a) $T(n) = 16T\left(\frac{n}{4}\right) + n!$

$a = 16 \quad b = 4$

$n! = f(n) \notin \Theta(n^d)$

for any constant $d \geq 0$

$n!$ is not polynomial, master theorem cannot be applied.

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

$a = \sqrt{2} \geq 1$ ✓

$b = 4 \geq 2$

$f(n) = \log(n)$

f(n) is not polinomial.

Master theorem cannot be applied.

c) $T(n) = 8T(\frac{n}{2}) + 4n^3$

$a = 8$

$b = 2$

$f(n) = 4n^3 \in \Theta(n^3)$ $\quad d = 3 \geq 0$

$a = 8 \qquad b^d = 2^3 = 8$

$a = b^d \Rightarrow T(n) \in \Theta(n^d \log n)$

$T(n) \in \Theta(n^3 \log n)$

d) $T(n) = 64\,T\left(\frac{n}{8}\right) - n^2 \log n$

$a = 64 \quad b = 8$

$f(n) = -n^2 \log n$ ×

f(n) is polynomial but it is negative

T(n) is not eventually non-decreasing function, so master theorem cannot be applied.

e) $T(n) = 3T\left(\frac{n}{3}\right) + \sqrt{n}$

$a = 3 \quad b = 3$

$f(n) = \sqrt{n} = n^{1/2}$

$f(n) \in \Theta(n^{1/2}) \quad d = 1/2 \geq 0$

$b^d = 3^{1/2} = \sqrt{3} < a = 3$

$T(n) \in \Theta(n^{\log_b a})$

$T(n) \in \Theta(n^{\log_3 3})$

$\boxed{T(n) \in \Theta(n)}$

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

T(n) is not eventually non-decreasing function. Master theorem cannot be applied.

g) $T(n) = 3T(\frac{n}{3}) + \frac{n}{\log n}$

$a = 3 \quad b = 3$

$f(n) = \frac{n}{\log n}$ f(n) is not polynomial, Master theorem cannot be applied.

2) a) $X(n) = 9X(\frac{n}{3}) + f(n)$

$f(n) \in \Theta(n^2)$

$a = 9 \quad d = 2 \geq 0$

$b = 3 \quad \boxed{b^d = 9 = a}$

$X(n) \in \Theta(n^d \log n)$

$\boxed{X(n) \in \Theta(n^2 \log n)}$

b) $Y(n) = 8\,Y(\frac{n}{2}) + f(n)$

$f(n) \in O(n^3)$

$a = 8 \quad b = 2 \quad f(n) = n^3$

$\boxed{b^d = 8 = 8 = a} \quad d = 3$

$Y(n) \in O(n^d \log n)$

$\boxed{Y(n) \in O(n^3 \log n)}$

c) $Z(n) = 2\,Z(\frac{n}{4}) + f(n)$

$f(n) \in O(\sqrt{n})$

$f(n) \in O(n^{1/2})$

$a = 2 \quad d = 1/2$

$b = 4$

$\boxed{b^d = 4^{1/2} = 2 = a}$

$\boxed{Z(n) \in O(n^{1/2} \log n)}$

$O(n^3 \log n) > O(n^2 \log n) > O(n^{1/2} \log n)$

$Y(n) > X(n) > Z(n)$

I would choose algorithm z, because it has the lowest running time.

3) i) 1 3 5 7 2 4 6 8

Because it compares sorted subarrays, if all of the elements in one of the subarrays has no remaining element, it will put other sub array's remaining elements at the end of the created array. There will not be any comparison. This array avoids this situation because no sub array will come to the end before other one.

ii) 1 2 3 4 5 6 7 8

When merging the subarrays, first sub array will not have any element greater than any element in the second sub array. So the first element of the second sub array will compared with all of the elements of first array, and then, all elements of second sub array will be placed at the end of created array without comparison.

b) i) 2 4 6 8 7 5 3 1

Actually, I found this answer by guessing. It causes 7 swap operations. We know that in every iteration quicksort places one of the elements to the proper position. And 7 is the maximum number of swaps, because it means that only 1 swap operation placed both elements to the proper position. The other 6 swap operations placed only one of the elements, and we can not move placed elements. So 1 move, which places both elements to the proper position must be always there. 8−1=7 it is the maximum # swap operation.

ii) 1 2 3 4 5 6 7 8

Because the array is sorted, there will not be any swap operation.

4) Basic operation $= \Theta(1)$

Problem size divided by two every time $T(\frac{n}{2})$

1 sub problem is called

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

Master theorem: $a = 1$

$b = 2$

$n^0 = 1 = n^d \Rightarrow d = 0$

$\boxed{a = b^d}$

$1 = 2^0 = 1$

$$T(n) \in \Theta(n^d \log n)$$

$$T(n) \in \Theta(\log n)$$

```
gifts = [2,1,4,5,3]
boxes = [3,1 ,5,2,4]

function putAtTheStart(arr[0..n],pivotValue ,low,high)
        for i = low to high do
            if (arr[i]== pivotValue) then
                    swap(arr, i, low)
                    break
            end if
        end for
end

function partition(arr[0...n], low, high, pivotValue)
    left = low
    right = high-1
    while(left<right)
        while(left<right and arr[left]<= pivotValue)
                left++
        end while
        while (right>low and arr[right]> pivotValue)
                right--
        end while
        if(left < right)
                swap(arr, left, right)
        end if
    end while
    arr[low] = arr[right]
    arr[right] = pivotValue
    return right
end
```

```
function matchGiftBox(boxes, gifts, low, high)
    if (low >= high or low < 0 or high > len(boxes))
        return
    putAtTheStart(gifts, boxes[low], low, high)
    index = partition(gifts, low, high, boxes[low])
    partition(boxes, low, high, gifts[index])
    matchGiftBox(boxes, gifts, low, index)
    matchGiftBox(boxes, gifts, index+1, high)
```

Analyze: Partition $\Rightarrow \Theta(n)$

PutAtTheStart $\Rightarrow O(n)$

matchGiftBox $\Rightarrow O(n) + \Theta(n) + \Theta(n) + T(\frac{n}{2}) + T(\frac{n}{2})$

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

Master Theorem: $a = 2$ $b = 2$ $f(n) = n = n^1 = n^d$

$d = 1$

$a = 2$

$b^d = 2^1 = 2$

$a = b^d$

$T(n) \in \Theta(n^d \log n)$

$\boxed{T(n) \in \Theta(n \log n)}$