Selection Sort

The idea of Selection Sort is rather simple, we repeatedly find the next largest element in the array and move it to its final position in the sorted array. Assume that we wish to sort the array in increasing order i.e the smallest element at the beginning of the array and the largest element at the end. We begin by selecting the largest ~~state~~ element and moving it to the largest index position. The selection sort works by selecting the smallest unsorted item remaining in the list, and the swapping it with the item in the next position to be filled.

SELECTION-SORT (A)

```
n ← length[A]
for j ← 1 to n-1
    smallest ← j
    for i ← j+1 to n
        If A[i] < A[smallest]
            Then smallest ← i
    exchange(A[j], A[smallest])
```

A[] =

1	2	3	4	5
5	2	1	4	3

Here n = 5

For j = 1 to 4

j = 1, Smallest = 1

for i = 2 to 5

i = 2, Smallest = 1

A[2] = 2 A[1] = 5, A[2] < A[1]

then Smallest = 2.

Now, i = 3. Smallest = 2

A[3] = 1 A[2] = 2 A[3] < A[2]

then smallest = 3

Now i = 4, Smallest = 3

A[4] = 4

A[3] = 1 A[4] > A[3] No change

Now i = 5, Smallest = 3.

A[5] = 3

A[3] = 1 A[5] > A[3] So, No change.

then exchange (A[1], A[Smallest])

ie exchange (5, 1)

Now,

A[] =

1	2	5	4	3

Now, j = 2, smallest = 2

for i = 3 to 5

i = 3, smallest = 2

A[3] = 5

A[2] = 2   A[3] > A[2]   No change.

Now   i = 4   smallest = 2   No change

i = 5   smallest = 2   No change

Now j = 3, smallest = 3

for i = 4 to 5

Now, i = 4, smallest = 3

A[4] = 4

A[3] = 5   A[4] < A[3] then smallest = 4

Now   i = 5, smallest = 4

A[5] = 3

A[4] = 4   A[5] < A[4]   then smallest = 5

Now, exchange (A[3], A[4])

then

A[] =

1	2	3	4	5

For j = 4, smallest = 4, i = 5

A[5] = 5

A[4] = 4   A[5] > A[4]   No change.

Hence sorted array is

1	2	3	4	5

Analysis

ie first

cation

(n-2)

pass

Con

Analysis of Selection Sort

The first pass required $(n-1)$ Comparison to find the location of Smallest element, the Second pass required $(n-2)$ .... k^{th} pass requires $(n-k)$, and the last pass requires only one comparison. Total no of Comparison are.

$$= (n-1) + (n-2) + (n-3) \cdots + 3 + 2 + 1$$

$$= n(n-1)/2 = O(n^2)$$

Quick Sort

C. A. R Hoare implements quick sort by divide and conquer method. That means divide the big problem into two smaller problems and then those two small problems into two small ones and soon.

In Quick Sort, we divide the original list into two sublists. we choose the items from list called key or pivot from which all the left side of elements are smaller and all the right side of elements are greater than that element.

Thus, Quick Sort works by partitioning a given array A[p...r] into two non-empty sub-arrays A[p...q] and A[q+1...r] such that every key in A[p...q] is less than or equal to every key in A[q+1...r]. Then the two sub-arrays are sorted by recursive calls to Quick Sort. The exact position depends on the given array and index q is computed as a part of the partitioning procedure.

```
QUICK_SORT (A, p, r)
    If p<r then
        q ← PARTITION (A, p, r)
        QUICK_SORT (A, p, q−1)
        QUICK_SORT (A, q+1, r)
```

Partitioning the Array:

Partitioning procedure rearranges the subarray in place.

(2)
```
PARTITION (A, p, r)
    x ← A[r]
    i ← p−1
    for j ← p to r−1
        do if A[j] ≤ x
            then i ← i+1
                exchange A[i] ↔ A[j]
    exchange A[i+1] ↔ A[r]
    return i+1
```

Partition selects the first key, A[p] as a pivoted key about which array will be partitioned

- key ≤ A[p] [Left side]
- key ≥ A[p] [right side]

Use quick sort algorithm to sort

36, 15, 40, 1, 60, 20, 55, 25, 50, 20

Is a stable sorting algorithm?

Let A[] =

1	2	3	4	5	6	7	8	9	10
36	15	40	1	60	20	55	25	50	20

Here $p = 1$ $r = 10$

$x = A[10]$ ie $x = 20$

$i = p - 1$ ie $i = 0$

$j = 1$ to 9

$j = 1$ and $i = 0$

$A[j] = A[1] = 36$ and $36 \nleq 20$

$j = 2$ and $i = 0$

So, $A[2] = 15$ and $15 \leq 20$. (True)

$i = 0 + 1 = 1$

and $A[1] \Leftrightarrow A[2]$

1	2	3	4	5	6	7	8	9	10
15	36	40	1	60	20	55	25	50	20

Now $j = 3$ and $i = 1$

$A[3] = 40 \nleq 20$

$j = 4$ and $i = 1$

$A[4] = 1 \leq 20$ (True)

$i = i + 1 = 2$

and $A[2] \Leftrightarrow A[4]$

1	2	3	4	5	6	7	8	9	10
15	1	40	36	60	20	55	25	50	20

Now $j = 5$ and $i = 2$

$A[5] = 60 \nleq 20$

$j = 6$ and $i = 2$

$A[j] = 20 \leq 20$.(True)

$i = i + 1 = 2$ ie $i = 2 + 1 = 3$

and $A[3] \longleftrightarrow A[6]$

ie

1	2	3	4	5	6	7	8	9	10
15	1	20	36	60	40	55	25	50	20

Now $j = 7$ and $i = 3$

$A[j] = 55$ and . $55 \nleq 20$

so, $j = 8$ and $i = 3$

$A[8] = 25 \nleq 20$

$j = 9$. $i = 3$

$A[9] = 50 \nleq 20$

Now $A[i+1]$ ie $A[4] \Longleftrightarrow A[10]$

ie

1	2	3	4	5	6	7	8	9	10
15	1	20	20	60	40	50	25	50	36

and two subarrays are

15	1	20

and

60	40	50	20	50	36

This is stable Algorithm