# EXPERIMENT 1-B

**NAME : ROHIT SHEKHAR BAVISKAR**
**UID : 2021700004**
**BATCH : CSE DS D1**
**AIM :** Experiment on finding the running time of an insertion sort and selection sort algorithm.

## ALGORITHM :

Selection sort is a simple sorting algorithm that works by repeatedly selecting the minimum element from the unsorted portion of the list and swapping it with the first element of the unsorted portion. The process is repeated until the entire list is sorted. Insertion sort is another simple sorting algorithm that works by maintaining a sorted portion of the list and inserting each subsequent element in the correct position in the sorted portion. The process is repeated until the entire list is sorted. Both selection sort and insertion sort have a time complexity of O(n^2) in the worst-case scenario, making them inefficient for large lists. However, they are simple to understand and implement, making them useful for small lists or as building blocks for more efficient algorithms.

**Selection sort algorithm:**
1. Start with an unsorted list of elements.
2. Select the minimum element from the list.
3. Swap the minimum element with the first element of the list.
4. Repeat steps 2 and 3 for the remaining unsorted portion of the list.
5. Repeat steps 2 to 4 until the entire list is sorted.

**Insertion sort algorithm:**
1. Start with an unsorted list of elements.
2. Consider the first element as a sorted portion of the list.
3. For each subsequent element, compare it with the elements in the sorted portion of the list.
4. If the element is smaller than any of the elements in the sorted portion, insert it in the correct position.
5. Repeat steps 3 and 4 for each element in the unsorted portion of the list.
6. Repeat steps 3 to 5 until the entire list is sorted

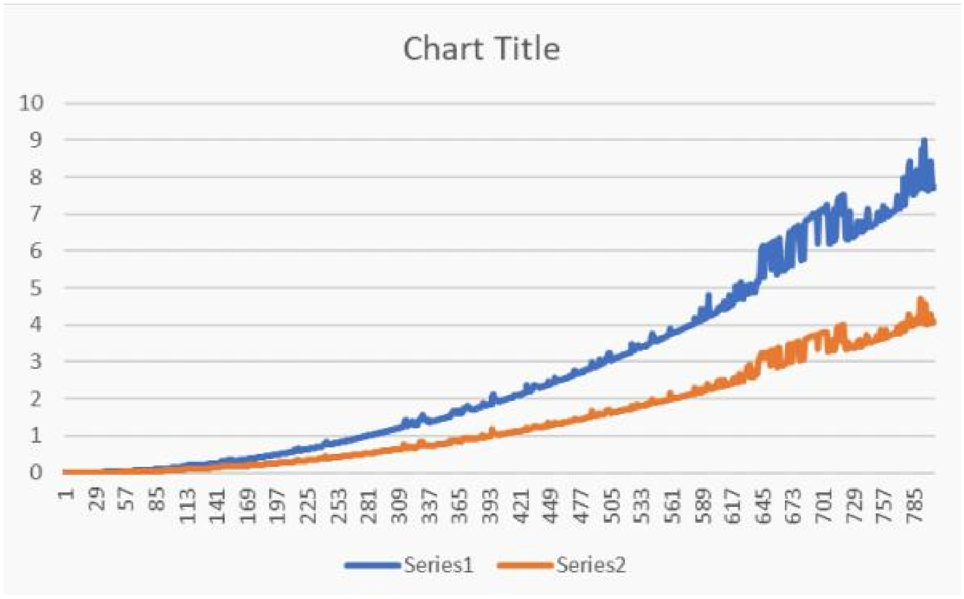
## PROGRAM :

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
void selection_sort(int arr[], int size)
{

   for (int i = 0; i < size - 1; i++)
   {
      int min_idx = i;
      for (int j = i + 1; j < size; j++)
      {
         if (arr[j] < arr[min_idx])
         {
            min_idx = i;
         }
      }
      int temp = arr[min_idx];
      arr[min_idx] = arr[i];
      arr[i] = temp;
   }
}

void insertion_sort(int arr[], int size)
{

   int i, key, j;
   for (i = 1; i < size; 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(void)
{
   clock_t a, b, c;
   int arr[100000], arr2[100000];
   double d, e;
   FILE *fptr;
   int num;

   fptr = fopen("integers", "w");
```

```
if (fptr != NULL)
{
   printf("File created successfully!\n");
}
else
{
   printf("Failed to create the file.\n");

   return -1;
}

int i = 0;
while (i != 100000)
{
   num = rand() % 100000;
   if (num != -1)
   {
      putw(num, fptr);
   }
   else
   {
      break;
   }
   ++i;
}

fclose(fptr);

fptr = fopen("integers", "r");

i = 0;
while ((num = getw(fptr)) != EOF)
{
   arr[i] = num;
   arr2[i] = num;
   i++;
}

printf("\nEnd of file.\n");

fclose(fptr);

for (int j = 100; j <= 100000; j += 100)
{
   a = clock();
   selection_sort(arr, j);
   b = clock();

   d = (double)(b - a) / CLOCKS_PER_SEC;
   printf("%f\n", d);
   c = clock();
```

```
        printf("\t");
        insertion_sort(arr2, j);
        e = (double)(c - b) / CLOCKS_PER_SEC;
        printf("%f\n", e);
    }
    d = (double)(b - a) / CLOCKS_PER_SEC;
    printf("%f", d);

    return 0;
}
```

## GRAPH :



## OBSERVATION / CONCLUSION :

Here we have plotted graphs of time complexities of insertion sort and selection Sort. Through this graph it is quite evident that insertion sort is better than selection sort,as there is a vast difference between the execution time of both the Algorithms. In insertion sort it inserts the value in the presorted array to sort the set of values in the array,whereas, in selection sort it finds the minimum number from the list and sort it in ascending / descending order.