Selection sort summary

# Overview of Selection sort

The Selection sort algorithm is one of the easiest algorithms to understand and implement. On the face of it there is a similarity to bubble sort and insertion sort. It is faster than bubble sort even if both are O(n2) algorithms. Compared to insertion sort selection sort has to continuously iterate through the unsorted list, whereas the former can pick the right element.

On average, Selection sort takes O(n2), comparisons to sort n items. Worst case performance is O(n2), but this is rare. Selection sort is efficient with small arrays (< 20 elements) and can be used to create a hybrid algorithm.

# Run-through of algorithm

Selection sort consists of two for-loops. The inner loop is specifically there to iterate over the unsorted part of the list. Consider the following unsorted array.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 4 | 2 | 5 | 9 | 0 | 1 | 3 | 6 | 8 |

The algorithm starts by assuming the first element is the smallest. It remembers the index of this element J and sets a variable to its value (in this case index is 0). Next the inner loops goes over the remaining elements in the list. It thus starts at element J + 1.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 4 | 2 | 5 | 9 | 0 | 1 | 3 | 6 | 8 |

^ ^

The inner loop I will compare every element with that stored by the outer loop J. If the element is smaller the index stored in the outer loop will be updated.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 4 | 2 | 5 | 9 | 0 | 1 | 3 | 6 | 8 |

^ ^

In this case by the time the inner loop reaches the last element with value 8 the smallest element with value 0 is at index 5. Finally as the last step in the outer loop before the index is updated the elements are swapped. Element value 7 at index 0 is swapped with element value 0 at index 5.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 4 | 2 | 5 | 9 | 7 | 1 | 3 | 6 | 8 |

^ ^

The outer loop sets the smallest element at index 1 and the inner loop starts to compare elements from J + 1 onwards. This is repeated until the outer loop reaches index size n-1.

#### Run-time and memory use

Selection is considered an in-place sort – meaning all operations are performed on the same indices stored in memory. At first glance there is a lot similar to bubble sort, but the number of swaps is limited.

# Implementation in Python and Java

Below is my implementation Python

import sys

import timeit

class Selectionsort(object):

def \_\_init\_\_(self):

print()

def selectionsort(self, unsorted):

for i in range(0, len(unsorted)):

minvalue = unsorted[i]

index = None

for j in range(i+1, len(unsorted)):

if unsorted[j] < minvalue:

minvalue = unsorted[j]

index = j

if index is not None:

unsorted[index] = unsorted[i]

unsorted[i] = minvalue

def main():

unsorted = [7, 3, 8, 2, 1, 9, 4, 6, 5, 0]

selection = Selectionsort()

selection.selectionsort(unsorted)

print(unsorted)

if \_\_name\_\_ == "\_\_main\_\_":

main()

And now in Java

public class Selection {

public static void sort(Comparable[] a) {

int N = a.length;

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

int min = i;

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

if (less(a[j], a[min])) {

min = j;

}

}

exch(a, i, min);

}

}

private static boolean less(Comparable v, Comparable w) {

return v.compareTo(w) < 0;

}

private static void exch(Comparable[] a, int i, int j) {

Comparable t = a[i];

a[i] = a[j];

a[j] = t;

}

public static void show(Comparable[] a) {

for (int i = 0; i < a.length; i++) {

StdOut.print(a[i] + " ");

}

StdOut.println();

}

public static boolean isSorted(Comparable[] a) {

for (int i = 1; i < a.length; i++) {

if (less(a[i], a[i-1])){

return false;

}

}

return true;

}

public static void main(String[] args) {

StdOut.println("Starting selection sort");

String[] a = StdIn.readAllStrings();

sort(a);

assert isSorted(a);

show(a);

}

}