

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



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



**CO to be achieved:**

| CO 1 | Analyze the asymptotic running time and space complexity of algorithms. |
| --- | --- |



**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.



**Topic: Sorting Algorithms**

**Theory:** Given a function to compute on n inputs the divide-and-conquer strategy suggests splitting the inputs into k distinct subsets, 1< k ≤n, yielding k sub problems. These sub problems must be solved and then a method must be found to combine sub solutions into a solution of the whole. If the sub problems are still relatively large, then the divide-and-conquer strategy can possibly be reapplied. Often the sub problems resulting from a divide-and-conquer design are the same type as the original problem. For those cases the reapplication of the divide-and- conquer principle is naturally expressed by a recursive algorithm. Now smaller and smaller sub problems of the same kind are generated until eventually sub problems that are small enough to be solved without splitting are produced.

**Algorithm Insertion Sort**

INSERTION\_SORT (*A,n*)

//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

**FOR** j ← 2 **TO** length[*A*]   
             **DO**  key ← *A*[*j*]      
                   {Put *A*[*j*] into the sorted sequence *A*[1 . . *j* − 1]}     
                    *i* ← *j* − 1      
                    **WHILE** *i* > 0 and *A*[*i*] > key  
                                 **DO** *A*[*i* +1] ← *A*[*i*]              
                                         *i* ← *i* − 1       
                     *A*[*i* + 1] ← key

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

void insertionSort(int arr[], int n) {

int i, key, j;

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

key = arr[i];

j = i - 1;

while (j >= 0 && arr[j] > key) {

arr[j + 1] = arr[j];

j = j - 1;

}

arr[j + 1] = key;

}

}

int main() {

int n;

printf("Enter the size of the array: ");

scanf("%d", &n);

int arr[n];

// Generate 'n' random numbers

srand(time(NULL));

printf("Randomly generated array:\n");

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

arr[i] = rand() % 100; // Generate random numbers between 0 and 99

printf("%d ", arr[i]);

}

clock\_t start\_time = clock();

insertionSort(arr, n);

clock\_t end\_time = clock();

double insertionSortTime = ((double) (end\_time - start\_time)) / CLOCKS\_PER\_SEC;

printf("\nInsertion Sort:\n");

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

printf("%d ", arr[i]);

printf("\nTime taken: %f seconds\n", insertionSortTime);

return 0;

}

**Algorithm Selection Sort**

SELECTION\_SORT (A,n)

//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

**FOR** *i* ← 1 **TO** *n*-1 **DO**    
    min *j* ← *i*;  
    min *x* ← A[*i*]  
   **FOR** *j* ← *i* + 1 to n do  
        **IF** A[*j*] < min x then  
            min *j* ← *j*  
            min *x* ← A[j]  
    A[min *j*] ← A [*i*]  
    A[*i*] ← min *x*

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

void selectionSort(int arr[], int n) {

int i, j, min\_idx;

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

min\_idx = i;

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

if (arr[j] < arr[min\_idx])

min\_idx = j;

}

int temp = arr[min\_idx];

arr[min\_idx] = arr[i];

arr[i] = temp;

}

}

int main() {

int n;

printf("Enter the size of the array: ");

scanf("%d", &n);

int arr[n];

// Generate 'n' random numbers

srand(time(NULL));

printf("Randomly generated array:\n");

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

arr[i] = rand() % 100; // Generate random numbers between 0 and 99

printf("%d ", arr[i]);

}

clock\_t start\_time = clock();

selectionSort(arr, n);

clock\_t end\_time = clock();

double selectionSortTime = ((double) (end\_time - start\_time)) / CLOCKS\_PER\_SEC;

printf("\nSelection Sort:\n");

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

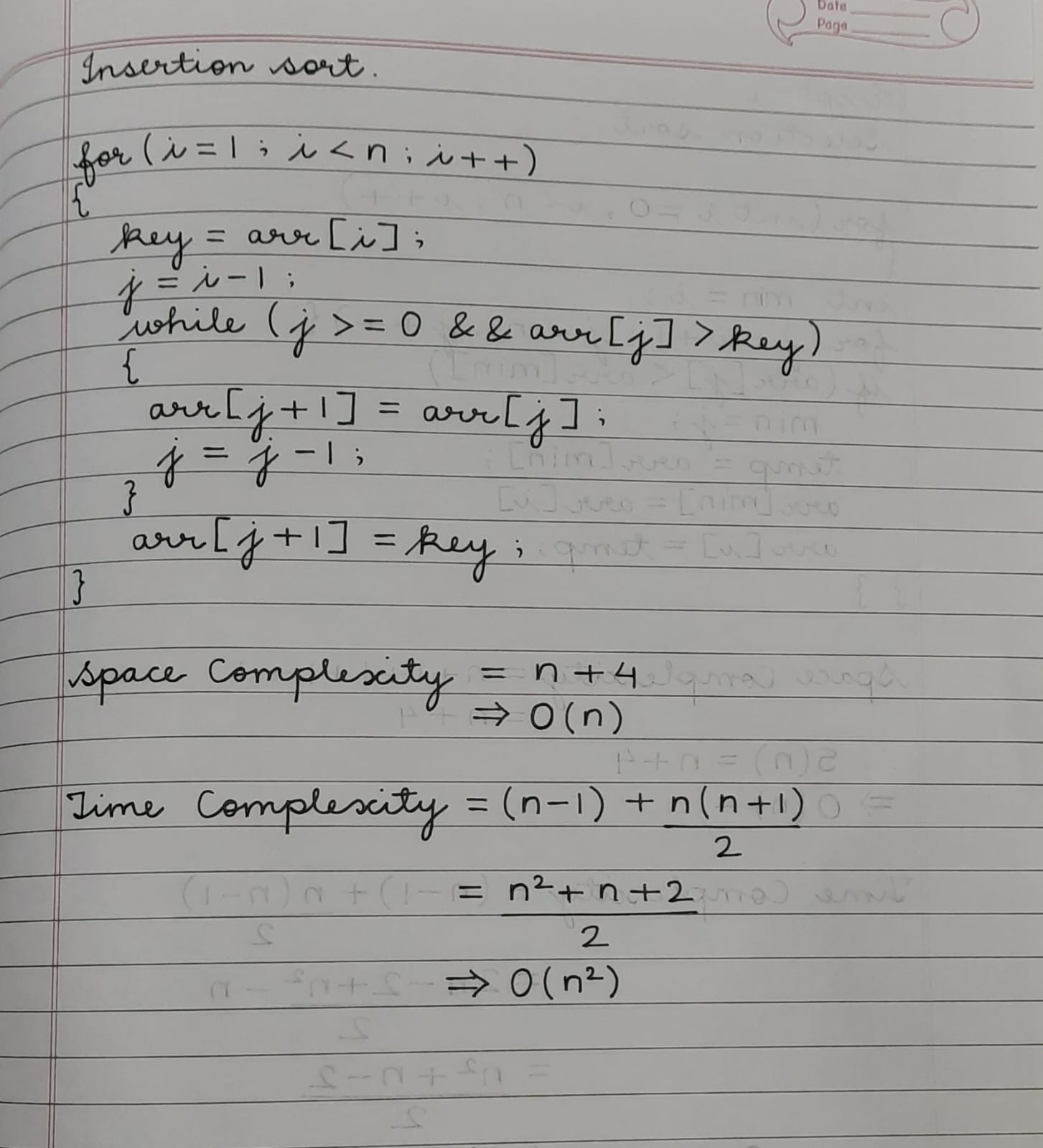
printf("%d ", arr[i]);

printf("\nTime taken: %f seconds\n", selectionSortTime);

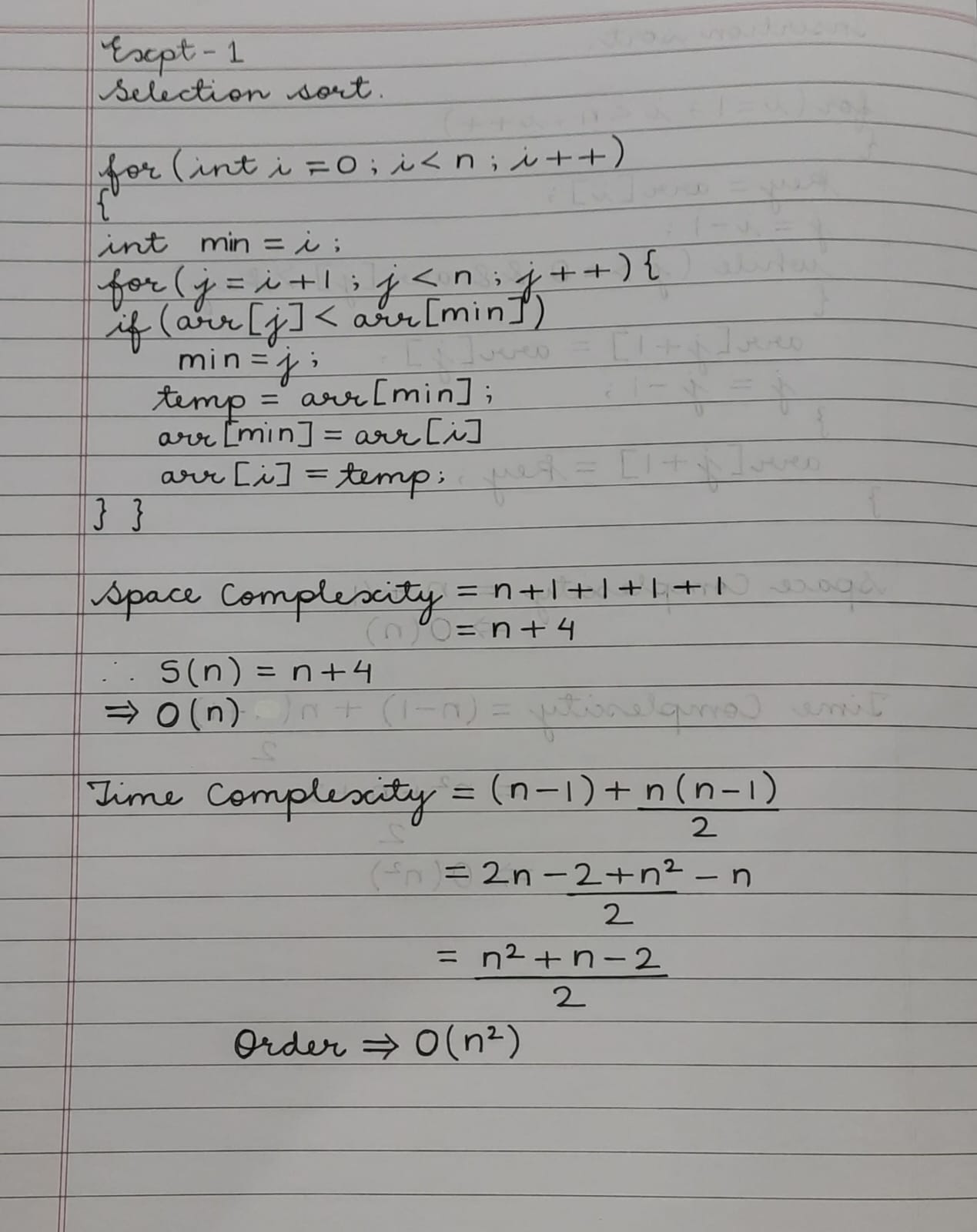
return 0;

}

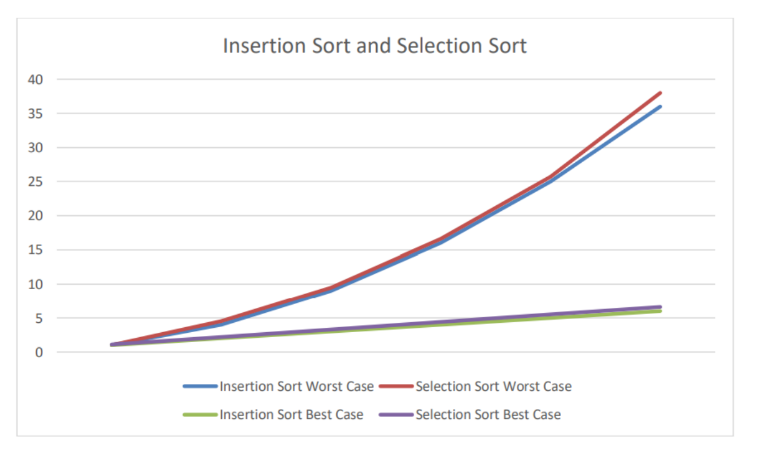
**The space and time complexity of Insertion sort:**

****

**The space and time complexity of Selection sort:**

****

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



**CONCLUSION:**

I have understood the logic behind insertion sort and selection sort and the analysis of their space and time complexities.