O(1)
    call partition = (x, len1, Box, int1) -> O(1)
    call quick-sort(x, len1-1, Box, gifts) -> T1(n)
    call quick-sort(x+1, len1, Box, Gifts) -> T2(n)
end of function
Function matchBoxandGift(Box, Gifts) do
    print(Box) -> O(1)
    print(Gifts) -> O(1)
    print("Box and Gifts are pairing") -> O(1)
end of function
Function partition(start, end, array, pivot_index)
    while start < end. do -> T3(n)
T4(n) <- while start < len(array) and array[start] <= pivot index. do
            repeat start += 1
        end of while
T5(n) <- while array[end-1] > pivot_index:
            repeat end -= 1
        end of while
        if start < end then
            array[start], array[end] = array[end], array[start]
    end of while
    array[end], array[start] = array[start], array[end]

    i = 0
T6(n) <- while i < len(array). do
            if array[i] == pivot_index
            pivot1 = i  end of if
        end of while
        i += 1
    return pivot1
end of function
```

Recurrence Relation:

$T(n) = (T_1(n) + T_2(n)) \cdot (T_3(n) + T_4(n) + T_5(n) + T_6(n))$

From quicksort: $T(n/2) + T(n/2) \rightarrow S_2(n)$

From partition function: $S_1(n)$

$S_1(n) = O(n) + O(n) + O(n) + O(n)$

$S_1(n) = O(n)$

$S_2(n) = T(n/2) \rightarrow \log(n)$

$S_2(n) = O(\log n)$

$T(n) = S_1(n) * S_2(n)$

$T(n) = O(n \log n)$ //

CamScanner ile tarandı