**Experiment No: 2**

**Aim**

Write a Program to demonstrate Selection Sort and analyse the Time Complexity with different range of inputs

**Introduction**

Selection sort is a simple algorithm for sorting a list of elements. It works by repeatedly finding the smallest element in the unsorted portion of the list and swapping it with the first element of the unsorted portion.

The basic idea behind selection sort is to divide the input list into two parts: the **sorted part** and the **unsorted part**. Initially, the sorted part is empty, and the unsorted part contains all the elements. In each iteration, the algorithm finds the smallest element in the unsorted part, and then places it at the end of the sorted part by swapping it with the first element of the unsorted part. This process is repeated until the unsorted part is empty, and the entire list is sorted.

**Algorithm**

* Selection Sort is not a **STABLE** algorithm by the default implementation.
* It is **IN-PLACE** algorithm since it does not require any extra space.

1. Initialize **minimum value(min\_index)** to location 0.
2. Traverse the array to find the minimum element in the array.
3. While traversing if any **element smaller than min\_index is found** then **swap** both the values.
4. Then, increment min\_index to point to the next element.
5. Repeat the above steps until the array is sorted.

**Program**

**#include <stdio.h>**

**#include <stdbool.h>**

**#include <stdlib.h>**

**#include <time.h>**

**#define N 100000**

**int i, j;**

**void swap(int \*a, int \*b)**

{

\*a = \*a + \*b;

\*b = \*a - \*b;

\*a = \*a - \*b;

}

**void reverse\_array(int \*arr)**

{

for (i = 0; i < N / 2; i++)

{

swap(&arr[i], &arr[N - i - 1]);

}

}

**bool isSorted(int \*arr)**

{

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

{

if (arr[i + 1] < arr[i])

{

return false;

}

return true;

}

}

**void selection\_sort(int \*arr)**

{

int min\_index;

long long int swap\_count = 0, comparisons = 0;

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

{

min\_index = i;

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

{

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

{

min\_index = j;

}

comparisons++;

}

if (min\_index != i)

{

swap(&arr[min\_index], &arr[i]);

swap\_count++;

}

}

printf("\nNo of Comparisons is %llu", comparisons);

printf("\nNo of Swap Counts is %llu\n", swap\_count);

}

**void calculateTime(int \*arr)**

{

clock\_t time;

time = clock();

selection\_sort(arr);

time = clock() - time;

double time\_taken = ((double)time) / CLOCKS\_PER\_SEC;

printf("The Time taken is equal to %.18lf seconds\n", time\_taken);

}

**int main()**

{

int arr[N];

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

{

arr[i] = rand();

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* A V E R A G E C A S E \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

printf("\nAverage Case");

calculateTime(arr);

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* B E S T C A S E \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

printf("\nBest Case");

isSorted(arr) ? printf("\nThe Array is Sorted , All is Well\n") : printf("The Array is not Sorted :( \n");

calculateTime(arr);

printf("\n");

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* W O R S T C A S E \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

printf("\nWorst Case");

reverse\_array(arr);

calculateTime(arr);

printf("\n");

}

**Output**

* ***“Comparisons”* refers the total number of checks of a[i+1] < a[i] for all i >= 0 to i < N-1 .**
* ***“Swap Counts“* refers to the total number of swaps made**

**when min\_index != i**

|  |  |  |  |
| --- | --- | --- | --- |
| **INPUT SIZE**  **( N )** | **AVERAGE CASE**  **TIME COMPLEXITY**  **(in ms)** | **BEST CASE**  **TIME COMPLEXITY**  **(in ms)** | **WORST CASE TIME COMPLEXITY**  **(in ms)** |
|  | **RunTime , Comparisons,**  **Swap Counts** | **RunTime , Comparisons,**  **Swap Counts** | **RunTime , Comparisons,**  **Swap Counts** |
| 10 | 0 ms  45  7 | **0 ms**  **45**  **0** | **0 ms**  **45**  **5** |
| 100 | **4 ms**  4950  92 | **4 ms**  **4950 0** | **5 ms**  **4950**  **50** |
| 1000 | 6 ms  499500  994 | 5 ms  499500  0 | 6 ms  499500  507 |
| 10000 | 117 ms  49995000  9992 | 116.4 ms  49995000  0 | 131 ms  49995000  5431 |
| 100000 | 11454 ms  4999950000  99981 | 11367 ms  4999950000  0 | 17759 ms  4999950000  66861 |

**Analysis**

**Best Case ( Sorted Array )** takes the least time since it undergoes only ( N-1 ) Comparisons. ( Time Complexity ≈ O(N) )

**Average Case ( Random Input Array )** takes on an average of the time of Best and Worst Cases. ( Time Complexity ≈ O( N2 / 4 ) ≈ O(N2) )

**Worst Case ( Reversed Array )** undergoes the most number of Comparsions and hence it takes the most time of all of cases.

No.of Comparisons = ( n-1 ) + ( n-2 ) + ( n-3 ) + …. 3 + 2 + 1

= **( n2 – n ) / 2 ;**

∴ Time Complexity ≈ O( N2 / 2 ) ≈ O(N2)

|  |  |  |  |
| --- | --- | --- | --- |
| **CASES** | **AVERAGE** | **BEST** | **WORST** |
| **TIME COMPLEXITY** | O(N2) | O(N2) | O(N2) |

**Applications**

**1 ) Sorting a small list of numbers:**

If you have a small list of numbers that needs to be sorted, insertion sort can be an efficient option. It's simple to implement and its time complexity is O(n^2) which make it good for small lists.

**2 ) Sorting Linked List :**

Insertion sort is also good for sorting linked list, when compared to other sorting algorithms like merge sort which needs extra space for merge operations

**References**

1 ) <https://www.geeksforgeeks.org/insertion-sort/>

( Press Ctrl + Click to open )

2) Image

https://www.geeksforgeeks.org/recursive-insertion-sort/