1. **Algorithm 1:** It's a recurrent relation, each iteration splits it into 4 more subproblems of half the size and then combines them linearly:

$$T(n) = 4T(n/2) + n$$

so using the Master Theorem (pg 53 of Algorithms book), and a=4, b=2, d=1:

Master theorem[2] *If* $T(n) = aT(\lceil n/b \rceil) + O(n^d)$ *for some constants* $a > 0$, $b > 1$, *and* $d \geq 0$, *then*

$$T(n) = \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a\,. \end{cases}$$

we get:

$$O(n^{\log_{\frac{4}{2}}}).$$

Algorithm 2: Another recurrent relation. Splits into 2 sub problems of size n-1 and combines in constant time:

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

By examining, n=4:
T(4)=2T(3)+c
T(3)=2T(2)+c
T(2)=2T(1)+c
T(1)=2T(0)+c

Nesting them all:
T(4)=2(2(2(2T(0)+c)+c)+c)+c
Which can be rewritten as: 2^4*(T(0)+c) so we get:

$$O(2^n)$$

Algorithm 3: A lot like Algorithm 1, again we use the Master Theorem:

$$T(n) = 9T(n/3) + n^2$$
$$a = 9, b = 3, d = 2 \; and \; log9_3 = 2 \; so$$
$$O(n^2 logn)$$

Comparing all 3 O times, we can see that Algorithm 1 has the best performance time.

2. This looks like an ideal case for bucket-sort:

```
function bucketSort(array, n) is
  buckets ← new array of n empty lists
  for i = 0 to (length(array)-1) do
    insert array[i] into buckets[msbits(array[i], k)]
  for i = 0 to n - 1 do
    nextSort(buckets[i]);
  return the concatenation of buckets[0], ...., buckets[n-1]
```

which yields O(n) when M is really small, O(n+M) average/best case for larger values and O(n^2) for worst case. As we can see from the pseudocode above, we have a for loop that iterates n times to insert values into the buckets then we iterate M times to sort them into buckets so O(n+M)

3. a) Normal merge-sort complexity: O(nlogn)
   and given our n and k:
   O (n*k log k)

   b) Quicksort. Pick a pivot then partition the set into 2 parts, swap the ones on the right that are less than the pivot with the ones on the left that are larger than the pivot and iterate. Once done, pick another pivot and partition a set again. Divide and conquer!

4. Note: Open prices aren't accurate indicators of performance, after hours trading is often not an accurate reflection of true value and its rare for you to be able to sell at opening price so I picked closing price to do my testing on instead.

Results:

GOOG
from: 2015/09/22 to 2015/06/22
Buy Date: 2015/07/17
Sell Date: 2015/08/24
Profit: 90.87
****
FNMA
from: 2015/09/22 to 2015/06/22
Buy Date: 2015/07/22
Sell Date: 2015/08/21
Profit: 0.5
****
AAPL
from: 2015/09/22 to 2015/06/22
Buy Date: 2015/07/20
Sell Date: 2015/08/21
Profit: 28.95
****
[Finished in 0.1s]