Tutorial Link https://codequotient.com/tutorials/Selection Sort/5a12e94346765b2b63e34754

TUTORIAL

Selection Sort

Chapter

This sorting algorithm is an in-place comparison-based algorithm in which the list is divided into two parts, the sorted part at the left end and the unsorted part at the right end. Initially, the sorted part is empty and the unsorted part is the entire list. The smallest element is selected from the unsorted array and inserted at the end of sorted array. This process continues moving unsorted array boundary by one element to the right. Following algorithm explains selection sort: -

```
for i := 1 to n-1
   select the smallest among A[i], . . . , A[n]
   swap it with A[i];
end
```

Let's take the below example: -

```
15   11   14   12   18
```

To sort this array in ascending order, Selection sort will perform following steps: -

```
While i=0, sorted array is empty and unsorted array is from 0 to 4
```

Find the minimum element in array[0...4] (which is 11) and place it at end of sorted array

```
11   15   14   12   18
```

```
While i=1, sorted array is from 0 to 0 and unsorted array is from 1
to 4
```

Find the minimum element in array[1...4] (which is 12) and place it at end of sorted array

```
11   12   15   14   18
```

```
While i=2, sorted array is from 0 to 1 and unsorted array is from 2
to 4
```

Find the minimum element in array[2...4] (which is 14) and place it at end of sorted array

```
11   12   14   15   18
```

```
While i=3, sorted array is from 0 to 2 and unsorted array is from 3
to 4
```

Find the minimum element in array[3...4] (which is 15) and place it at end of sorted array

```
11   12   14   15   18
```

```
While i=4, sorted array is from 0 to 3 and unsorted array is from 4
to 4
```

Find the minimum element in array[4...4] (which is 18) and place it at end of sorted array

```
11   12   14   15   18
```

The last element is always at the proper position so we can skip the last phase.

Implementation of selection sort

```c
#include<stdio.h>

void printArray(int array[], int size)
{
  int i;
  for (i=0; i < size; i++)
    printf("%d ", array[i]);
  printf("\n");
}

void selectionSort(int array[], int n)
{
  int i, j, index, temp;
  for (i = 0; i < n-1; i++)  // n-1 as last element is always sorted.
  {
```

```

    index = i;
    for (j = i+1; j < n; j++)        // Find the minimum element
      if (array[j] < array[index])
        index = j;
    printf("While i = %d\n",i);
    printf("Minimum Element = %d\n",array[index]);

    temp = array[index];    /* Swap the minimum element with first element of unsorted array so that size of sorted array will increase. */
    array[index] = array[i];
    array[i] = temp;
    printf("Elements swapped are %d & %d\n",array[i], array[index]);
  }

  printf("Array after %d iterations - \n",i+1);
  printArray(array, n);   // During Sorting
  printf("\n");

}

int main()
{
  int array[] = {15, 11, 14, 12, 18};
  int n = 5;
  // we can calculate the number of elements in an array by using sizeof(array)/sizeof(array[0])
  printf("Un-Sorted array: \n");
  printArray(array, n);     // Unsorted array
  selectionSort(array, n);     // Call the sorting routine
  printf("\nSorted array: \n");
  printArray(array, n);   // Sorted array
  return 0;
}

```

```
import java.util.Scanner;
```
Java

```
// Other imports go here
// Do NOT change the class name
class Main {


    static void printArray(int array[], int size) {
        int i;
        for (i = 0; i < size; i++)
            System.out.printf("%d ", array[i]);
        System.out.printf("\n");
    }

    static void selectionSort(int array[], int n) {
        int i, j, index, temp;
        for (i = 0; i < n - 1; i++) // n-1 as last element is always sorted.
        {
            index = i;
            for (j = i + 1; j < n; j++) // Find the minimum element
                if (array[j] < array[index])
                    index = j;
            System.out.printf("While i = %d\n", i);
            System.out.printf("Minimum Element = %d\n", array[index]);

            temp = array[index]; /* Swap the minimum element with first element of unsorted array so that size of sorted array will increase. */
            array[index] = array[i];
            array[i] = temp;
            System.out.printf("Elements swapped are %d & %d\n", array[i], array[index]);
        }

        System.out.printf("Array after %d iterations - \n", i + 1);
        printArray(array, n); // During Sorting
        System.out.printf("\n");

    }

```

```
    public static void main(String[] args) {
        int array[] = {15,11,14,12,18};
        int n = array.length;
        // we can calculate the number of elements in an array by using sizeof(array)/sizeof(array[0])
        System.out.printf("Un-Sorted array: \n");
        printArray(array, n); // Unsorted array
        selectionSort(array, n); // Call the sorting routine
        System.out.printf("\nSorted array: \n");
        printArray(array, n); // Sorted array
    }
}
```

Python 3

```

def printArray(A,size):
    for i in range(size):
        print(A[i],end=' ');
    print()

def selectionSort(A,n):
    for i in range(len(A)-1):
        index = i
        for j in range(i+1, len(A)):
            if A[index] > A[j]:
                index = j
        print('While i = '+str(i));
        print('Minimum Element = '+str(A[index]));
        A[i], A[index] = A[index], A[i] # Swap the elements
        print('Elements swapped are '+str(A[i])+' and '+str(A[index]));
    print('Array after '+str(len(A))+' iterations');
    printArray(A,n);
    print();



if __name__=="__main__":
    A = [15,11,14,12,18]
```

```
    print('Unsorted Array:')
    printArray(A,len(A));
    print()


    selectionSort(A,len(A));

    print('Sorted Array');
    printArray(A,len(A));

```

C++

```
#include<iostream>
using namespace std;
void printArray(int array[], int size){
    int i;
    for (i=0; i < size; i++)
        cout<<array[i]<<" ";
    cout<<'\n';
}

void selectionSort(int array[], int n){
    int i, j, index, temp;
    for (i = 0; i < n-1; i++)  // n-1 as last element is always sorted.
    {
        index = i;
        for (j = i+1; j < n; j++)        // Find the minimum element
            if (array[j] < array[index])
                index = j;
        cout<<"While i = "<<i<<endl;
        cout<<"Minimum Element = "<<array[index]<<endl;

        temp = array[index];    /* Swap the minimum element with first element of unsorted array so that size of sorted array will increase. */
        array[index] = array[i];
        array[i] = temp;
        cout<<"Elements swapped are "<<array[i]<< " & " <<array[index]<<endl;
```

```
    }

    cout<< "Array after"<<i+1<<" iterations - \n";
    printArray(array, n);   // During Sorting
    cout<<endl;

}

int main(){
    int array[] = {15, 11, 14, 12, 18};
    int n = 5;
    // we can calculate the number of elements in an array by using sizeof(array)/sizeof(array[0])
    cout<<"Un-Sorted array:"<<endl;
    printArray(array, n);     // Unsorted array
    selectionSort(array, n);     // Call the sorting routine
    cout<<"\nSorted array:"<<endl;
    printArray(array, n);   // Sorted array
    return 0;
}

```

Output of above program is: -

```
Un-Sorted array:
15 11 14 12 18
While i = 0
Minimum Element = 11
Elements swapped are 11 & 15
While i = 1
Minimum Element = 12
Elements swapped are 12 & 15
While i = 2
Minimum Element = 14
Elements swapped are 14 & 14
While i = 3
Minimum Element = 15
Elements swapped are 15 & 15
Array after 5 iterations -
11 12 14 15 18
```

```
Sorted array:
11 12 14 15 18
```

The output above shows the swapping done in each iteration and final array after each pass. The algorithm always run n-1 times, where n is the number of elements in the array. In each iteration it has to search for minimum element in unsorted part. The time complexity of this algorithm is O(n^2) as both loops in selectionSort() function will run n times.

Properties of Selection sort

Worst and Average Case Time Complexity: O(n^2). Worst case occurs when array is sorted in opposite direction.

Best Case Time Complexity: O(n^2). As in all cases, it will search minimum linearly for all n-1 iterations.

Auxiliary Space: O(1)

Sorting In Place: Yes

Stable: Depends on implementation

The good thing about selection sort is it never makes more than O(n) swaps and can be useful when memory write is a costly operation.