**Batch: A1 Roll No.: 1911004**

**Experiment No. 1**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **Title: Implementation of selection sort/ Insertion sort** |

**Objective:** To analyse performance of sorting methods

**CO to be achieved:**

|  |  |
| --- | --- |
| Sr. No | Objective |
| CO 1 | Analyze the asymptotic running time and space complexity of algorithms. |
| CO 2 | Describe various algorithm design strategies to solve different problems and analyse Complexity. |
| CO 3 | Develop string matching techniques |
| CO 4 | Describe the classes P, NP, and NP-Complete |

**Books/ Journals/ Websites referred:**

1. **Ellis horowitz, Sarataj Sahni, S.Rajsekaran,” Fundamentals of computer algorithm”, University Press**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein,” Introduction to algortihtms”,2nd Edition ,MIT press/McGraw Hill,2001**
3. [**http://en.wikipedia.org/wiki/Insertion\_sort**](http://en.wikipedia.org/wiki/Insertion_sort)
4. [**http://www.sorting-algorithms.com/insertion-sort**](http://www.sorting-algorithms.com/insertion-sort)
5. [**http://www.princeton.edu/~achaney/tmve/wiki100k/docs/Insertion\_sort.html**](http://www.princeton.edu/~achaney/tmve/wiki100k/docs/Insertion_sort.html)
6. [**http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/Sorting/insertionSort.htm**](http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/Sorting/insertionSort.htm)
7. [**http://en.wikipedia.org/wiki/Selection\_sort**](http://en.wikipedia.org/wiki/Selection_sort)
8. [**http://www.sorting-algorithms.com/selection-sort**](http://www.sorting-algorithms.com/selection-sort)
9. [**http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/Sorting/selectionSort.htm**](http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/Sorting/selectionSort.htm)
10. **http://courses.cs.vt.edu/~csonline/Algorithms/Lessons/SelectionCardSort/selectioncardsort.html**

**Pre Lab/ Prior Concepts:**

Data structures, sorting techniques

**Historical Profile:**

There are various methods to sort the given list. As the size of input changes, the performance of these strategies tends to differ from each other. In such case, the priori analysis can helps the engineer to choose the best algorithm.

**New Concepts to be learned:**

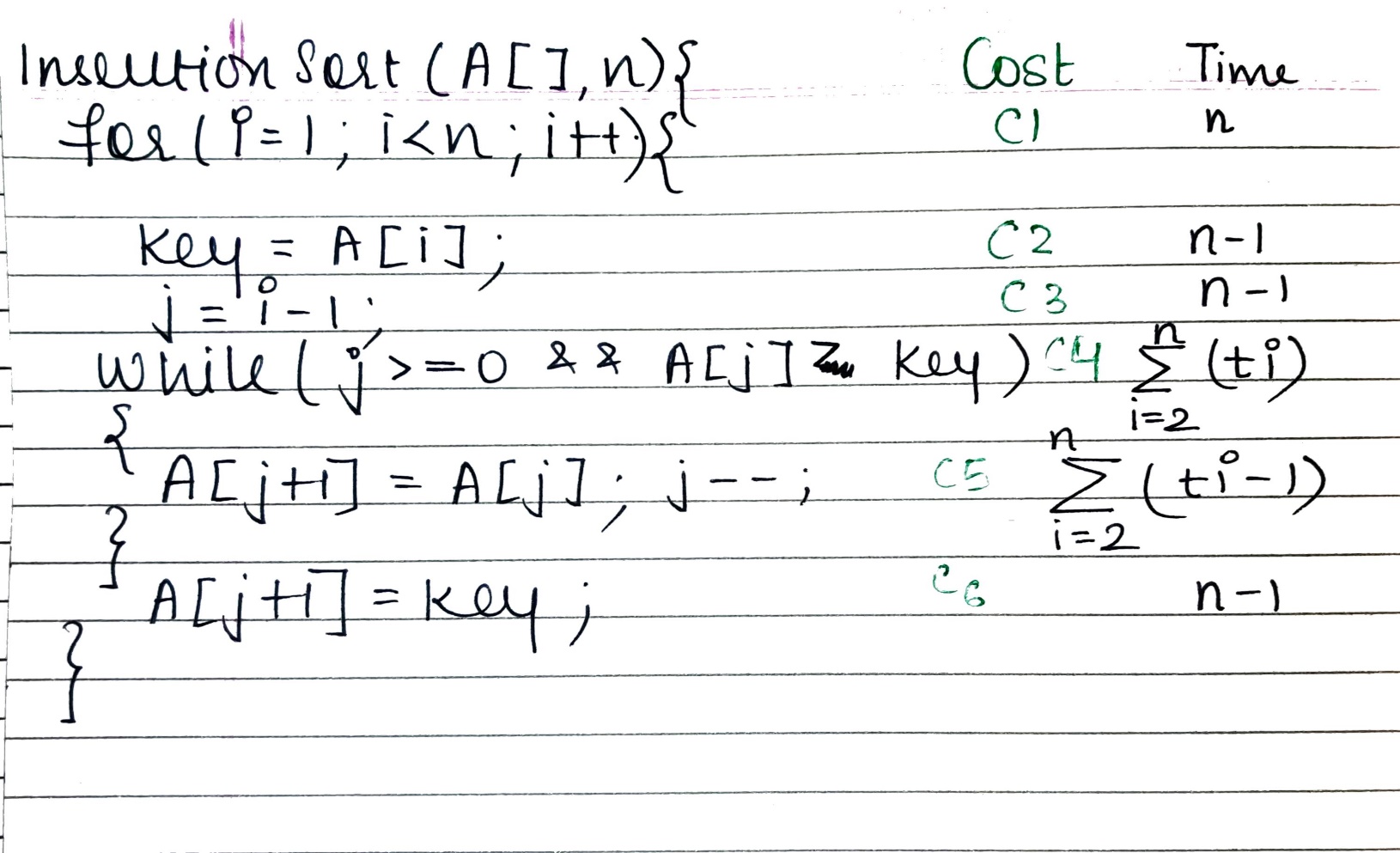
Space complexity, time complexity, size of input, order of growth.

**Algorithm InsertionSort**

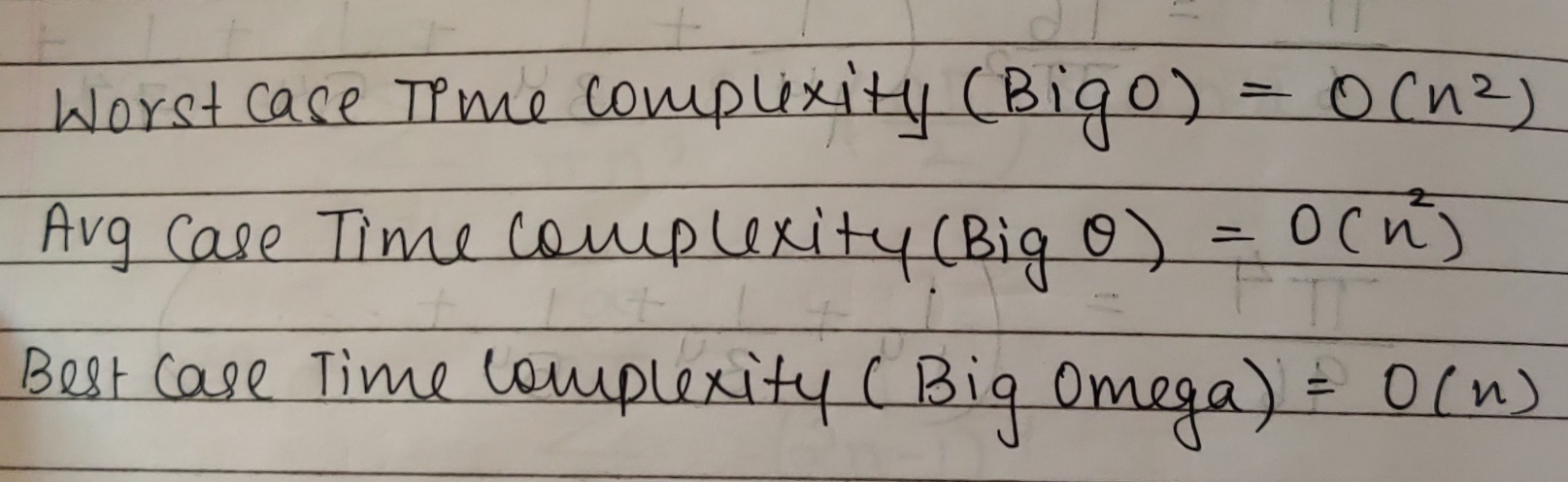
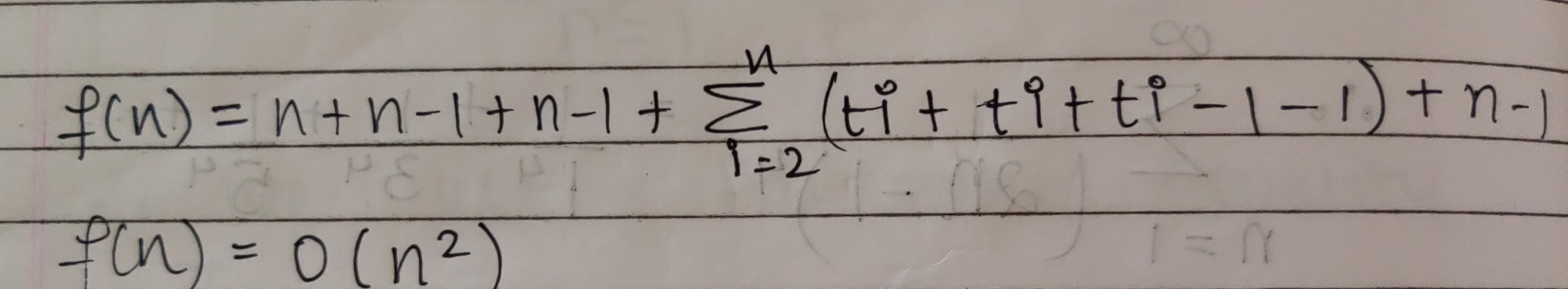
//The algorithm takes as parameters an array A[1.. n] and the length n of the array.

//The array A is sorted in place: the numbers are rearranged within the array

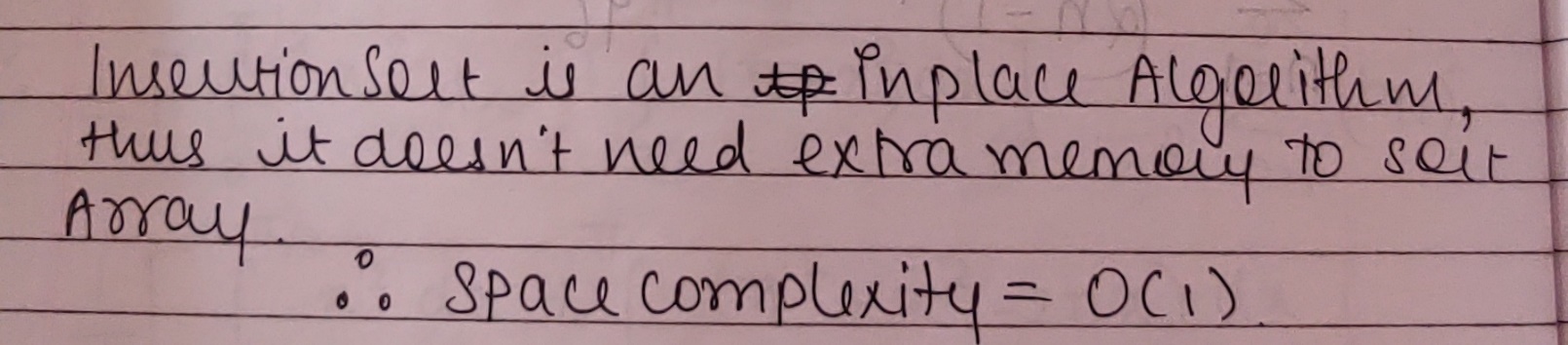
// A[1..n] of else type, n: integer



**Time complexity for Insertion sort:**



**The space complexity of Insertion sort:**

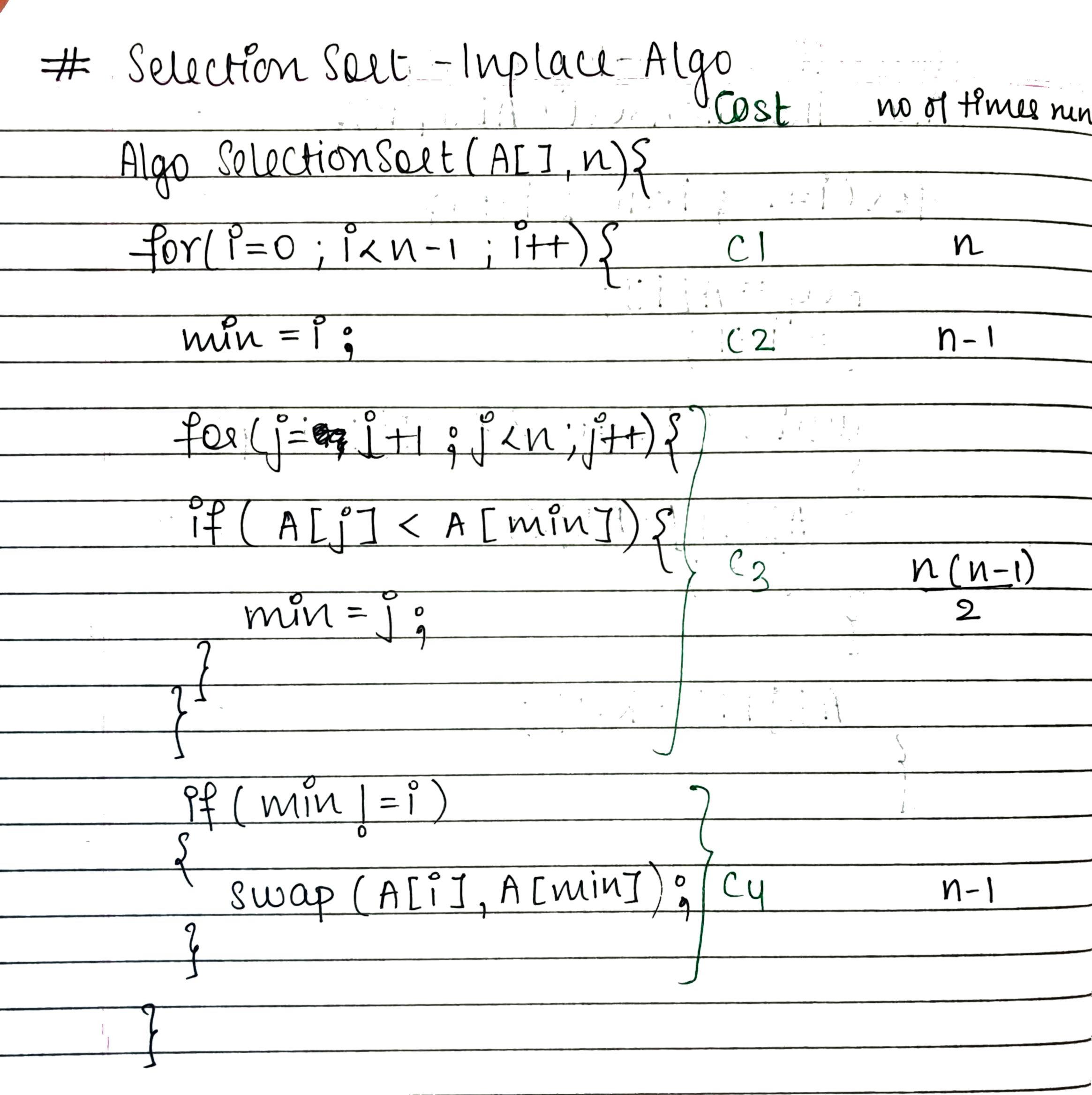


**Algorithm SelectionSort**

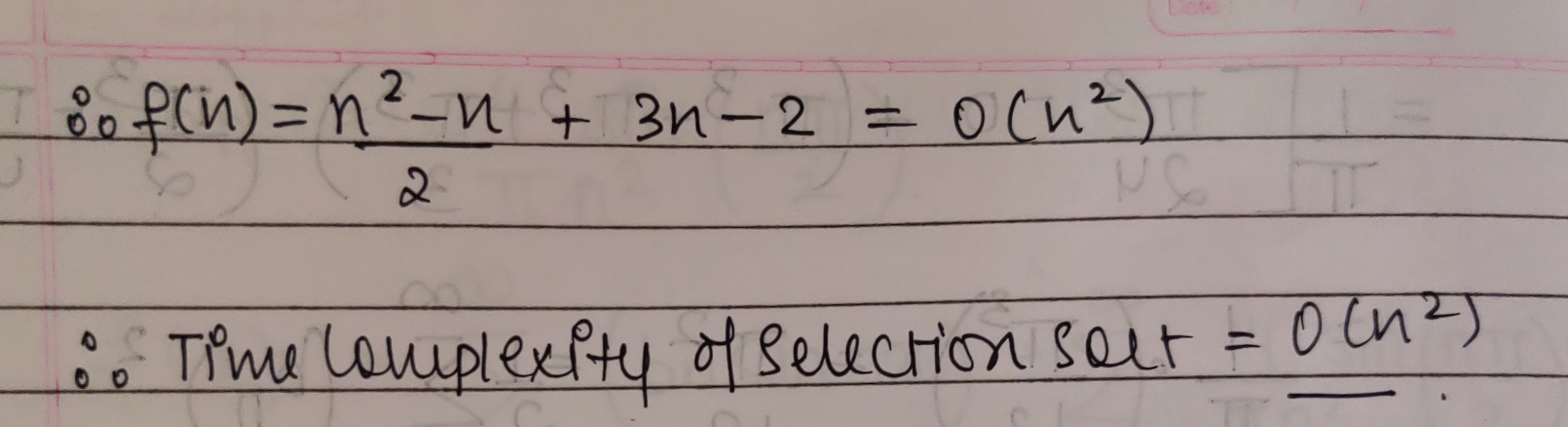
//The algorithm takes as parameters an array A[1.. n] and the length n of the array.

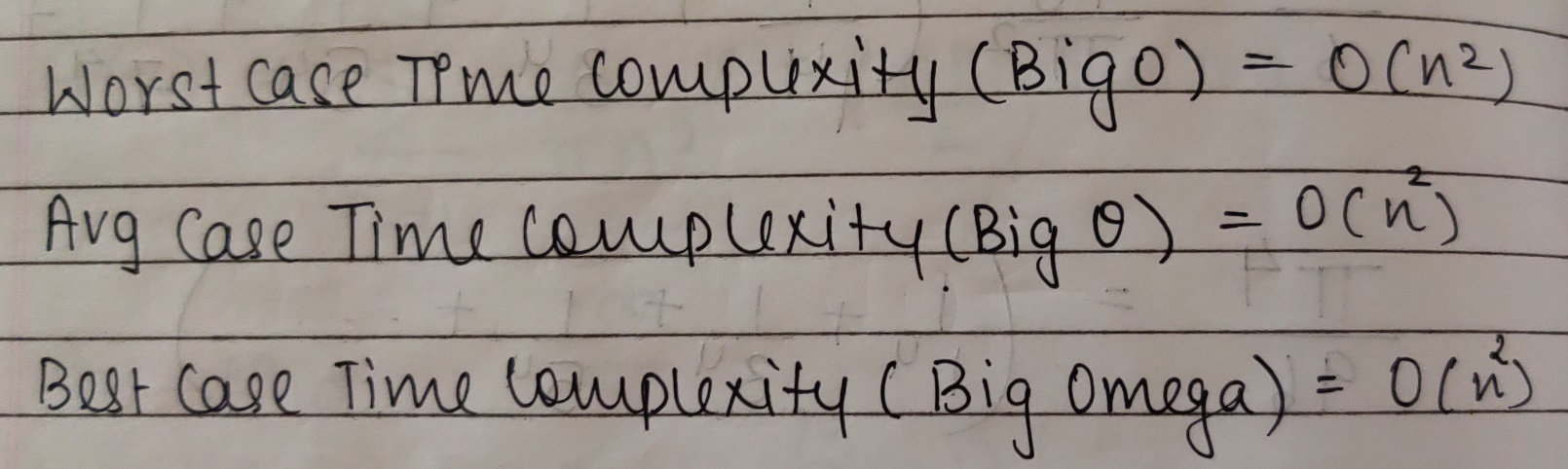
//The array A is sorted in place: the numbers are rearranged within the array

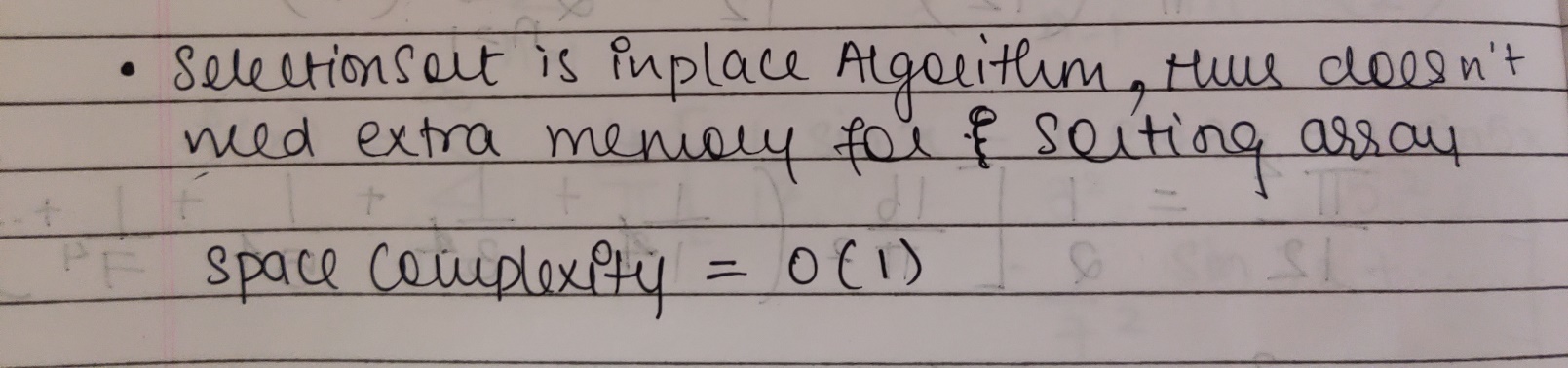
// A[1..n] of eletype, n: integer



**Time complexity for selection sort:**



**The space complexity of Selection sort:**

**Implementation details [Code + Output]**

**import java.util.\*;**

**class SelectionSort {**

**public static Random R;**

**public static void main(String[] args) {**

**Scanner ob = new Scanner(System.in);**

**System.out.println("****Selection Sort");**

**System.out.println("Enter the no of elements ");**

**int n = ob.nextInt();**

**int[] A = new int[n];**

**R = new Random();**

**for(int i = 0; i < n; i++)**

**A[i] = (R.nextInt(n)+1);**

**System.out.println("Unsorted Array");**

**ptr(A);**

**long start = System.currentTimeMillis();**

**selectionSort(A,A.length);**

**System.out.println("\nSorted Array");**

**ptr(A);**

**long end = System.currentTimeMillis();**

**System.out.println("\nTime Taken = "+(end - start)+" in millisec");**

**}**

**public static void selectionSort(int A[],int n){**

**int min;**

**for (int i = 0;i<n-1;i++){**

**min = i;**

**for(int j=i+1;j<n;j++)**

**if(A[j]<A[min])**

**min = j;**

**swap(A[min],A[i]);**

**System.out.println("\nPass "+(i+1)+"\t");**

**ptr(A);**

**}**

**}**

**public static void swap(int x,int y){**

**int t = x;**

**x = y;**

**y = t;**

**}**

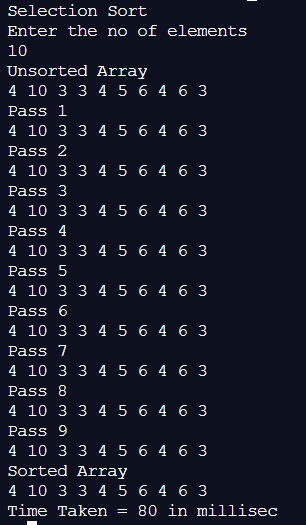
**public static void ptr(int A[]){**

**for(int i = 0; i < A.length; i++)**

**System.out.print(A[i]+" ");**

**}**

**}**



**import java.util.\*;**

**class InsertionSort {**

**public static Random R;**

**public static void main(String[] args) {**

**Scanner ob = new Scanner(System.in);**

**System.out.println("Insertion Sort");**

**System.out.println("Enter number of elements");**

**int n =ob.nextInt();**

**int[] A = new int[n];**

**R = new Random();**

**for(int i=0;i<n;i++)**

**A[i]=R.nextInt(n)+1;**

**System.out.println("Unsorted Array");**

**print(A);**

**long start = System.currentTimeMillis();**

**insertionSort(A,A.length);**

**System.out.println("\nSorted Array");**

**print(A);**

**long end = System.currentTimeMillis();**

**System.out.println("\nTime Taken = "+(end - start)+" in millisec");**

**}**

**public static void insertionSort(int[] A,int n){**

**int key,i,j;**

**for(i=1;i<n;i++){**

**key=A[i];**

**j=i-1;**

**while(j>=0 && key<A[j]){**

**A[j+1]=A[j];**

**j--;**

**}**

**A[j+1]=key;**

**System.out.println("\nPass "+i+"\t");**

**print(A);**

**}**

**}**

**public static void print(int A[]){**

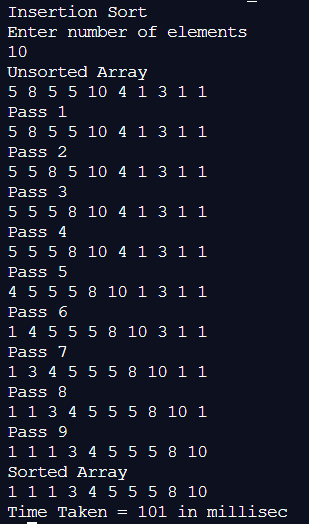
**for(int i=0;i<A.length;i++){**

**System.out.print(A[i]+" ");**

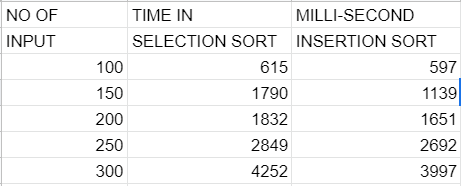
**}**

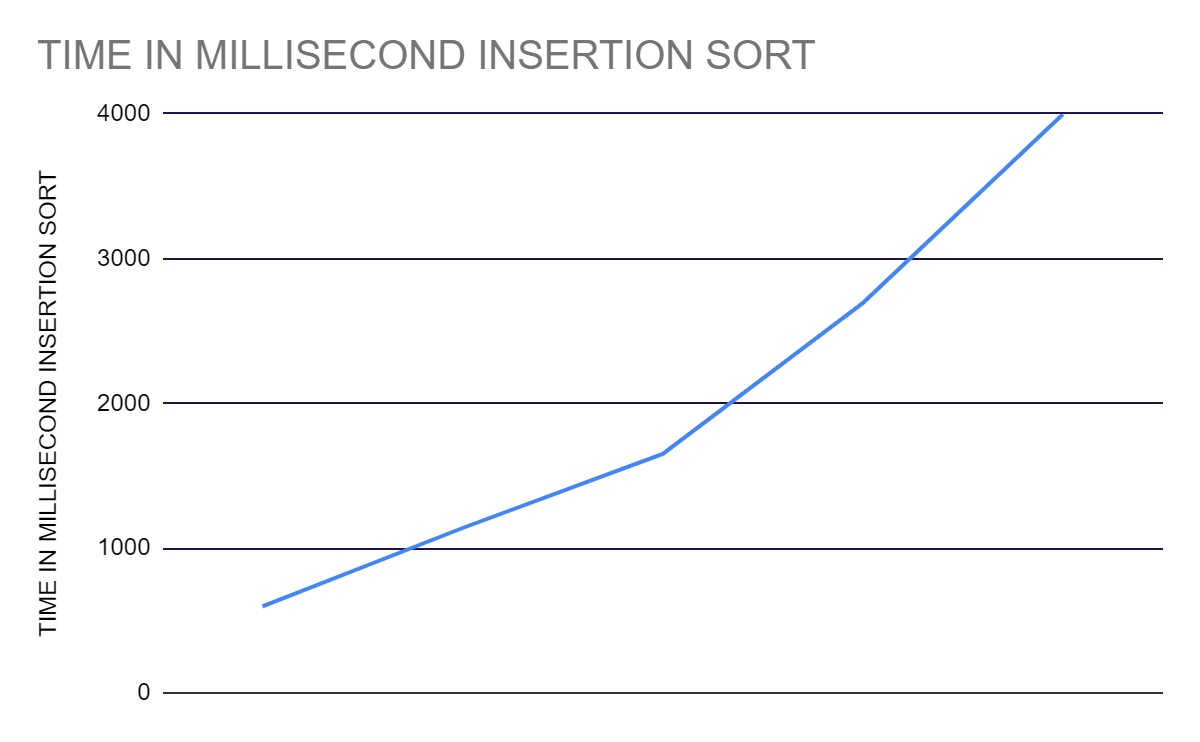
**}**

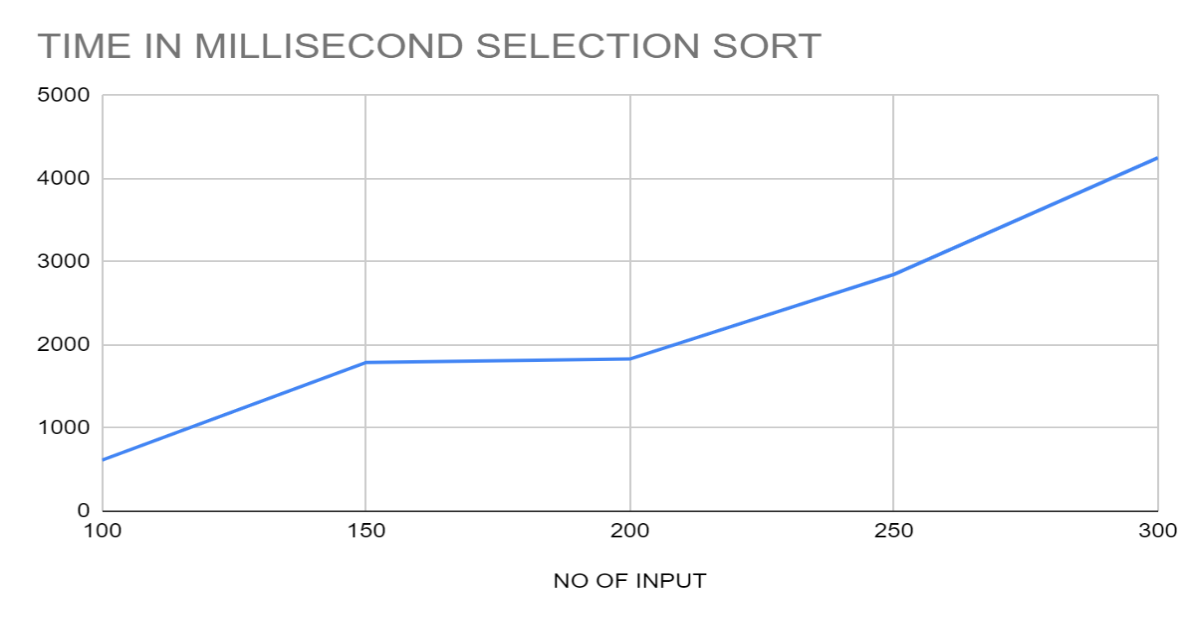
**}**

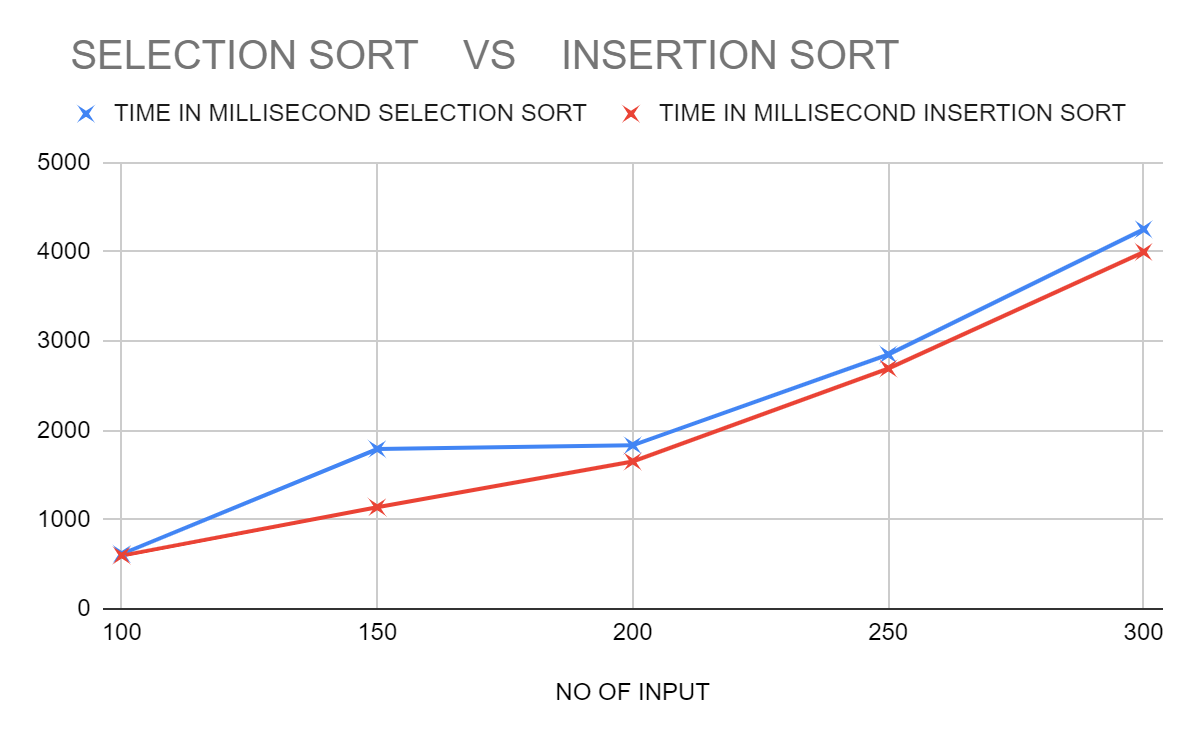


**Graphs for varying input sizes: (Insertion Sort & Selection sort)**



****

****

****

**CONCLUSION:**

From the above Graph, we can Analyse & conclude that Insertion Sort will perform Less no of Comparison than Selection Sort . Selection Sort will Compare to find MINIMUM ELEMENT in entire array & then Swap .But Insertion will Compare the key with SORTED array thus comparisons can be reduced & appropriate location can be found. Using this Concept Insertion Sort takes less Time for Sorting Array than Selection Sort. Thus, when the array is already sorted or almost sorted, insertion sort performs in O(n) time.

Thus when Sorting Array **Insertion Performs Better than Selection Sort**.