PSEUDOCODE

```
A[n]
d←n
bool run ← true
while run is true
        run ← false
        for  x ← 0 to d-2
                if A[x] > A[x+1]
                        swap A[x] and A[x+1]
                        run ← true
        d ← d-1
```

```
swap A[x] and A[y]
        temp = A[x]
        A[x] = A[y]
        A[y] = temp
```

PROVING CORRECTNESS

Invariant:

The invariant is that any element below A[d] should be <= any element above or equal to A[d]

Initialization:

In the first loop(d = size) it brings the largest element to the end of the array. This means all indexes larger that d-1 is sorted

Maintenance:

The sub-array of A[d] through A[size-1] is sorted and if we keep decrementing d by one and pushing back all elements smaller then the new largest elements from the unsorted smaller sub-array (A[0] through A[d-1]) in order to place this new element in the respective place in the sorted array above A[d] then, we will be left with a sub-array of A[d] through A[size-1] where all elements above A[d] are larger then those below the invariant is maintained.

Termination:

The final iteration will yield A[0] through A[size-1] sorted which means all elements below one another is smaller than that element. The invariant is withheld.

BEST CASE

Already sorted is the best case. This will run

```
1    A[n]
1    d←n
1    bool run ← true
1    while run is true
1          run ← false
1          for  x ← 0 to d-2
d-1              if A[x] > A[x+1]
0                      swap A[x] and A[x+1]
0                      run ← true
```

```
1         d ← d-1


0 swap A[x] and A[y]
0         temp = A[x]
0         A[x] = A[y]
0         A[y] = temp
```

A total of d+6 lines will run. Since d is a factor of n, this program is Theta(n).

WORST CASE

A reverse sorted algorithm is the worst case.

```
1 A[n]
1 d←n
1 bool run ← true
d-1 while run is true
d -1      run ← false
d-1       for  x ← 0 to d-2
(d-1)+(d-2)+.. if A[x] > A[x+1]
(d-1)+(d-2)+.. swap A[x] and A[x+1]
(d-1)+(d-2)+.. run ← true
d-1       d ← d-1


(d-1)+(d-2)+…          swap A[x] and A[y]
(d-1)+(d-2)+...                  temp = A[x]
(d-1)+(d-2)+...                  A[x] = A[y]
(d-1)+(d-2)+...                  A[y] = temp
```

Since the highest factor here is (d-1) + (d-2) +… which equals n(n+1)/2= (n^2 + n)/2 which yields a Theta(n^2) as the worst case.

PROBLEM 2

a) Is 2^(2n) = ϴ(2^n), Ω(2^n), or O(2^n)?

no because 2^(n*2) = (2^n)^2 which is not O(2^n)

b) Is 2^(n+1) = ϴ(2^n), Ω(2^n), or O(2^n)?

Yes because 2^(n+1) = (2^n)*2 = (2^n)*c = O(2^n)

PROBLEM 3

10^12 * 60 *60 * 2 = operations in 2 hours = 7.2*10^15 = totalCalcs

a) 200 n^2 + 5n + 40000 = 7.2*10 ^15

divide both sides 200

n = quadratic formula

n= 5,999,999

b) n^3 + 3 = 7.2*10^15

n = cube root (totalCalcs – 3)

n = 193,097

c) 10n^2 + 50000 = totalCalcs

n = Squareroot (totalCalcs/10 – 5000)

n = 26,832,815

d) n log_2 (n) = totalCalcs

Really hard to solve without just pretending log_2(n) is close enough to 1 compared to n lets just guess it is like 50 because log_2 of totalCalcs is 52 so we know its gonna be less but pretty close

50n = totalCalcs = 1.44*10^14

pretty close as the log base 2 of this is 47.03\

e)

2^(2n) = totalCalcs

2^n = sqrt(totalCalcs)

n = log_2 (squrt(TotalCalcs))

n = 26.3