Quick Sort

- Another method which is different from Merge Sort in which, the divided sub files are not need to be merged later.
- This method uses a procedure known as "PARTITION" (C.A.R. Hoare).

Partitioning

- To partition data is to divide it into two groups, so that all the items with a key value higher than a specified value are in one group, and all the item with a lower key value are in another.

  - Data could be partitioned more than 2-ways

- The value used to determine in which of two groups an item is placed is called a **pivot value**

- Basically the quicksort algorithm operates by partitioning an array into two subarrays, and then calling itself to quicksort each of these subarrays.

- Algorithm
  - Partition the array into left and right subarrays
  - Call itself to sort left subarray
  - Call itself to sort right subarray
    - If array has only one element it is sorted

- How is the pivot selected

Global n: integer;

A:array [1..n] of items;

procedure PARTITION(m:integer {index}, var p:integer{index+1});

{within A[m], A[m+1],..., A[p-1] the elements are rearranged in such a way that if initially t = A[m], then after completion A[q] = t, for some q between m and p-1, $A[k] \leq t$, $m \leq k \leq q$, $A[k] \geq t, q < k < p$, final value of p is q}

var i: integer; v: item;

begin

v ← A[m]; i← m; {A[m] is a partition element}

while (i < p) do

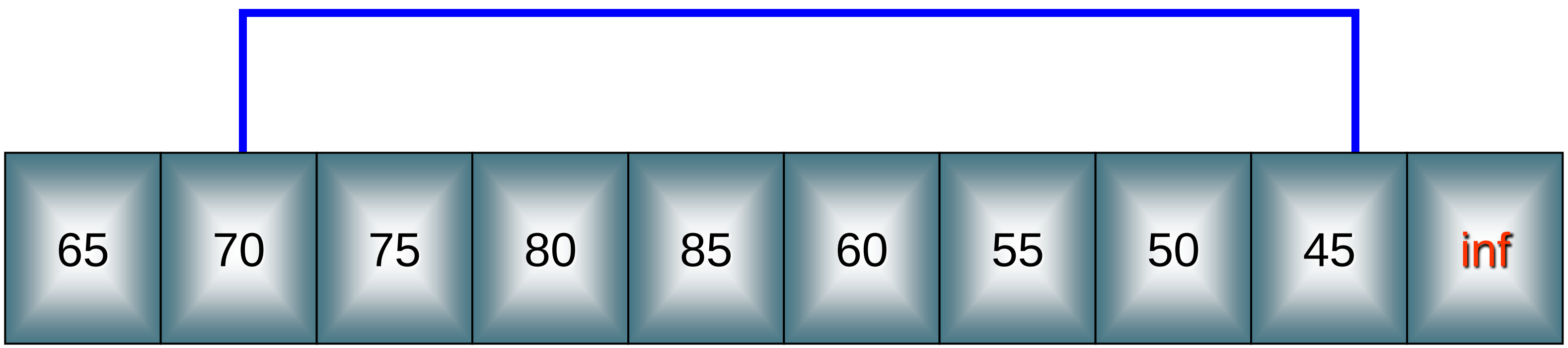
```
    begin
      i ← i+1;
      while(A[i] < v) do i ← i+1;   {i moves left to right}
      p ← p-1;
      while(A[p] > v) do p ← p-1; {p moves right to left}
      if (i < p) then swap(A[i], A[p]);
    end
      A[m] ← A[p]; A[p] ← v;        {the partition element
                                              belongs at p}
end.
```

Global n: integer;   A:array [1..n] of items;

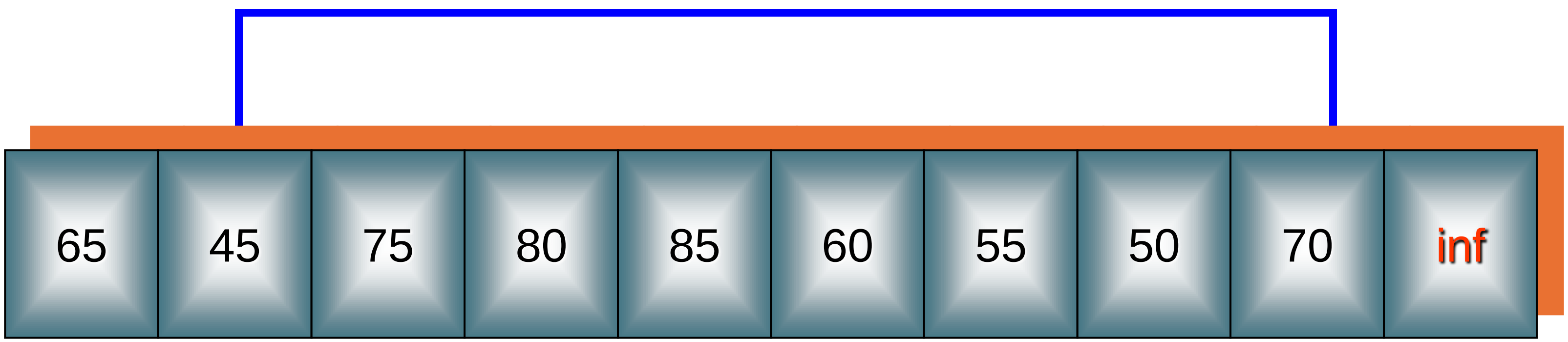
procedure QUICKSORT(p, q:integer);

{sort elements A[p]..A[q] which resides in the A[1..n], in ascending order. A[n+1] >= all elements in A[p..q], and set to infinity}

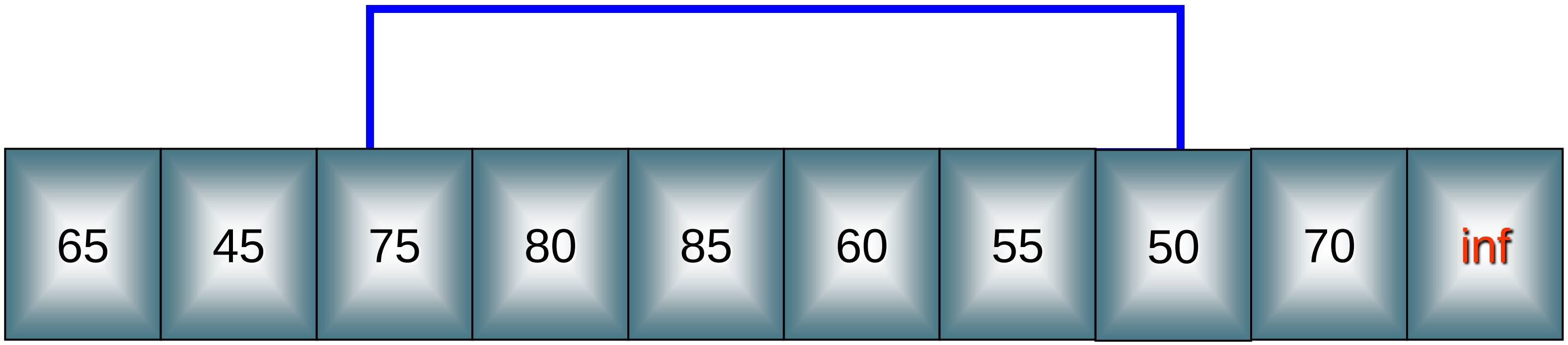
```
begin
        if (p < q) then
        begin
                j ← q+1;
                PARTITION(p,j);
                QUICKSORT(p, j-1);
                QUICKSORT(j+1, q);
        end
end.
```



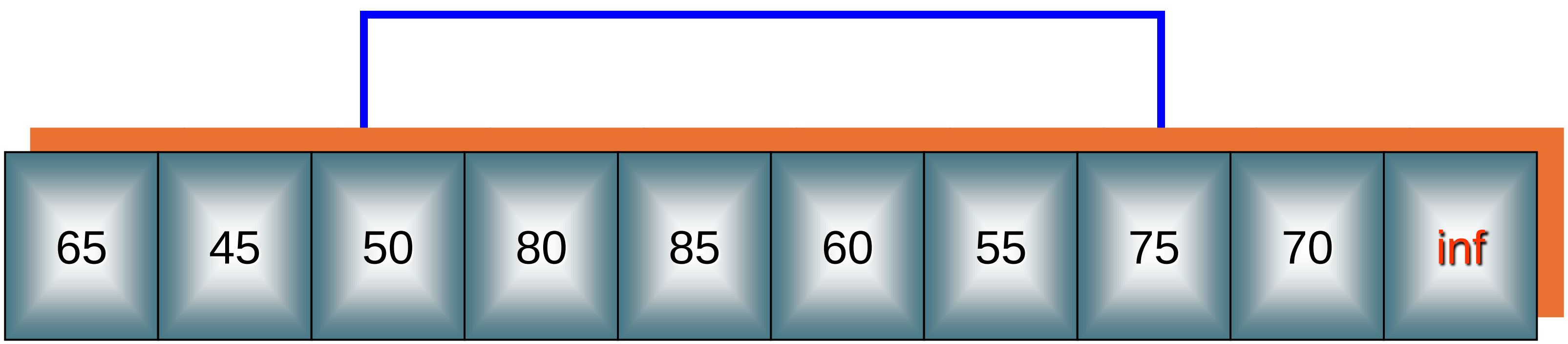
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	i	p
65	70	75	80	85	60	55	50	45	∞	2	9



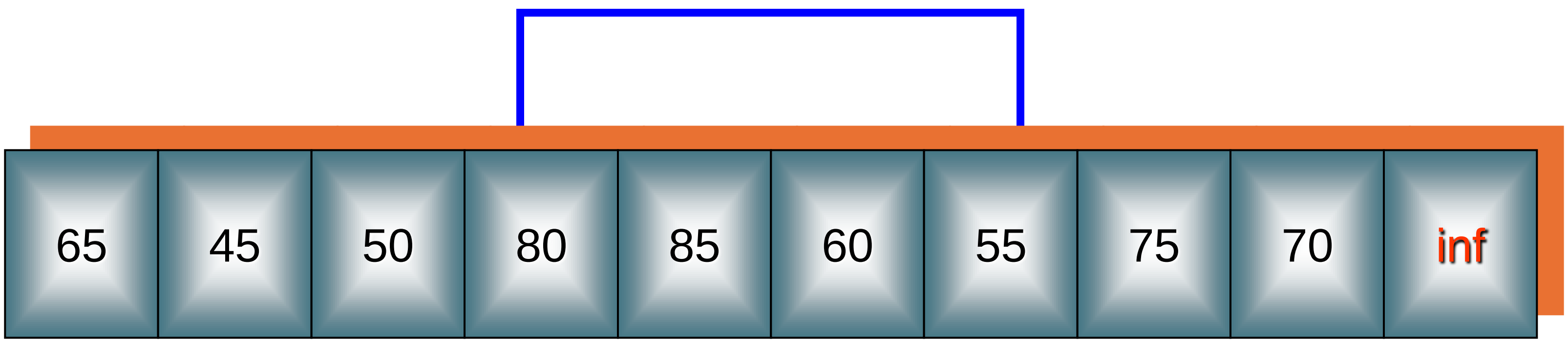
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	i	p
65	45	75	80	85	60	55	50	70	∞	2	9



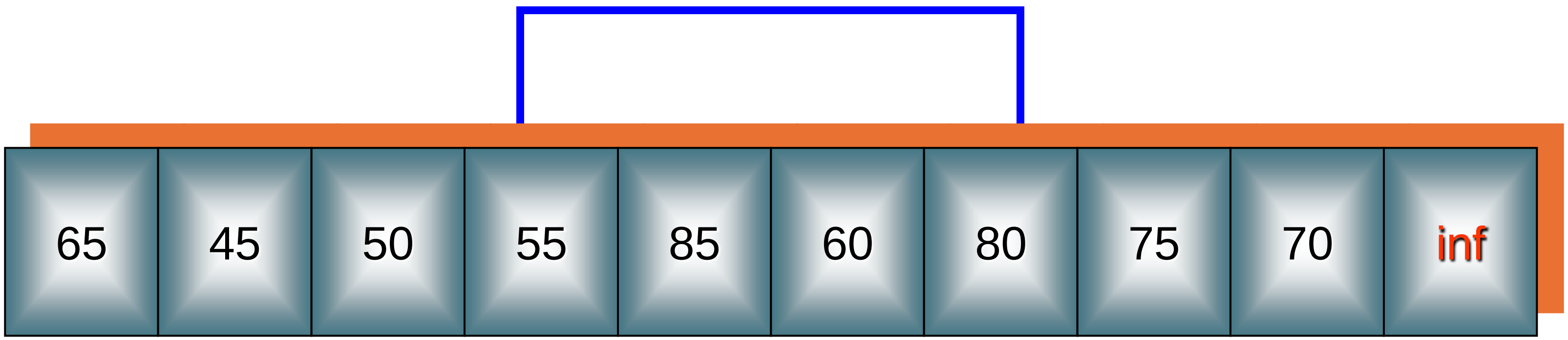
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	i	p
65	45	75	80	85	60	55	50	70	∞	3	8



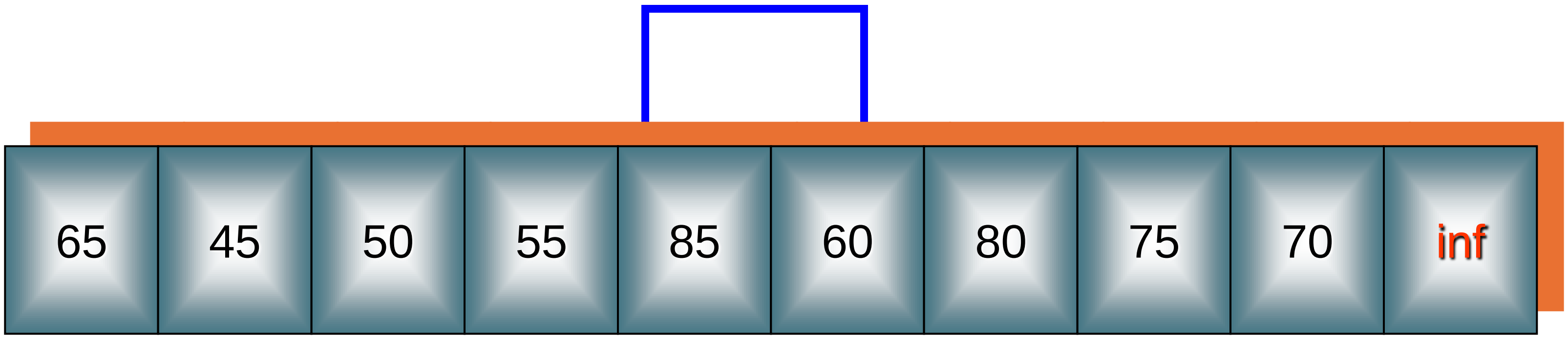
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	i	p
65	45	50	80	85	60	55	75	70	∞	3	8



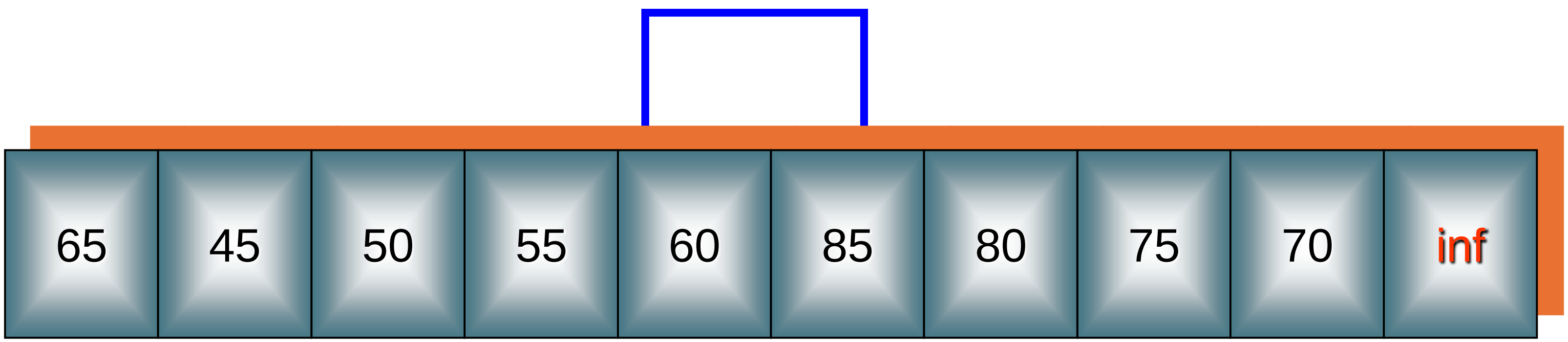
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	i	p
65	45	50	80	85	60	55	75	70	∞	4	7



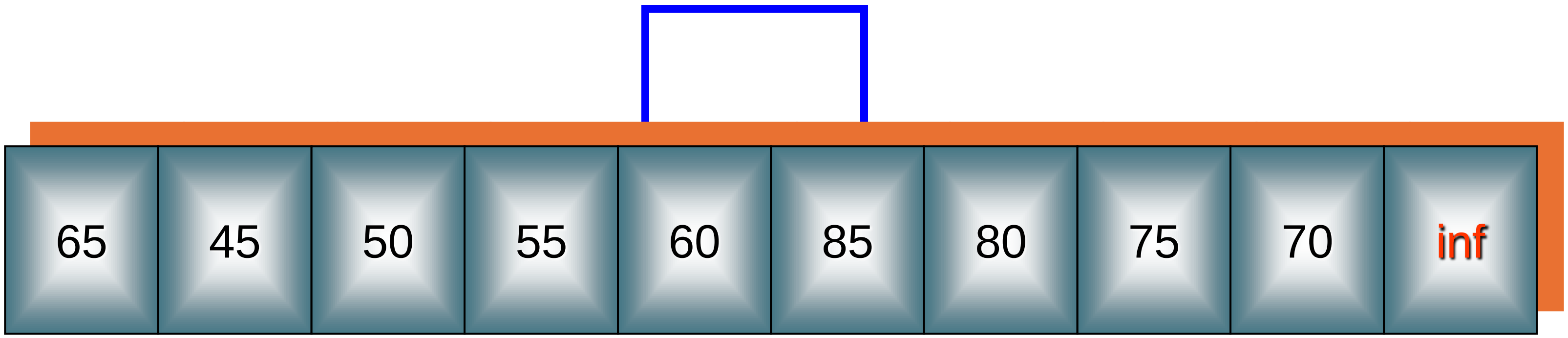
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	i	p
65	45	50	55	85	60	80	75	70	∞	4	7



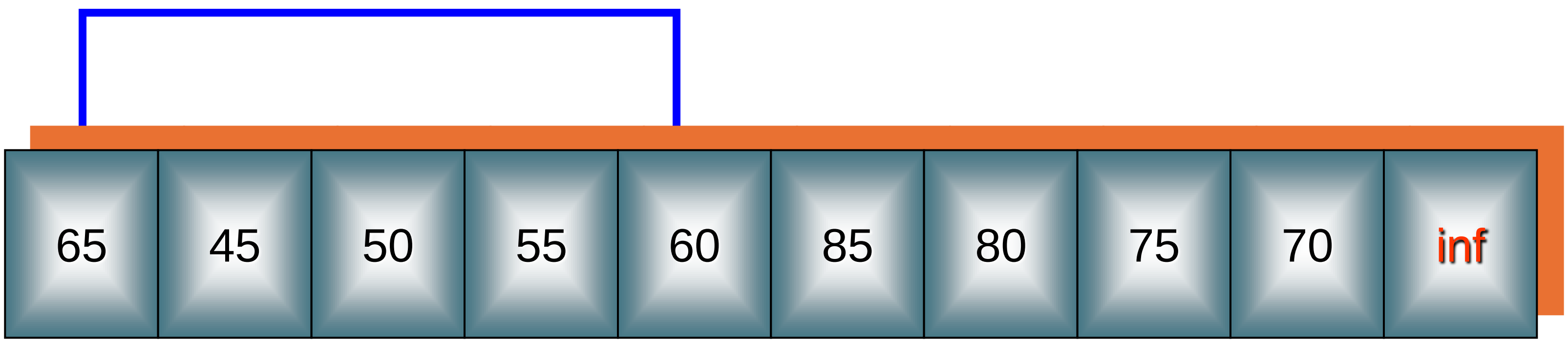
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	i	p
65	45	50	55	85	60	80	75	70	∞	5	6



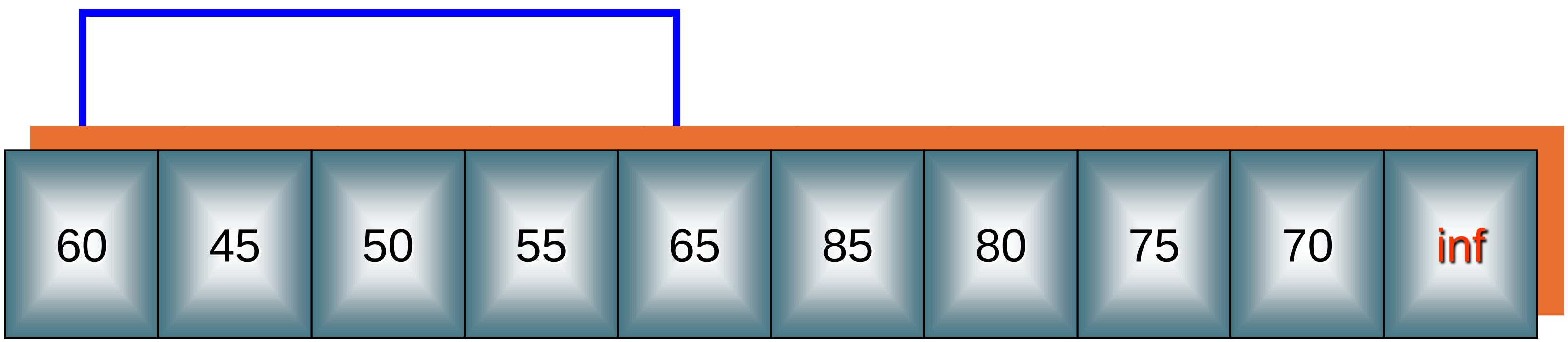
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	i	p
65	45	50	55	85	60	80	75	70	∞	5	6



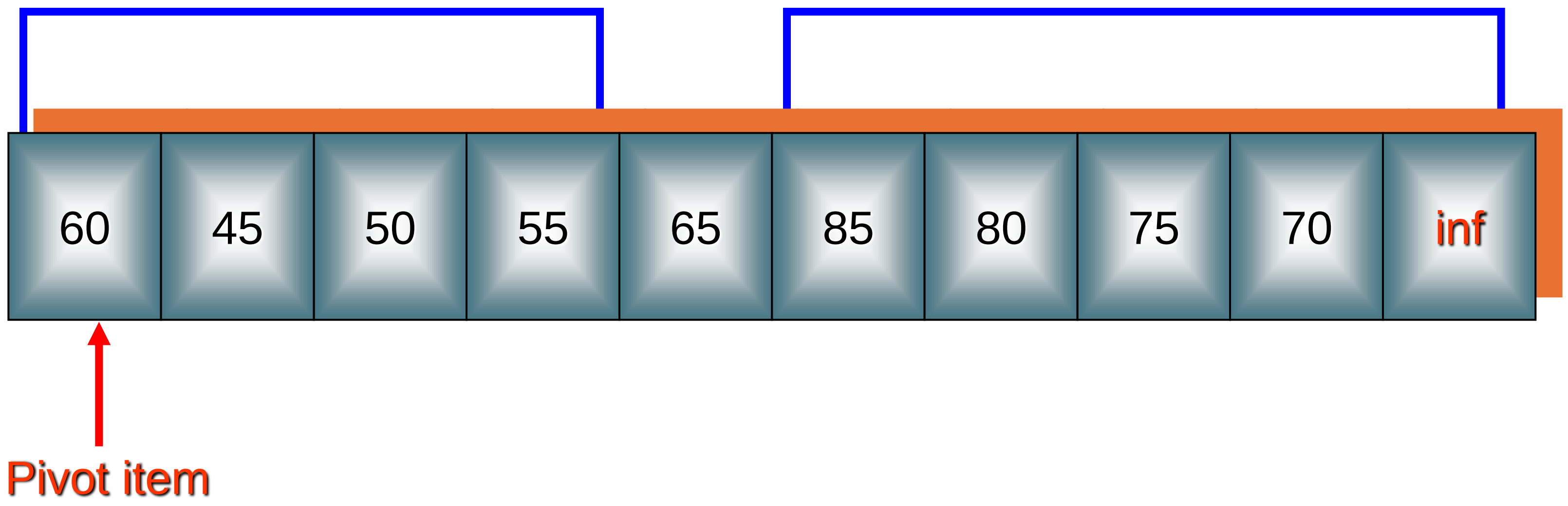
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	i	p
65	45	50	55	85	60	80	75	70	∞	6	5



(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	i	p
65	45	50	55	85	60	80	75	70	∞		



(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	i	p
60	45	50	55	85	65	80	75	70	∞		


60
45
50
55
65
85
80
75
70
inf
Pivot item

(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	i	p
65	70	75	80	85	60	55	50	45	∞	2	9
65	45	75	80	85	60	55	50	70	∞	3	8
65	45	50	80	85	60	55	75	70	∞	4	7
65	45	50	55	85	60	80	75	70	∞	5	6
65	45	50	55	60	85	80	75	70	∞	6	5
60	45	50	55	65	85	80	75	70			

- Selection of the pivot

- Ideally the pivot should be the median of the items being sorted. That is, half the items should be greater than the pivot, and half smaller.
- The worst situation results when a subarray with n elements is divided into one subarray with 1 element and the other with n-1 elements.
- There are two problems with it
  - Performance of algorithm $\Theta(n^2)$
  - Recursive function calls take space on the machine stack

Worst case analysis of QUICKSORT

- Assumptions:

    1. The n elements to be sorted are distinct

    2. The partitioning element in PARTITION is chosen with some random selection procedure.

    The number of element comparisons in each call of "PARTITION" is

    p-m+1 (distinct elements)

    p-m+2 (repeated elements)

Let r be the total number of elements comparisons required at any level of recursion-

- At level one, one call PARTITON(1, n+1); r = n;
- At level two, at most two calls are made and r = n-1;
- At each level of recursion O(r) elements comparisons are made so-

$$C_w(n) = \sum_{r=2}^{n} O(r) = O(n^2)$$

or $\Omega(n^2)$

Average case analysis of QUICKSORT

Average case value $C_A(n)$ is much less than its worst case, under the assumptions made earlier.

The partitioning element v in the call of PARTITION(m, p) has equal probability of being the i[th] , $1 \le i \le p - m$, smallest element, in A[m,p-1], hence two files remaining to be sorted will be A[m:j] and A[j+1:p-1] with probability 1/(p-m), for all $m \le j < p$.

From this we obtain the recurrence relation

$$C_A(n) = n + 1 + \frac{1}{n}\left(\sum_{1 \le k \le n} (C_A(k-1) + C_A(n-k))\right) \quad \ldots(1)$$

n+1 is the number of comparisons required by PARTITION on its first call

$$nC_A(n) = n(n+1) + 2(C_A(0) + C_A(1) + \ldots + C_A(n-1)) \quad \ldots(2)$$

replacing <u>n</u> by <u>n-1</u>we get

$$(n-1)C_A(n-1) = n(n-1) + 2(C_A(0) + C_A(1) + \ldots + C_A(n-2)) \quad \ldots(3)$$

subtracting (3) from (2) we get

$$nC_A(n) - (n-1)C_A(n-1) = 2n + 2(C_A(n-1))$$ or

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

repeatedly using this equation to substitute for C_A(n-1), C_A(n-2),… we get

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

$$= \frac{C_A(n-3)}{n-2} + \frac{2}{n-1} + \frac{2}{n} + \frac{2}{n+1}$$

$$= \frac{C_A(1)}{2} + 2 \sum_{3 \le k \le n+1} \frac{1}{k}$$

$$\sum_{3 \le k \le n+1} \frac{1}{k} \le \int_{2}^{n+1} \frac{1}{x} dx = \log_e(n+1) - \log_e 2$$

$$C_A(n) \le 2(n+1)[\log_e(n+1) - \log_e 2]$$

= $O(n \log n)$

$$= \frac{C_A(n-3)}{n-2} + \frac{2}{n-1} + \frac{2}{n} + \frac{2}{n+1}$$

$$= \frac{C_A(1)}{2} + 2 \sum_{3 \le k \le n+1} \frac{1}{k}$$

$$= \log_e(n+1) + \lambda - 3/2$$, where $\lambda \approx 0.577$

is known as Euler's constant

= $O(n \log n)$

Matrix Multiplication

Let A and B are two $n \times n$ matrices, the product C = AB is also a $n \times n$ matrix defined as-

$$C_{i,j} = \sum_{k=1}^{n} A_{i,k} B_{k,j}$$

As a divide and conquer solution of this problem ($n = 2^k$), for n = 2 the solution needs *8 multiplications and 4 additions* and complexity of it is $O(n^3)$.

Strassen Matrix Multiplication

Strassen discovered a way to compute C_{ij} using 7 multiplic-ations and 18 additions.

7 multiplication of size are computed as-

$$P = (A_{11}+A_{22})\ (B_{11}+B_{22})$$

$$Q = (A_{21}+A_{22})\ B_{11}$$

$$R = A_{11}\ (B_{12}-B_{22})$$

$$S = A_{22}\ (B_{21}-B_{11})$$

$$T = (A_{11}+A_{12})\ B_{22}$$

$$U = (A_{21}-A_{11})\ (B_{11}+B_{12})$$

$$V = (A_{12}-A_{22})\ (B_{21}+B_{22})$$

and then

C_{11} = P+S-T+V

C_{12} = R+T

C_{21} = Q+S

C_{22} = P+R-Q+U

required time T(n) is given by

$$T(n) = \begin{cases} 7T(n/2) + an^2, & n > 2 \\ b, & n \leq 2 \end{cases}$$

$$T(n) = 7T(n/2) + an^2$$

$$= 7[7T(n/4) + an^2/4] + an^2$$

$$\vdots$$

$$= 7^k T(1) + an^2((7/4)^0 + (7/4)^1 + \ldots + (7/4)^{k-1})$$

$$= 7^{\log n} + an^2\left[\frac{(7/4)^k - 1}{(7/4) - 1}\right]$$

$$\leq 7^{\log n} + cn^2[(7/4)^k]$$

$$= cn^{\log 4} (7/4)^{\log n} + 7^{\log n}$$

$$= cn^{\log 4}[n^{\log(7/4)}] + n^{\log 7}$$

$$= cn^{\log 4}\, n^{\log 7 - \log 4} + n^{\log 7}$$

$$= cn^{\log 4 + \log 7 - \log 4} + n^{\log 7}$$

$$= O(n^{\log 7}) = O(n^{2.81})$$

Victor Pan improved time to $O(n^{2.681})$ & then $O(n^{2.496})$

End of Chapter 3