1) a) $T(n) = 5 * T(n/2) + O(n)$

Master theorem;

$a=5$

$b=2$

$d=1$

$d < \log_b a \Rightarrow 1 < \log_2 5 \Rightarrow \underline{O(n^{\log_2 5})}$

b) $T(n) = 2 * T(n-1) + O(1)$

$T(2) = 2 * T(1)$

$T(3) = 2 * T(2) = 2 * 2 * T(1)$

⋮

$T(n+1) = 2^n * T(n) \Rightarrow \underline{O(2^n)}$

c) $T(n) = 9 * T(n/3) + O(n^2)$

Master theorem;

$a=9$

$b=3$

$d=2$

$d = \log_b a \Rightarrow 2 = \log_3 9 \Rightarrow \underline{O(n^2 \log n)}$

$n^{\log_2 5} = n^{2,32}$, $n^2 \log n < n^2$ so $\underline{n^2 \log n < n^{\log_2 5}}$

$\lim_{n \to \infty} \frac{2^n}{n^2} = \infty$ so $2^n > n^2$ so;

$O(n^2 \log n) < n^{\log_2 5} < 2^n$

so we should choose algorithm C.

2) Master's theorem;

$T(n) = aT\left(\frac{n}{b}\right) + n^c$

where $c < \log_b a$ then $T(n) = \Theta(n^{\log_b a})$

where $c = \log_b a$ then $T(n) = \Theta(n^c \log n)$

where $c > \log_b a$ then $T(n) = \Theta(f(n))$

a) $T(n) = 2T(n/3) + 1$

$a=2, b=3, c=0$

$0 < \log_3 2 \Rightarrow T(n) = \Theta(n^{\log_3 2})$

b) $T(n) = 5T(n/4) + n$

$a=5, b=4, c=1$

$1 < \log_4 5 \Rightarrow T(n) = \Theta(n^{\log_4 5})$

c) $T(n) = 7T(n/7) + n$

$a=7, b=7, c=1$

$1 = \log_7 7 \Rightarrow T(n) = \Theta(n^1 \log n)$

d) $T(n) = 9T(n/3) + n^2$

$a=9, b=3, c=2$

$2 = \log_3 9 = \log_3 3^2 \Rightarrow T(n) = \Theta(n^2 \log n)$

e) $T(n) = 8T(n/2) + n^3$

$a=8, b=2, c=3$

$3 = \log_2 8 = \log_2 2^3 \Rightarrow T(n) = \Theta(n^3 \log n)$

f) $T(n) = 49T(n/25) + n^{3/2} \log n$

$a=49, b=25, c=3/2$ (n dominates log n)

$3/2 > \log_{25} 49 \cong 1,2 \Rightarrow T(n) = \Theta(n^{3/2} \log n)$

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

$T(n-1) = T(n-2) + 2 \Rightarrow T(n) = T(n-2) + 4$

⋮

$T(n) = T(n-k) + 2k$ if we assume $T(1) = 0$ and $k = n-1$

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

$T(n) = 2n - 1$ so $T(n) = \Theta(n)$

1) $T(n) = T(\sqrt{n}) + 1$

Lets suppose

$m = \log n$

$n = 2^m$

$T(2^m) = T(2^{m/2}) + 1$

Suppose

$S(m) = T(2^m)$

$S(m) = S(m/2) + 2$

Masters theorem

$C=0, a=1, b=2$

$0 = \log_2 1 \Rightarrow S(m) = \Theta(\log m)$

$T(2^m) = \Theta(\log m)$

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

3) We can do that by doing some modification on merge sort.

```
while (leftIndex < leftMax && rightIndex < rightMax)
{
    if(a[leftIndex] < a[rightIndex])
    {
      output[outputIndex] = a[leftIndex];
      ++leftIndex;
    }
    else if (a[leftIndex] > a[rightIndex])
    {
      output[outputIndex] = a[rightIndex];
      ++rightIndex;
    }
    else
    {
        // don't add duplicate item just increase index.
        ++rightIndex;
    }
}
```

4) If we can access the array only via comparisons that means the algorithm keeps compare until there is no possible item to compare in the array. Thus, we can consider the problem as desicion tree problem. In desicion trees, there must be one leaf foreach item on the array. Therefore, the tree must has $\log(n)$ height and the algorithm should take $\Omega(\log n)$ time on the best case.

5) Function f(n)

① if n<1 :
② println("still going")
③ f(n/2)
④ f(n/2)

Sub problem count : ③ and ④ lines so ⇒ 2

Sub problem size : n/2

How many work on one step = ② so ⇒ 1

Then :

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

$$a=2, b=2, c=0$$

$$0 < \log_2 2 \Rightarrow T(n) = \Theta(n^{\log_2 2}) = \Theta(n)$$