D) Recurrence Relation :-

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

$T(0) = 1$

$$T(n) = T(n-1) + 2n \quad \text{---(1)}$$

lets find $T(n-1)$ by substitution

$$T(n-1) = T(n-2) + 2(n-1) \quad \text{---(2)}$$

Now find $T(n-2)$

$$T(n-2) = T(n-3) + 2(n-2) = T(n-3) + 2n - 4 \quad \text{---(3)}$$

Now substitute (2) into (1)

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

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

Now substitute value of $T(n-2)$ into (3) Here

$= T(n-3) + 2n - 4 + 4n - 2$

$= T(n-3) + 6n - 6$

.
.
.

So we get

$T(n) = T(n-3) + 6n - 6$

⋮

We will keep doing this upto k times and lets assume that at k

$(n-k) = 0$

$n = k$ ——— (A)

So we have

$T(n) = T(n-3) + 6(n-1)$

$= T(n-3) + 2 \times 3(n-1)$

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

As we was doing upto k time

$= T(0) + 2k(n-1)$

$= 0 + 2k(n-1)$

from (A) we know that $\boxed{k = n}$

$= 2n(n-1)$

$= 2n^2 - 2$

$= 2(n^2 - 1)$

So this belong to $O(n^2)$

4(a) Best case - $O(n^2)$

Worst case - $O(n^2)$

Average - $O(n^2)$

(b) For all cases, the analysis shows that big O is always n^2 because whatever the values are, all the loops in given algorithm will be executed at every condition.

Lets Assume, in best case if array is sorted, still all loops will be executed and check the conditions for all elements. Similarly in worst and Average case complexity will remain (n^2).

E) For sort function sort();

Big O is = $O(n^2)$

As it is clear, the loops works as

$$\frac{n}{2}\left(O(n) + O(n)\right)$$

$$= \frac{n}{2}\left(n + n\right) = \frac{n}{2}\left(2n\right)$$

$$= \frac{2}{2}n^2$$

$$= n^2$$

Hence

Complexity for sort() = $O(n^2)$

3(a) Best case - $O(n^2)$
Worst case - $O(n^2)$
Average case - $O(n^2)$

b) For all cases the analysis is same that is $O(n^2)$ As defined above that all loops will work for a particular amount of time so their time complexity will also remains same for all cases

F)

Assignment :- 4

- mMultiplication

```
int I (int i, int J, int n ){
    return n* i +J;
}

int dotProduct ( const int A[], const int B[],
                 int i, int J, int n ){
    int t = 0;
    for (int K=0; K<n; K++){                    // n times
        t += A[ I[i,K,n)] * B[ I(K,J,n) ];
    }
    return t;
}

Void mMultiply ( const int A[], const B[], int C[],
                 int n ){
    for ( int i=0; i<n; i++){                   // n times
        for ( int J=0; J<n; J++){               // n times
            C[ I(i,J,n)] = dotProduct ( A, B, i, J, n );
        }
    }
}
```

So there are three nested loops and each loop executes n times
So

Time Complexity $= n \times n \times n + 1$

$= n^3 + 1$

So Big O is

$O(n^3)$