Time complexity for EOA algorithm when n == 1

```
int i;                                                              - 1 s.c.
float dist = 0;                                                     - 1 s.c.
    for(int i = 0; i < sizeA - 1; i++){                             - 15 * sizeA = 13sizeA s.c.
        dist += abs(P[A[i]].x - P[A[i + 1]].x) + abs(P[A[i]].y - P[A[i+1]].y);      - 15
    }
    dist += abs(P[A[sizeA -1]].x - P[A[0]].x) + abs(P[A[sizeA - 1]].y - P[A[0]].y); - 15 s.c.
    if(dist < bestDist){                                            - 1 + max(3sizeA,0) = 3sizeA + 1 s.c
        for(int i = 0; i < sizeA; i++){                             - sizeA * 3
            bestSet[i] = A[i];                                      - 3
        }
        bestDist =  dist;                                           - 1 s.c.
    }
```

Number of executions for both for loops = $\frac{(sizeA - 1) - 0}{1}$ + 1

= sizeA

Time complexity for algorithm when n == 1
= 1 + 1 + 15sizeA + 3sizeA + 1 + 1 + 15
= 16sizeA + 19

Time complexity for INNA Algorithm

Time complexity for farthest_point:

```
int farthest_point(int n, point2D *P){
  int max_dist = 0;                              - 1 s.c.
  int i, j, A;
  int dist;

  for(i=0; i < n-1; i++)                                    - (n - 1) * 11n = 11n^2 - 11n s.c.
    for(j=0; j < n;j++) {                                   - 11 * n = 11n
      dist = abs(P[i].x - P[j].x) + abs(P[i].y - P[j].y);   -8
      if (max_dist < dist){                                 - 1  + max(2,0) = 3
          A = i;                                            - 1
          max_dist = dist;                                  - 1
      }
    }
  return A;
}
```

Number of executions for inner loop = $\frac{(n-1)-0}{1} + 1 = n$

Number of executions for outer loop = $\frac{(n-2)-0}{1} + 1 = n - 1$

Time complexity = $11n^2 - 11n + 1$

Time complexity for nearest function:

```
int nearest(int n, point2D *P, int A, bool *Visited){
   int dist = 0, mindist = 0, V;                               - 2 s.c.
   for(int i = 0; i < n; i++){                                 - 15 * n  = 15n
      if(!Visited[i]){                                         - 2 + max(13 ,0) = 15
            dist = abs(P[A].x - P[i].x) + abs(P[A].y - P[i].y); - 8
            if(mindist == 0 || mindist > dist){                - 3 + max(2,0) = 5
               V = i;                                          - 1
               mindist = dist;                                 - 1       - 1 + 1 = 2
            }
      }
   }
   return V;
}
```

Number of executions for loop = $\frac{(n-1)-0}{1} + 1 = n$

Time complexity = 15n + 2

Time complexity for INNA with the given functions

Visited = new bool[n];	- 2 s.c.
for(i=0; i<n; i++){	- n * 2 = 2n s.c.
Visited[i] = false;	
}	

$$\text{Number of executions for loop} = \frac{(n-1)-0}{1} + 1 = n$$

// calculate the starting vertex A	
A = farthest_point(n,P);	- 1 + $11n^2$ - 11n - 1 = $11n^2$ - 11n s.c.
// add it to the path	
i=0;	- 1 s.c.
M[i]= A;	- 2 s.c.
// set it as visited	
Visited[A] = true;	-2 s.c.
for(i=1; i<n; i++) {	- n * (15n + 8) = $15n^2$ + 8n s.c.
B = nearest(n, P, A, Visited);	- 1 + 15n + 2 = 15n + 3
A = B;	- 1 - = 15n + 8
M[i] = A;	- 2
Visited[A]=true;	- 2
}	

$$\text{Number of executions for loop} = \frac{(n-1)-0}{1} + 1 = n$$

dist = 0;	- 1 s.c.
for (i=0; i < n-1; i++) {	- (n-1) * 14 = 15n - 15 s.c.
dist += abs(P[M[i]].x - P[M[i+1]].x) + abs(P[M[i]].y - P[M[i+1]].y);	- 15
}	
dist += abs(P[M[0]].x - P[M[n-1]].x) + abs(P[M[0]].y - P[M[n-1]].y);	- 15 s.c.

$$\text{Number of executions for loop} = \frac{(n-2)-0}{1} + 1 = n - 1$$

Time complexity for the full algorithm

= 2 + 2n + $11n^2$ - 11n + 1 + 2 + $15n^2$ + 8n + 15n - 15 + 15

= $26n^2$ + 14n + 5