Idris Ratanwala
C31- 2003145

Experiment No : 1A

Aim : Implementation of Selection sort.

Theory :

In selection sort, the smallest value among the unsorted elements of the array is selected in every pass and inserted to its appropriate position into the array. It is also the simplest algorithm. It is an in-place comparison algorithm.

Here, the array is divided into two parts, first is sorted & another is unsorted part. Initially the sorted part is empty & unsorted part is the given array. Sorted part is in the left whereas unsorted is on the right.

In selection sort, the first smallest elem is selected from the unsorted array and placed at first position. After that second smallest element is selected and placed at second position, & repeats until whole array is sorted.

Algorithm: (pesude code)

```
selectionSort(array, size):
    repeat (size - 1) times:
    set the first unsorted element as the minimum
    for each of the unsorted elements
```

if element < currentMin
set element as new minimum
swap minimum with first unsorted position
end selectionSort.

Analysis :

At the beginning the size of sorted sub array (let S1) is 0 & that of unsorted (let S2) is N.

At each step, the size of sorted sub array increases by 1 and the size of unsorted sub array decreases by 1.

Hence, for a few steps are as follows :

Step 1 : S1 : 0 , S2 : N
Step 2 : S1 : 1 , S2 : N - 1
Step 3 : S1 : 2 , S3 : N - 2
⋮ ⋮ ⋮
and so on till S1 = N, hence there will be N+1 steps
ie. S2 = N - S1

The time complexity of finding the smallest element in a list of 'M' is O(M), which is constant for all worst, average or best cases.

Hence time req for finding smallest element in unsorted array will be O(S2)

For step 1, S1 will be I-1 & S2 will be N - S1
= N - I + 1

So time complexity for step 1 will be,
- $O(N - I + 1)$ - for finding smallest element.
- $O(1)$ for swapping smallest element.

I will range from 1 to N+1

∴ Time complexity of all the operations would be
Sum $[O(N - I + 1) + O(1)]$ for $I \in [1, N+1]$
= Sum $[O(N + I + 1)]$ + Sum $[O(1)]$ ------(1)

Now,
Sum $[O(1)] = 1 + 1 + 1 \ldots 1$ [(N+1) times]
$= N + 1 = O(N)$

& Sum $[O(N + I + 1)] = N + (N-1) + \ldots + 1 + 0$
$= 1 + 2 + \ldots + N$
$= N * (N+1)/2$

$= O(N^2)$

∴ We get eqn 1 as
$O(N^2) + O(N)$
$= O(N^2)$

Hence time complexity of selection sort is:
Best case is $O(n^2)$
Average case is $O(n^2)$
& Worst case is $O(n^2)$

Example :

We take an array 5, 3, 2, 4, which is to be sorted in ascending order.

1. Set variable 5, find the min from unsorted array
   min = 2
   ∴ swap 2 & 5

   2 | 3, 5, 4
   sorted | unsorted

2. Find min from unsorted array,
   min = 3
   swap 3 with itself,

   2, 3 | 5, 4

3. Find min from unsorted array,
   min = 4
   ∴ swap 4 with 5,

   2, 3, 4 | 5

∴ 2, 3, 4, 5 is the sorted array

Application :

1) Selection sort is useful when memory write is a costly operation
2) It always outperforms bubble sort & gnome sort

3) It almost always for exceeds the number of writes that cycle sort makes, as cycles ~~is~~ sort is optimal in the number of writes.
4) This can be useful when writes are significantly more expensive than ~~re~~reads, such as EEPROM or flash memory, where ~~e~~every write lessens the lifespan of memory.

Program :

```
import java.util.Scanner;

public class SelectionSort {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number of elements : ");
        int n = sc.nextInt();
        int[] arr = new int[n];

        System.out.print("Enter the Elements : ");
        for(int i=0;i<n;i++) arr[i]= sc.nextInt();

        int swaps = selectnSort(arr);
        System.out.println("\nSorted array :");
        for (int k=0;k<arr.length;k++) {
            System.out.print("\t"+arr[k]);
        }
        System.out.println("\n\nNumber of swaps : "+ swaps);
    }

    private static int selectnSort(int[] arr) {
        int swaps=0;
        System.out.println("\n***Selection Sort*** \nPasses :");
        for(int i=0;i<arr.length;i++){
            print(arr,i);
            int min = i;
            for(int j=i+1;j<arr.length;j++){
                if(arr[min]>arr[j]) min = j;
            }
            int temp = arr[i];
            arr[i] = arr [min];
            arr[min] = temp;
            swaps++;
        }
        return swaps;
    }

    private static void print(int[] arr, int i) {
        for (int k=0;k<arr.length;k++) {
            System.out.print("\t"+arr[k]);
            if(i-1 == k) System.out.print("   |");
        }
        System.out.println();
    }
}
```

Output :

```
Enter the number of elements : 6
Enter the Elements : 6 7 5 3 2 8

***Selection Sort***
Passes :
        6       7       5       3       2       8
        2   |   7       5       3       6       8
        2       3   |   5       7       6       8
        2       3       5   |   7       6       8
        2       3       5       6   |   7       8
        2       3       5       6       7   |   8

Sorted array :
        2       3       5       6       7       8

Number of swaps : 6
PS C:\Users\IsmailRatlamwala\Documents\College prog\AOA>
```

Conclusion: Selection sort is an unstable algorithm that is good for sorting small data sets. Time taken by this algo is more but it is benificial to senarios where we have memory limitations. It is simple & easy to implement.