|  |  |
| --- | --- |
| **NAME:** | Pratiksha Patil |
| **UID:** | 2022301016 |
| **SUBJECT** | Design Analysis of Algorithm |
| **EXPERIMENT NO :** | 1B |
| **AIM:** | Experiment on finding the running time of an algorithm. |
| **OBJECTIVE:** | To find out running time of 2 sorting algorithms like Selection sort and Insertion sort. |
| **THEORY** | **Insertion Sort** :  is a simple sorting algorithm that works similar to the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.  Lightbox  **Selection sort:-** is a simple and efficient sorting algorithm that works by repeatedly selecting the smallest (or largest) element from the unsorted portion of the list and moving it to the sorted portion of the list. The algorithm repeatedly selects the smallest (or largest) element from the unsorted portion of the list and swaps it with the first element of the unsorted portion |

|  |  |
| --- | --- |
|  | What Is Selection Sort Algorithm In Data Structures? | Simplilearn |
| **ALGORITHM** | step 1: start  Step2: call rand\_num() function  Step 2: create rand\_num file and store the random numbers in it.  Step3: open rand\_num file in reading mode Step 4: Store all random numbers in an array  Step5: Traverse all elements using for loop take n as 100  Step6: Perform insertion and selection sort on each block of 100 numbers  Step7: Calculate time required to perform insertion and selection sort at each iteration  Step8: Increment n by 100  Step 9: If n reaches 1000 then end else go to step 6 rand\_num() function:  step 1: start  step 2: create the file pointer  step 3: open the file in writing mode step 3: starts the loop from 0 to 100000  step 4: insert the 100000 random numbers in the file step 5: close the file handle  step 6: end Insertion sort:  Step 1: start  Step 2: start the loop from 1 to n |

|  |  |
| --- | --- |
|  | Step 3: initialize j with i-1  Step 4: current element is array(i) Step 5: if array(key)>0 and j>=0 Repeat below steps 6,7  Step 6: j+1th element will jth element Step 7: decrement j  Step 8: array(j+1) = current. Step 9: end.  selection sort:  step 1: start  step 2: start the loop  step 3: initialize the min element step 4: start the loop from i+1 to n step 5: check the condition:  if jth element less than min element then minimum element will be j.  step 6: if minimum element not equal to i, then initialize variable t with array(i) perform ith element = array of min array(min) = t  step 7: end. |

|  |  |
| --- | --- |
| **PROGRAM:** | H:\Screenshot (139).png |

|  |  |
| --- | --- |
|  | H:\Screenshot (140).png  H:\Screenshot (141).png |

|  |
| --- |
| **RESULT ( SNAPSHOT):**  H:\Screenshot (142).png  **Observation:** As we can see from the above graph time taken for insertion sort is always ranging whereas as time taken by selection sort is increasing as we keep adding 100 numbers to it. Hence we can say that insertion sort is better than selection sort. Both insertion sort and selection sort have a space complexity O(1). |