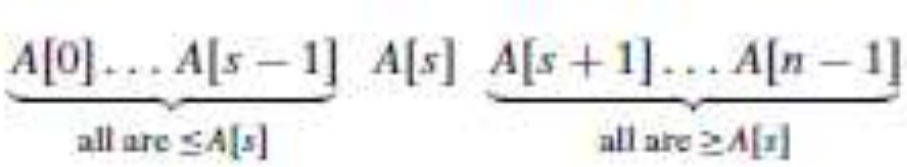
Unit II – Divide and Conquer

- Merge sort
- **Quick sort**
- Binary search
- Multiplication of large Integers
- Strassen's Matrix Multiplication

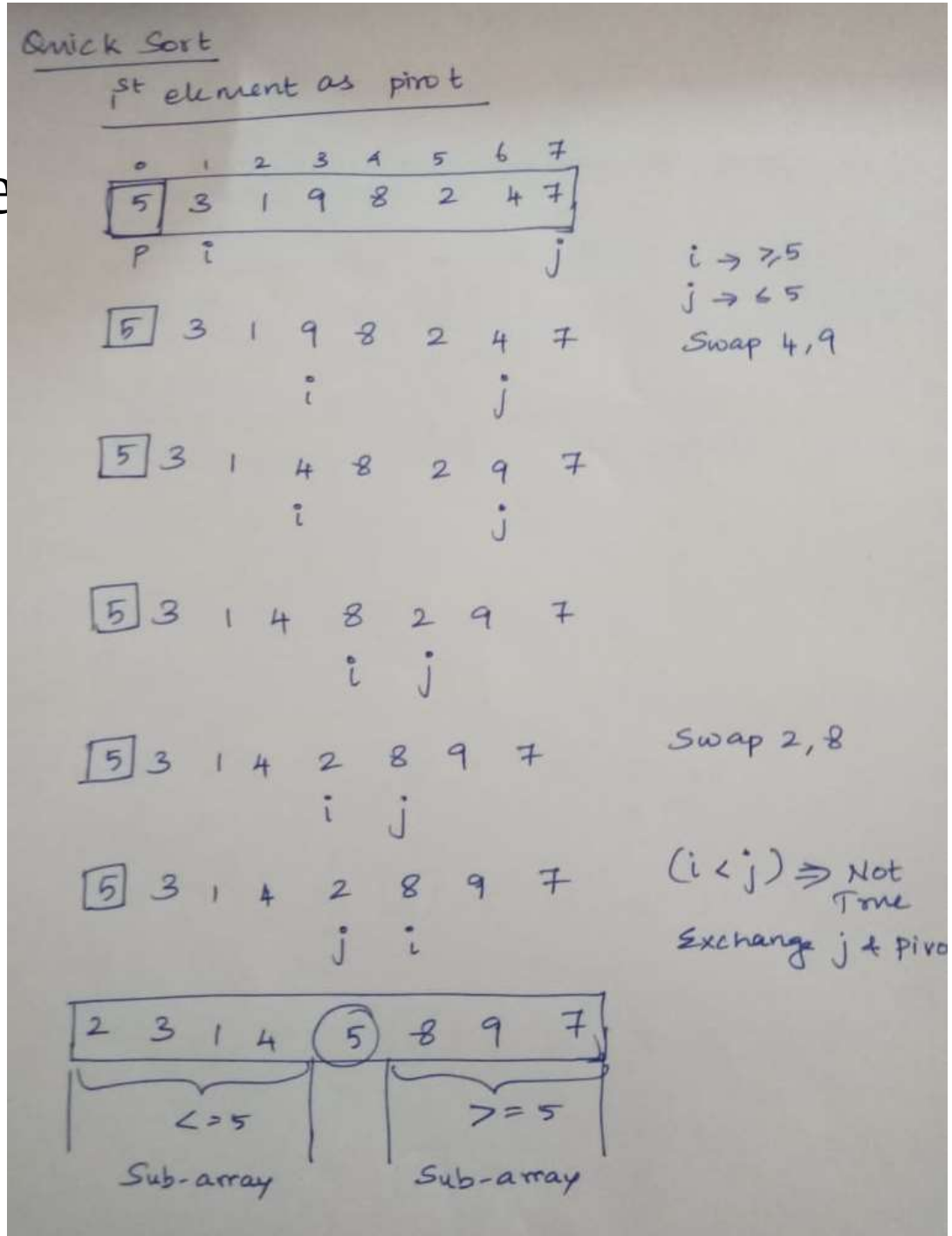
QUICK SORT

- Partition the array such that all elements before the position are smaller than or equal to A[s] and all the elements after the position s are greater than or equal to A[s]



- Divide the array based an element selection – pivot
- Pivot selection can be done as follows
  - First element as pivot
  - Last element as pivot
  - Middle element as pivot
  - Random number as pivot

- Example


Quick Sort
1st element as pivot
0 1 2 3 4 5 6 7
5 3 1 9 8 2 4 7
p i j
i → ≥5
j → <5
5 3 1 9 8 2 4 7
i j
Swap 4,9
5 3 1 4 8 2 9 7
i j
5 3 1 4 8 2 9 7
i j
Swap 2,8
5 3 1 4 2 8 9 7
i j
5 3 1 4 2 8 9 7
j i
(i < j) ⇒ Not True
Exchange j & Pivo
2 3 1 4 5 8 9 7
<=5
>=5
Sub-array
Sub-array

QUICK SORT ALGORITHM

```
Quicksort (l,h)
{
   if ( l < h)
   {
         j= partition (l,h);
        Quicksort (l,j);
        Quicksort (j+1,h);
    }
}
```

```
Partition (l,h)
{
pivot=A[l];
i=l+1;
j=h;
while(i<j)
{    do
     {  i++; } while(A[i]<=pivot);
    do
     {  j--; } while(A[j]>pivot);
    if i<j
         swap(A[i],A[j]);
} swap(A[l], A[j]);
return j;
}
```

QUICK SORT ANALYSIS

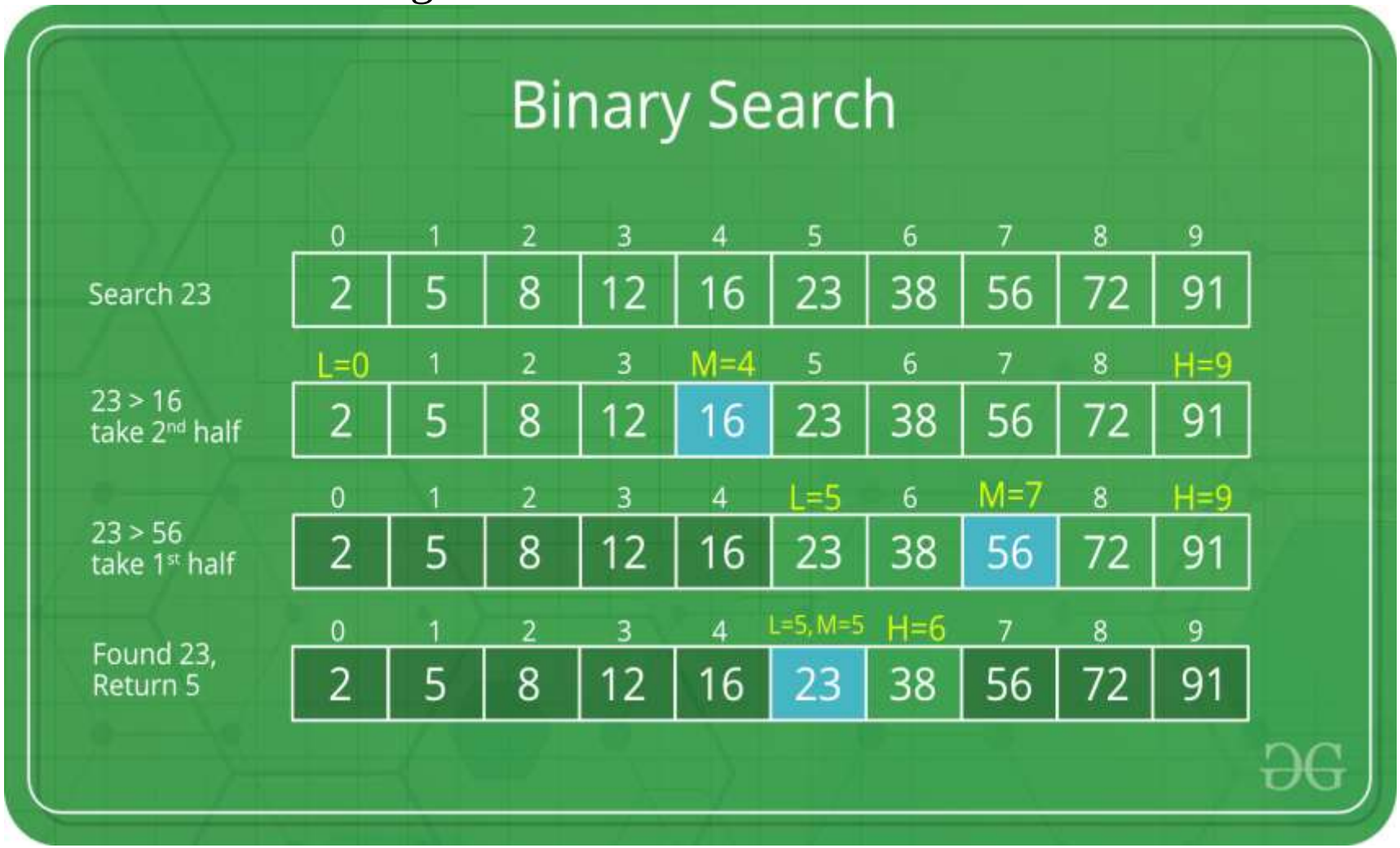
- $n/2/ 2 /2 \rightarrow n/2^k$
- $n/2^k = 1$
- $n = 2^k$
- $K = \log_2 n \rightarrow$ No. of levels
- Time for partitioning $\rightarrow n$
- So, the time complexity is **$O(n \log_2 n)$**
- Best case - $O(n \log_2 n)$, Average case - $O(n \log_2 n)$
- Worst case – sorted array , always partitioning happens at the beginning $O(n^2)$
- So to overcome the worst case of quick sort always select ***middle*** element as pivot and select a ***random*** element as pivot.

Unit II – Divide and Conquer

- Merge sort
- Quick sort
- **Binary search**
- Multiplication of large Integers
- Strassen's Matrix Multiplication

Binary Search

- Efficient searching method



Binary Search – Algorithm

```
BinarySearch(A[0..n-1],K)
l←0; r←n-1
while l<=r do
    m←(l+r)/2
    if k=A[m] return m
    else if k<A[m] r←m-1
    else l←m+1
return -1
```

Analysis

Worst case - $O(\log_2 n)$

Best case - $O(1)$

Average case - $O(\log_2 n)$

Binary Search – Analysis

- ***Analysis***

$C(n) = C(n/2) + 1$

Sublist + middle element

Assume $n = 2^k$, $k = \log_2 n$

$C(2^k) = C(2^k/2^1) + 1$

$= C(2^{k-1}) + 1$

Using Backward substitution

$C(2^k) = [C(2^{k-2})+1] + 1$ $\quad C(2^{k-1}) = C(2^{k-2})+1$

$C(2^k) = C(2^{k-2})+2$

$C(2^k) = C(2^{k-3})+3$

$C(2^k) = C(2^{k-4})+4$

$C(2^k) = C(2^{k-k})+k$

$C(2^k) = C(2^{k-k})+k$

$C(2^k) = C(2^0)+k \rightarrow C(1) + \log_2 n \rightarrow 1+\log_2 n \rightarrow \mathbf{\log_2 n}$