# Sorting By Selection

- In case of insertion sort techniques - it is not necessary that all keys to be sorted should be available before the sorting start.
  The key values are read from the keyboard or a file one at a time while the sorting procedure continues.
  It may be noted that a pass in the Insertion sorting reads a key and puts it into the output list, which is sorted with key values read so far.

- In contrast to this, Selection Sort requires all the keys under sorting be available prior to the execution.
  Further Selection Sort considers the following two basic operations:→

| |
|---|
| (a) Select :→ Selection of an item from the input list. |
| (b) Swap :→ Interchange two items in the list. |

These two above → steps in Selection Sort are iterated to produce the sorted list, which progress from one end as the iteration continues.

Selection Sort → is <u>in place</u> Sorting Technique.

SORTING BY SELECTION →
- STRAIGHT SELECTION SORT
- TREE SORT
- HEAP SORT

## Straight Selection Sort

→ Suppose there are $n$ number of key values in the list to be sorted. This technique ~~remains~~ requires $n-1$ iterations to complete the sorting.

→ Let us consider the case of $i^{th}$ iteration, where $1 \le i < n$. The $i^{th}$ iteration begins with $i-1$ keys, say, $K_1 \le K_2 \le \dots \le K_{i-1}$ which are already in sorted order.

→ Select :→ Select the smallest key in the list of remaining key values say $K_i, K_{i+1}, \dots, K_n$. Let the smallest key value be $K_j$ $(i \le j \le n)$.

→ Swap : Swap the two key values $K_i$ and $K_j$.

### ALGORITHM OF STRAIGHT SELECTION SORT.

Input :- An input list $A[1 \dots N]$

Output :- The list $A[1 \dots N]$ in sorted order.

Remark :- Sorted in Ascending Order.

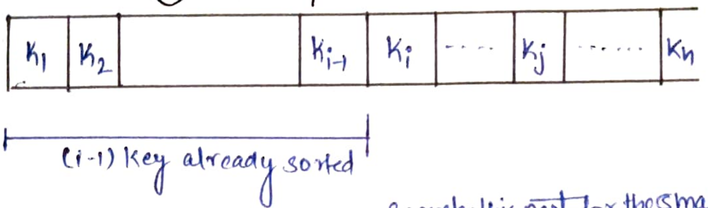
Steps :

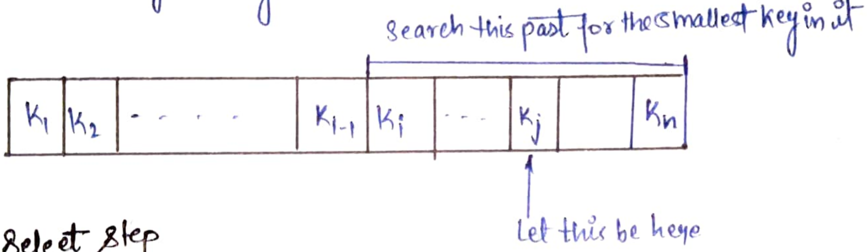
1. For $i = 1$ to $(n-1)$ do // $(n-1)$ iteration
2. $j$ = SelectMin$(i, n)$ // Select the smallest from the remaining part of the list
3. If $(i \neq j)$ then // Interchange when the minimum is in remote.
4. Swap$(A[i], A[j])$
5. EndIf
6. Endfor
7. Stop

procedure SelectMin (i, n) is to find the smallest element in the list b/w the location i and n both inclusive, which is described as the algorithm SelectMin below.

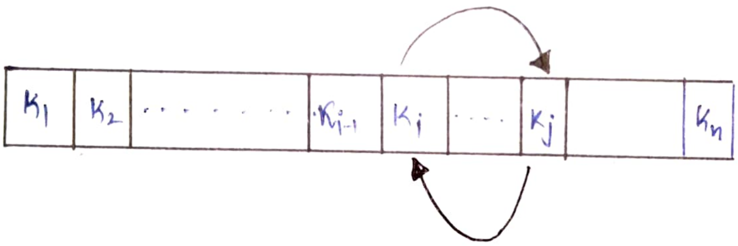
→ The procedure to perform the swap operation b/w the data value X and Y is described in the algorithm Swap.

ⓐ The list prior to the ith iteration




ⓑ Select Step




ⓒ Swap Step

<u>Illustration of the Straight Selection Sort</u>

## ALGORITHM SELECT MIN

Input: A list in the form of an array of size n with L(left) and R(right) being the selection range, where 1≤L, R≤n.

Output: The index of array A where the minimum is located.

Remark: If the list contains more than one minimum, then it returns the index of the first occurrence.

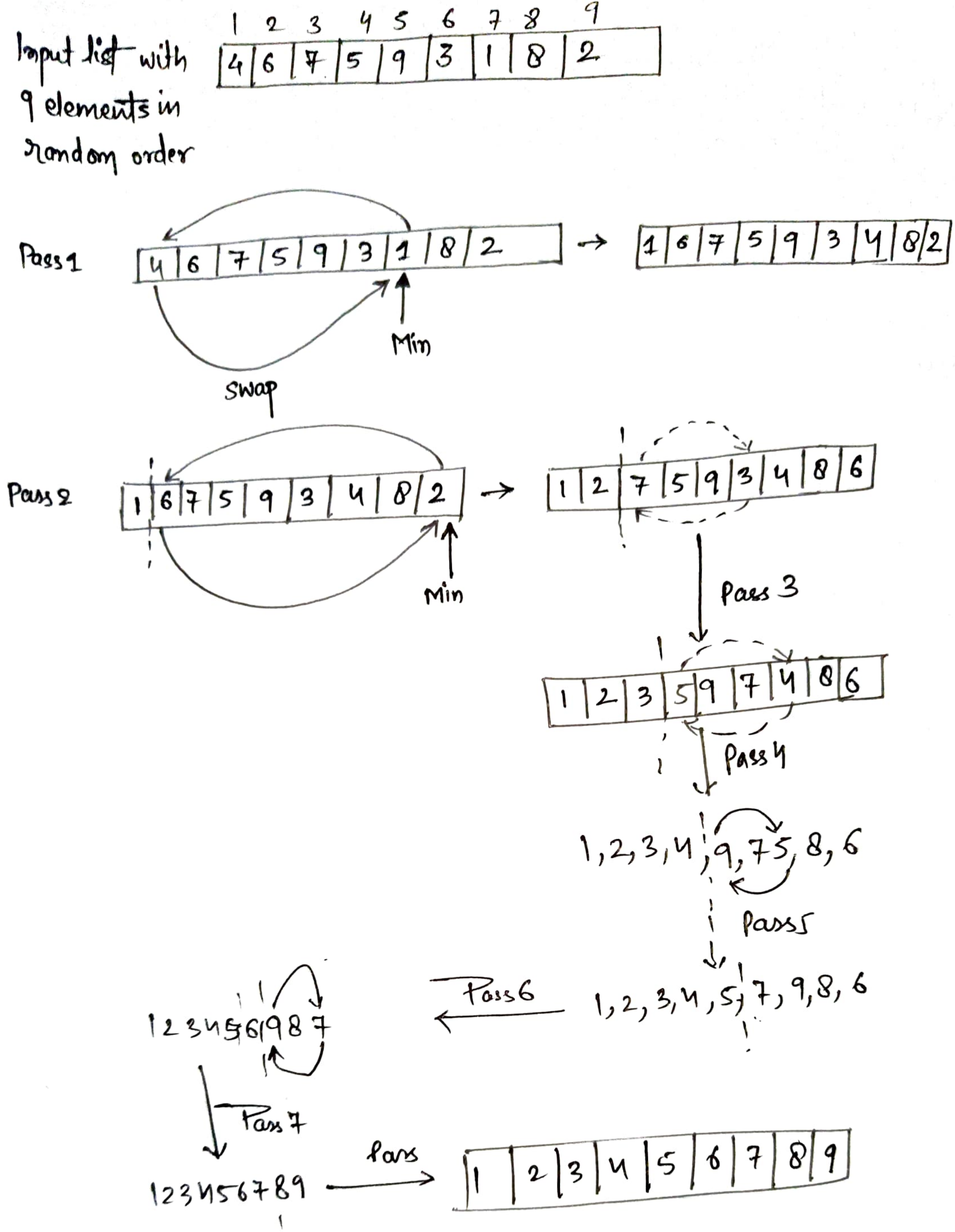
Steps

1 → min = A[L] // Initially, the item at the starting loc. is choosen as the smallest
2 → min Loc = L // minloc records the location of the minimum
3 → For i = L+1 to R do // Search the entire part
4 → If (min > A[i]) then
5 → min = A[i] // Now the smallest is updated here
6 → minLoc = i // New location of the smallest so far
7 → EndIf
8 → EndFor
9 → Return(minLoc) // Return the location of the smallest element
10 → Stop

## ALGORITHM SWAP

Input:→ X and Y are two variables
Output:→ The value in X goes to Y and vice-versa.
Remark:→ X and Y should be passed

Steps:
1. → temp = X // Store the value in X in a temporary storage space.
2. → X = Y // Copy value of ~~X~~ Y to X
3. → Y = temp // Copy the value in temp to Y.
4. → Stop



Input list with 9 elements in random order
1 2 3 4 5 6 7 8 9
4 6 7 5 9 3 1 8 2
Pass 1
4 6 7 5 9 3 1 8 2
Min
swap
1 6 7 5 9 3 4 8 2
Pass 2
1 6 7 5 9 3 4 8 2
Min
1 2 7 5 9 3 4 8 6
Pass 3
1 2 3 5 9 7 4 8 6
Pass 4
1, 2, 3, 4, 9, 7 5, 8, 6
Pass 5
1, 2, 3, 4, 5, 7, 9, 8, 6
Pass 6
1 2 3 4 5 6 9 8 7
Pass 7
1 2 3 4 5 6 7 8 9
Pass
1 2 3 4 5 6 7 8 9

## Analysis of the algorithm Straight Selection Sort

It may be noted that the straight selection sort performs the sorting on the same list as the input list.
Hence, it is an inplace sorting technique.
In other words, the straight selection sort does not require additional storage space other than the space to store the input list itself.

**Case 1: The input already in sorted order.**

(a) No. of Comparison

→ algorithm Straight Selection Sort requires a total of $(n-1)$ iterations and for each the SelectMin() function is invoked.

→ Now consider ith iteration → it is evident that in the ith iteration, $(i-1)$ items are already sorted and the Select Min(..) function searches for the minimum in the remaining $n-(i-1) = n-i+1$ elements.

To do this, it needs $(n-i+1)-1 = n-i$ no. of comparisons.

$$C(n) = \sum_{i=1}^{n-1} (n-i)$$

$$= (n-1)+(n-2)+\cdots+(n-\overline{n-2})+(n-\overline{n-1})$$

$$= \frac{n(n-1)}{2}$$

(b) No. of movement

In this case No SWAP OPERATION is involved since the input is in sorted order. Hence the number of movements is

$$M(n)=0$$

(c) Memory Requirement

$$S(n)=0$$

Case 2: The input list is stored but in reverse order.

(a). Number of comparisons

Like the case 1 analysis of the sorting algorithm, we can calculate that the no. of comparisons $C(n)$ to sort $n$ items is given by

$$\boxed{C(n) = \frac{n(n-1)}{2}}$$

(b). Number of Movements.

If we observed the algorithm on applying on example. It can be observed that when the list is in reverse order, there are movements of data until the half part of the list is sorted.

When we have done just sorting of half of the list, the element in the rest (unsorted) of the part are already in their final positions and hence NO DATA MOVEMENT OCCURS.

Therefore Swap( ) operation take place only in the first $(n-1)/2$ iterations.

Further, a single swap operation is associated with Three Data movement (considering the first version of the Swap(...), with temporary copy).

Therefore, the no. of movt. required with a list of size $n$ & when it contains the element in reverse order is

$$\boxed{M(n) = \frac{3}{2}(n-1)}$$

(c) Memory Requirement

$$\boxed{S(n) = 0}$$

Case 3 The element in the input list are in random order.

(a) No. of comparisons.

$$C(n) = \frac{n(n-1)}{2}$$

(b) No. of movements

→ The straight selection sort does not perform any swap operation, if the $i^{th}$ smallest element is present in the $i^{th}$ location.

→ Let $P_i$ be the probability that the $i^{th}$ smallest element is in the $i^{th}$ position.

→ Hence, the probability that there will be swap operation in the $i^{th}$ pass is $(1-P_i)$.

→ So, on the average the no. of swap in $(n-1)$ total iteration is

$$= (1-P_i) \times (n-1)$$

→ To simplify, the analysis, let us assume that all keys are distinct, and all permutations of keys are equally likely as input. Then we have

$$\boxed{P_1 = P_2 = \cdots = P_n = \frac{1}{n}}$$

→ With this assumption and considering that there are THREE data movements in a swap operation, the average no. of movements in the algorithm straight selection sort becomes

$$\boxed{M(n) = \left(1-\frac{1}{n}\right) \times (n-1) \times 3 = \frac{3(n-1)(n-1)}{n}}$$

(c) Memory Requirement

The storage space required is

$$\boxed{S(n) = 0}$$

## Analysis of the algorithm Straight Selection Sort

| Case | Comparison | Movement | Memory | Remark |
|---|---|---|---|---|
| Case 1 | $C(n) = \frac{n(n-1)}{2}$ | $M(n)=0$ | $S(n)=0$ | Input list is in order order. |
| Case 2 | $C(n)= \frac{n(n-1)}{2}$ | $M(n) = \frac{3(n-1)}{2}$ | $S(n)=0$ | Input list is sorted in reverse order. |
| Case 3 | $C(n)= \frac{n(n-1)}{2}$ | $M(n) = \frac{3(n-1)^2}{n}$ | $S(n)=0$ | Input list is in random order. |
| | | | | |

The time complexity $T(n)$ of the algorithm Straight Selection Sort can be calculated considering the no. of comparisons and no. of movements, that is,

$$T(n) = t_1 \cdot C(n) + t_2 \cdot M(n)$$

where $t_1$ and $t_2$ denote the times for a single comparison and movement respectively.

## Time Complexity of the algorithm of Straight Selection Sort

| Case | RunTime, $T(n)$ | Complexity | Remark |
|---|---|---|---|
| Case 1 | $C(n)= n(n-1)/2$ | $T(n)=O(n^2)$ | Best Case |
| Case 2 | $C(n)=\frac{(n-1)(n+3)}{2}$ | $T(n)= O(n^2)$ | Average Case |
| Case 3 | $C(n)= \frac{(n-1)(n+6)}{2}$ | $T(n)=O(n^2)$ | Worst Case |
| | | | |

Note: from the analysis of the straight selection sort, it is ~~ele~~ evident that the number of comparisons is independent of the ordering of elements in the input list. However the performance of the algorithm varies wrt the no. of movement involved. For the sake of ~~complexity~~ simplicity, let us consider the large value of $n$, such that $n-1 \approx n$. With this consideration, the no. of movements required in case 2 is $3n/2$ and in case 3 it is $3n$. Thus, case 2 is better compared to the case 3.

Therefore, we can term the best case execution of the algorithm when the input list is sorted & the worst case when the i/p list is in random.