|  |  |
| --- | --- |
| **NAME:** | Gitanjali Gangurde |
| **UID:** | 2021300034 |
| **CLASS:** | SE Comps-A |
| **EXPERIMENT NO:** | 1-b |

**AIM**: Finding the running time of insertion sort and selection sort algorithms.

**ALGORITHM:**

# Insertion sort

* Iterate from arr[0] to arr[size-1] over the array.
* Compare the current element (key) to its predecessor.
* If the key element is smaller than its predecessor, compare it to the elements before. Move the greater elements one position up to make space for the swapped element.

# Selection sort

* Run a for loop using i variable to traverse through the array.
* Assign i to min .
* Again run a for loop using j variable to traverse through the array.
* This time while traversing if the element is smaller than the element at min location(a[min]) then assign min the value of that new index( j).
* After the second for loop ends, swap the elements a[i] and a[min]

# CODE:

**Generating 100000 numbers:**

#include <stdio.h> #include <stdlib.h>

int main()

{

int num,offset=0; int i,j;

FILE \*fptr; fptr=fopen("numbers.txt","w");

for(i=0; i<1000; i++)

{

for (j= 0; j<100; j++)

{

num = rand() % 100 + offset; fprintf(fptr," %d ",num);

}

offset+=100;

}

fclose(fptr); return 0;

}

# Sorting algorithms:

#include <stdio.h> #include <stdlib.h> #include <time.h>

void selectionSort(int a[], int len)

{

int min,temp; int i,j,key;

for(i=0; i<len-1; i++)

{

min=i;

for(j=i+1; j<len; j++)

{

if(a[j]<a[min])

{

min=j;

}

}

temp=a[i]; a[i]=a[min]; a[min]=temp;

}

}

void insertionSort(int arr[],int size){ int i;

int j; int key;

for(int i=0;i<size;i++){ key=arr[i];

j=i-1; while(j>=0&&arr[j]>key){

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

}

arr[j+1]=key;

}

}

int main(){

FILE \*file = fopen("numbers.txt", "r"); int integers[100000];

int i=0; int num;

while(fscanf(file, "%d\t", &num) > 0) { integers[i] = num;

i++;

}

fclose(file);

for (int i = 0; i < 1000; i++)

{

int integers1[100\*(i+1)]; int integers2[100\*(i+1)]; for(int j=0;j<100\*(i+1);j++){ integers1[j]=integers[j]; integers2[j]=integers[j];

}

clock\_t launch=clock(); selectionSort(integers1,100\*(i+1)); clock\_t done=clock();

double stime=((double)done-launch)/CLOCKS\_PER\_SEC; clock\_t launch2=clock(); insertionSort(integers2,100\*(i+1));

clock\_t done2=clock();

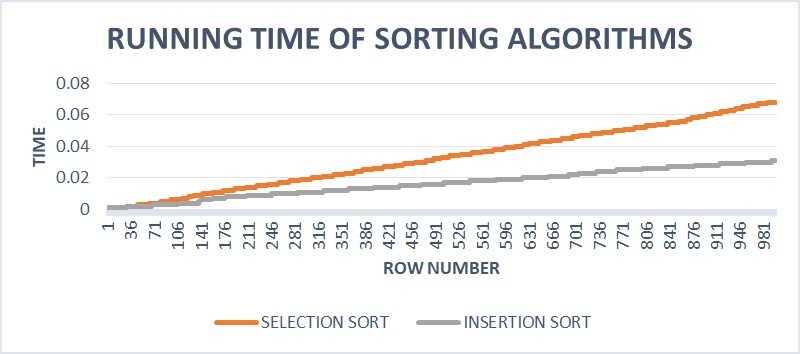
double itime=((double)done2-launch2)/CLOCKS\_PER\_SEC; printf("%d\t%0.3f\t",i+1,stime); printf("%0.3f\n",itime);

}

return 0;

}

**GRAPHICAL REPRESENTATION:**

****

**OBSERVATIONS:**

1. From the graph it can be observed that insertion sort is faster than selection sort.
2. The space complexity for both the algorithms is same i.e O(1).

**CONCLUSION:** I have successfully implemented both the sorting algorithms and have analysed their time and space complexity.