1a

Base case:

$n = 1$: the algorithm returns m, which is correct since $n \cdot m = 1 \cdot m = m$.

Inductive Hypothesis: Assume that for some $1 \leq k < n$ and for all m that the algorithm returns the product of n and m

Induction Step: We want to show that RPMult$(k+1, m)$ returns the product of $(k+1)(m)$

Case 1: if $(k+1)$ is even, then

$RPM(k+1, m) = RPM((k+1)/2, 2m)$

Since $k \leq n$ and $\frac{k+1}{2} \leq n$ from the I.H., the algorithm returns the product of $(k+1)(m)$.

Case 2: if $(k+1)$ is odd, then

$RPM(k+1, m) = m + RPM(k+1-1, m)$

which is equal to $m + RPM(k, m)$, which we assumed to be true from our I.H.

Thus, it returns the product of $(k+1)(m)$

We showed that the algorithm was correct when $n=1$. And we showed that if it is correct for inputs of $n>1$, then it is correct for inputs of $n+1$ for all m.

2a) RSS = RecursiveSelectionSort

$i=1$, $n=$ length of list

RSS (a[i], a[i+1], ..., a[n])

if $i == n$ //where i(ndex) is the starting element and //n is the size of the list

return list //base case

a[m] = minimum (a[i], a[i+1], ... a[n])

swap a[m] and a[i]

i++ //increase i by one

RSS (a[i], a[i+1], ..., a[n])

This algorithm receives a list. It then finds the smallest element in the list and places it in the first position of the list. Then, it calls itself (recursive call) and passes the list, but the variable i was incremented by 1. Thus the algorithm sorts the list from 2 to n, and so on until reaching $i=n$.

2b) **Base case:** if $i = n$ (trivially sorted)

When $i = n$, then there is only 1 element to sort in the list. Thus, 1 element is already sorted in the list, which is the correct output expected from the algorithm.

Inductive hypothesis: Assume that for some $k \geq 1$ and for any input list of length $k-1$ $RSS(a[i], a[i+1], \ldots, a[k-1])$ correctly sorts a list in increasing order.

Inductive step: We want to show that $RSS(a[i], a[i+1], \ldots, a[k])$ correctly sorts the list in increasing order.

$RSS(a[i], a[i+1], \ldots a[k])$ is equal by the algorithm statement to

$RSS(a[i+1], a[i+2], \ldots, a[k])$, which equals, by the inductive hypothesis, to the sorted list in increasing order from $a[i+1]$ to $a[k]$

We showed that the algorithm was correct when $i = n$. And we showed that if it is correct for lists of size $n \geq 1$, then it is correct for lists of size $n+1$.

2c Let $T(n)$ be the runtime of RSS for a list of size n

Recurrence $= T(n) = T(n-1) + cn$, $T(1) = c'$

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

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

(3) $= T(n-3) + cn + c(n-1) + c(n-2)$

$\vdots$

(k) $= T(n-k) + cn + c(n-1) + \ldots + c(n-k+1)$

(n-1) $= T(n-(n-1)) + cn + c(n-1) + \ldots + c(n-(n-1)+1)$

$= T(1) + cn + \ldots + c(2)$

$= c' + cn + \ldots + c(2)$

$= c' + c\sum_{i=2}^{n} i$

$= c' + c\left(\frac{n(n+1)}{2} - 1\right) \in \boxed{\Theta(n^2)}$

$\hookrightarrow \underbrace{T(n) = a * T(n/b) + g(n)}$ (Master Theorem)

3a Let T(n) be the runtime of the algorithm of input size of n.

Recurrence is $T(n) = 4T(n/3) + O(n)$

4 subproblems : $a = 4$

Each subproblem $= n/3$, $b = 3$

$d = 1$

Case 3:

Master Theorem:

$a > b^d$

$\boxed{T(n) \in O(n^{\log_3 4})}$, since $4 > 3^1$

$\approx O(n^{1.26})$

3b Let $T(n)$ be the runtime of the algorithm with input size n.

Recurrence is: $T(n) = 8T(n/5) + O(n^{1/2})$

8 subproblems : $a = 8$

Each subproblem $= n/5$, $b = 5$

non-recursive part: $O(n^{1/2})$, $d = 1/2$

Master Theorem:

case 3: $a > b^d$

$$\boxed{T(n) \in O(n^{\log_5 8})}$$, since $8 > 5^{1/2}$

$$\approx O(n^{1.29})$$

3c) Let $T(n)$ be the runtime of the algorithm with input size n.

① $T(n) = T(n-1) + T(n-1) + c$, $T(1) = c'$

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

②

$2[2T(n-2)] + c + c$

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

③ $= 2[4T(n-3)] + 2c + c$

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

⋮

(k) $= 2^k T(n-k) + kc$

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

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

$= c' 2^{n-1} + (n-1)c$

$\in \boxed{O(2^n)}$