- **INSERATION SORT:**

Which uses an incremental approach

The Role of Algorithms in Computing:

1. What are algorithms?
2. Why is the study of algorithms worthwhile?
3. What is the role of algorithms relative to other computer technologies?

Ans .

Informally, an algorithm is any well-defined computational step that takes some value or set of values as input and produces some value or set of values as output. An algorithm is thus a sequence of computational steps that transform the input into the output.

E.g.1 Sort a sequence of numbers into non-decreasing order

Input:

A sequence of n numbers $(a_1 + a_2 + \cdots + a_n)$

Output:

A permutation (re-ordering) $(a_1 + a_2 + \cdots + a_n)$ of input sequence such as that $(a_1 \leq a_2 \leq \cdots \leq a_n)$

E.g-2

Input: $(31,41,59,26,41,58)$

output:$(26,31,41,41,58,59)$

- **Data Structure:** A data structure is a way to store and organize data to facilitate access and modification
  $A = (31,41,59,26,41,58)$
  $=> 31,41,59,26,41,58$
  $=> 31,41,26,59,41,58$
  $=> 26,31,41,59,41,58$
  $=> 26,31,41,41,59,58$
  $=> 26,31,41,41,58,59$

```
1.    $for\ j = 2\ to\ A.length\ do$
2.    $key = A[j]$
3.    $i = j - 1$
4.    $while\ i > 0\ and\ A[i] > key$
5.    $A[i+1] = A[i]$
6.    $i = i\ \neq 1$
7.    $A[i+1] = key$
```

Find

i. Linear Searching
ii. Max
iii. Min

Input : $A = (10,15,5,6,7,8,25,2,1)$

Pseudocode for Linear Searching

```
1.    $V\ key\ to\ search$
2.    $for\ i = 1\ to\ A.length$
3.    $if\ A[i] = V$
4.    $return\ i$
5.    $end\ if$
6.    $end\ for$
7.    $return\ NIL$
```

Time complexity

- Number of loops 1 which runs from 1 to n times therefore

$$n\ times$$

- Number of memory access 1 on line number 3 i.e. $A[i]$

$$1\ access$$

Let :

$$T(n) = \sum_{i=1}^{n} i\ \ 1 = 1\ \sum_{i=1}^{n} i = 1 * n = n$$

Hence

$$\boldsymbol{\theta(n)}$$

Pseudocode for finding Max

```
1. $\max \leftarrow A[i]$
2. $for\ i \leftarrow 2\ to\ n$
3. $if \max < A[i]$
```

First line memory access 1 $time$

Summation of for i.e.

$$\sum_{i=2}^{n} i\ \ 2$$

Let

$$T(n) =\ 1\left(\sum_{i}^{n} i\ \ 2\right) = 2\sum_{i}^{n} i\ = 2n$$

```
4. $do$
5. $\max \leftarrow A[i]$
6. $end\ if$
7. $end\ for$
8. $output\ max$
```

Pseudocode for finding Min

```
1. $\min \leftarrow A[i]$
2. $for\ i \leftarrow 2\ to\ n$
3. $if \min > A[i]$
4. $do$
5. $\min \leftarrow A[i]$
6. $end\ if$
7. $end\ for$
8. $output\ max$
```

First line memory access 1 $time$

Summation of for i.e.

$$\sum_{i=2}^{n} i\ 2$$

Let

$$T(n) = 1 \left(\sum_{i=2}^{n} i\ 2 \right) = 2 \sum_{i=2}^{n} i\ = 2n$$

Hence in terms of n

$$\theta(n)$$

- **2-diamential Maxima**

See lecture slides

Time analysis of 2-diamential Maxima

Pseudocode

Assume it's a function

```
    $MAXIMA(int\ n, Point\ P[1 \dots n])$
1.  $for\ i \leftarrow 1\ to\ n$                                                    $n\ times$
2.  $do\ maximal \leftarrow true$
3.          $for\ j \leftarrow 1\ to\ n$                                            $n\ times$
4.          $do$
5.          $if\ (i \neq j)\ and\ (P[i].x\ \leq P[j].x)\ and\ (P[i].y\ \leq P[j].y)$    $4\ access$
6.                  $then\ maximal \leftarrow false$
7.                  $break$
8.          $end\ if$
9.          $end\ for$
10. $if\ maximal$
11. $then\ output\ P[i].x, P[i].y$                                              $2\ access$
12. $end\ if$
13. $end\ for$
```

In terms of $T(n)$

1. Calculate inner loop i.e. on line no. 3, Lets say $T(I)$

$$I = \sum_{j=1}^{n} 4 = 4\sum_{j=1}^{n} j\ = 4n$$

For outer for loop lets say $T(M)$

$$M = \sum_{i=1}^{n}(2 + T(I)) = \sum_{i=1}^{n}(2 + 4n) = 2\sum_{i=1}^{n} + 4n\sum_{i=1}^{n} = 2\sum_{i=1}^{n} + 4n^2$$

$$= 2n + \ 4n^2 => \ 4n^2 + 2n$$

In terms of n Hence the largest n would be the worst-time case i.e.

$$\theta(n^2)$$

- **<u>Complexity of Algorithm</u>**

We calculate the complexity on running time and memory access. There are three cases of the running time

i. Worst-case

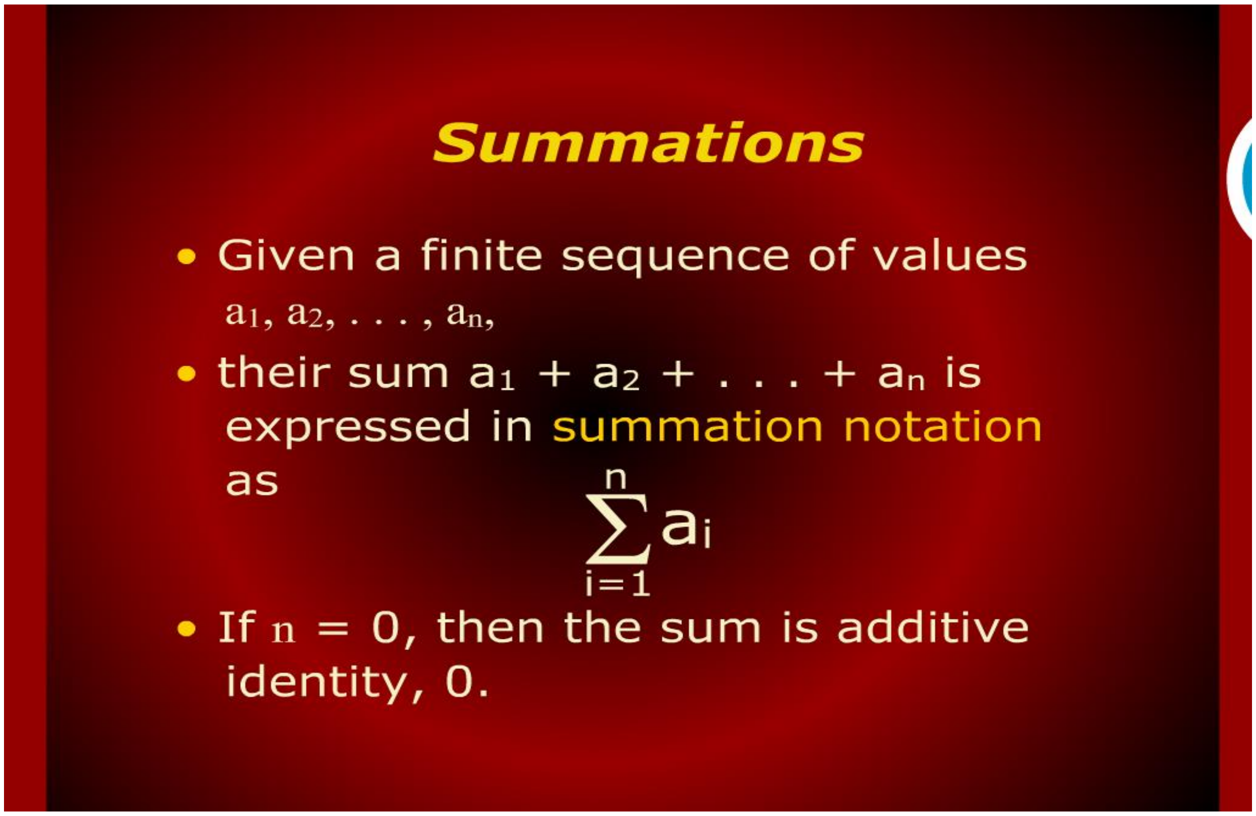
ii. Average-case
iii. Best-case

For the definition check lecture slides

There are here steps to calculate complexity

i. Nos. of steps
   Or
ii. Nos. of comparisons
   Or
iii. Nos. of Access of Array and running time of loops

Summations rules

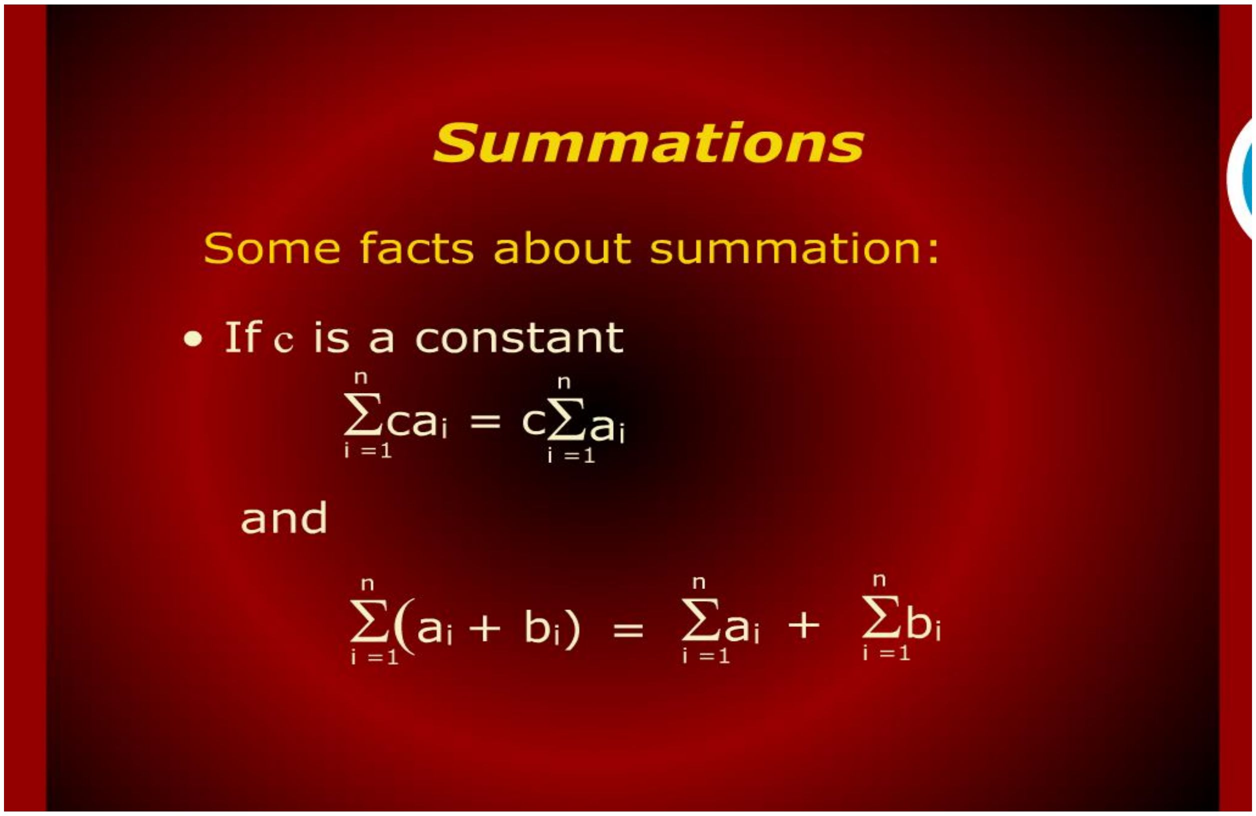


Summations

- Given a finite sequence of values $a_1, a_2, \ldots, a_n$,
- their sum $a_1 + a_2 + \ldots + a_n$ is expressed in summation notation as

$$\sum_{i=1}^{n} a_i$$

- If $n = 0$, then the sum is additive identity, 0.



Summations

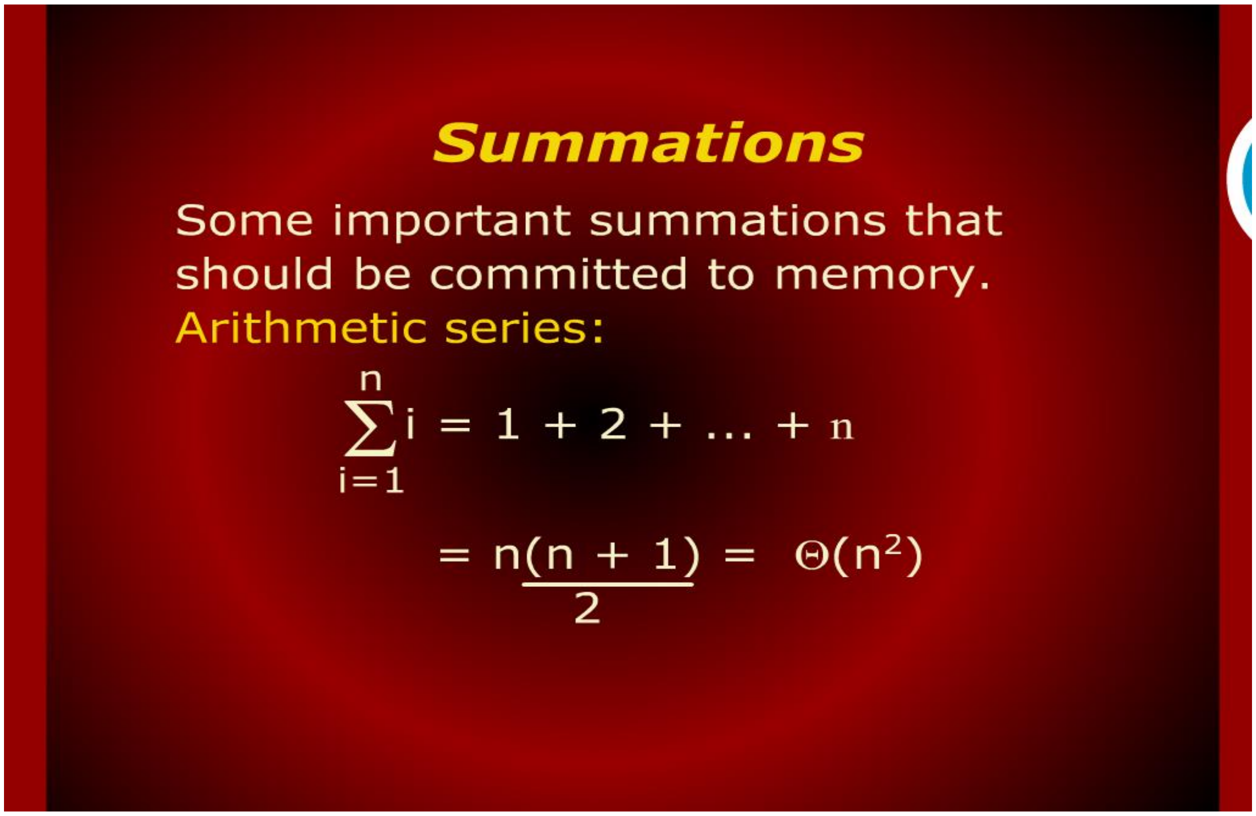
Some facts about summation:

- If c is a constant

$$\sum_{i=1}^{n} ca_i = c\sum_{i=1}^{n} a_i$$

and

$$\sum_{i=1}^{n}(a_i + b_i) = \sum_{i=1}^{n} a_i + \sum_{i=1}^{n} b_i$$



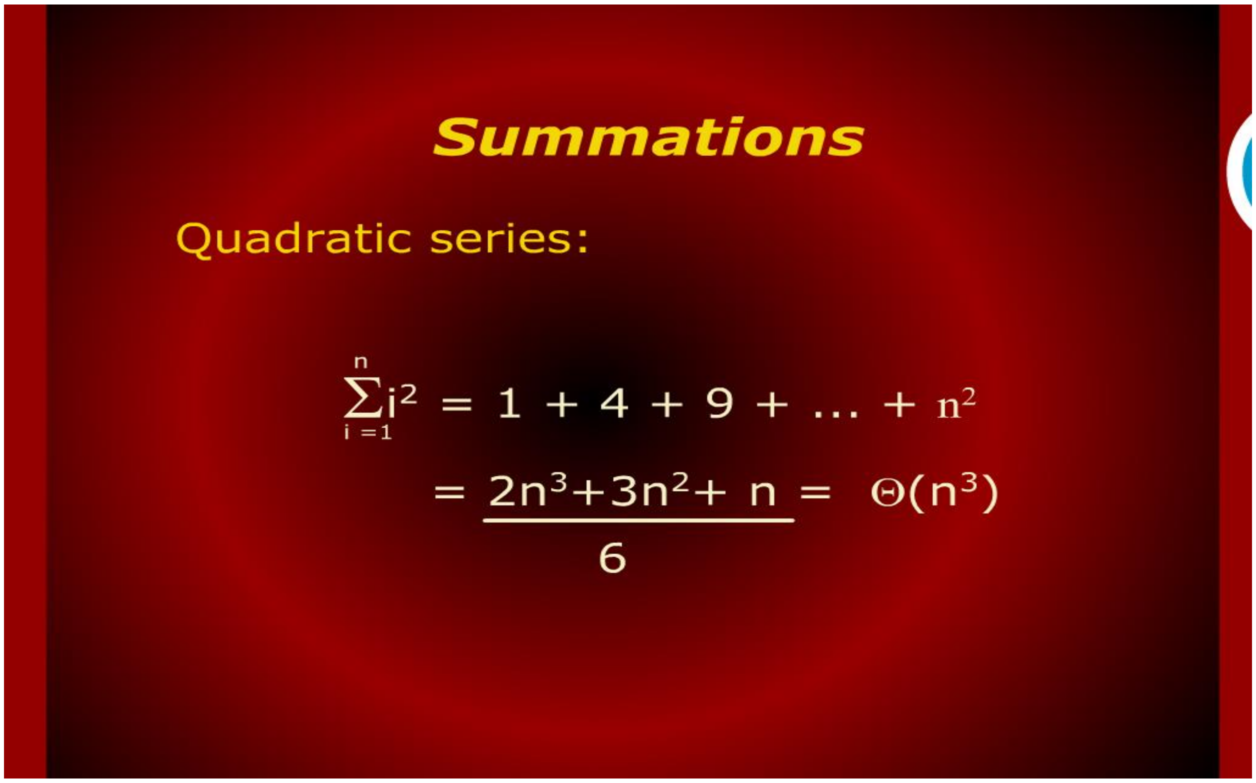
Summations

Some important summations that should be committed to memory.

Arithmetic series:

$$\sum_{i=1}^{n} i = 1 + 2 + \ldots + n$$

$$= \frac{n(n+1)}{2} = \Theta(n^2)$$

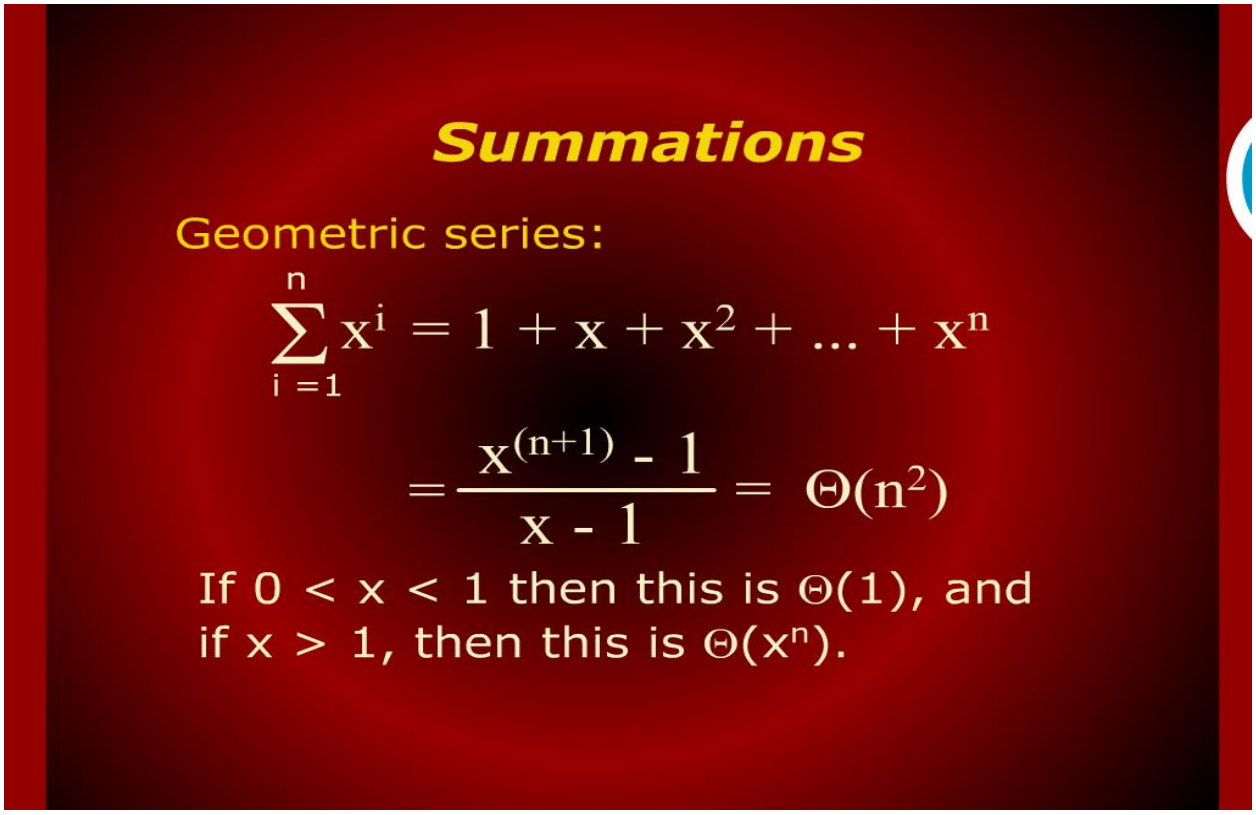


Summations

Quadratic series:

$$\sum_{i=1}^{n} i^2 = 1 + 4 + 9 + \ldots + n^2$$

$$= \frac{2n^3+3n^2+n}{6} = \Theta(n^3)$$



Summations

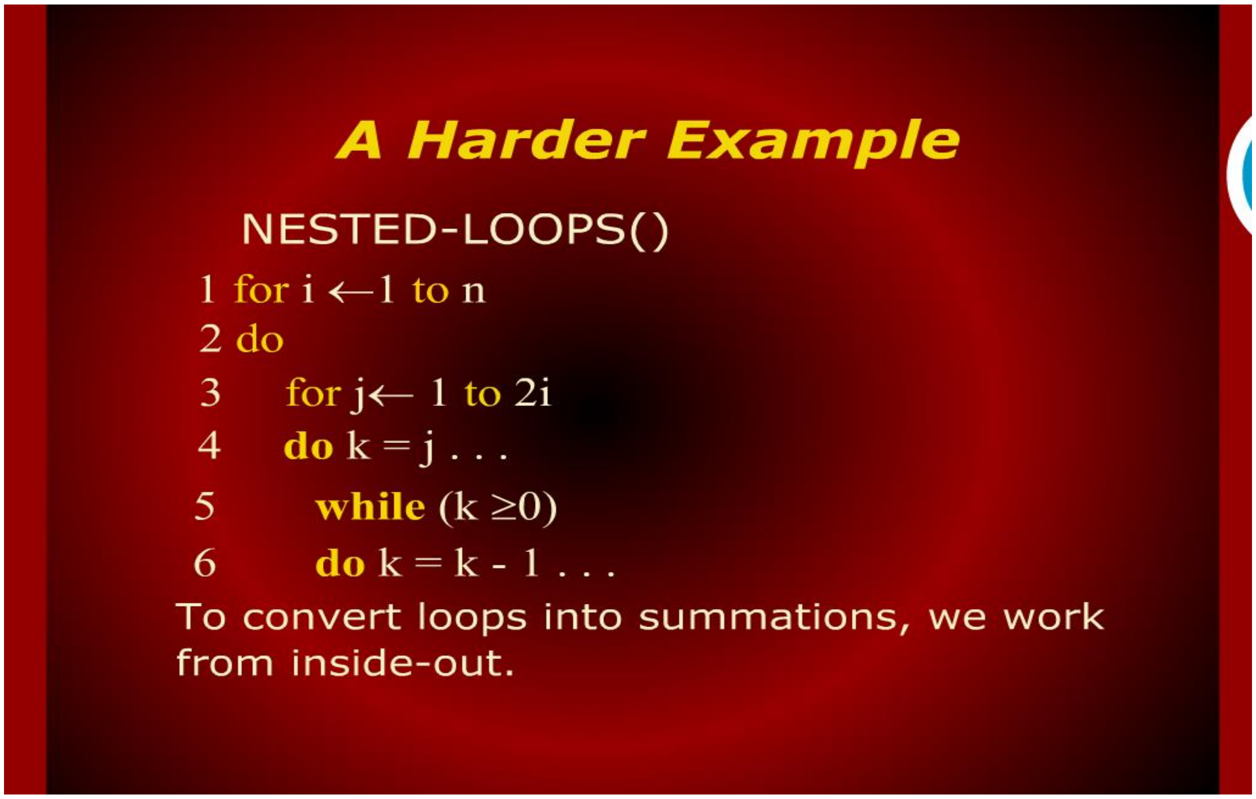
Geometric series:

$$\sum_{i=1}^{n} x^i = 1 + x + x^2 + \ldots + x^n$$

$$= \frac{x^{(n+1)} - 1}{x - 1} = \Theta(n^2)$$

If $0 < x < 1$ then this is $\Theta(1)$, and if $x > 1$, then this is $\Theta(x^n)$.

A HARDER EXAMPLE



A Harder Example

```
NESTED-LOOPS()
1 for i ← 1 to n
2 do
3     for j ← 1 to 2i
4     do k = j . . .
5         while (k ≥ 0)
6         do k = k - 1 . . .
```

To convert loops into summations, we work from inside-out.

Lets first solve inner loop i.e. from line no. 4 to 6

As we know the while loop based on the value of k where $k = j$

Lets say $T(J)$ for while loop

$$J = \sum_{k=0}^{j} 1 = j + 1$$

Hence $T(J) = j + 1$

Now calculate the inner for loop i.e. from 3 to 6

Lets say

$$T(I) = \sum_{j=1}^{2i} T(J) = \sum_{j=1}^{2i} (j + 1)$$

$$= \sum_{j=1}^{2i} j + \sum_{j=1}^{2i} 1$$

$$= \sum_{j=1}^{2i} j + 2i - - - - - - - - - eq1$$

Now calculate summation of j by applying summation rule arithmetic series we get:

$$\sum_{j=1}^{2i} j = \frac{n(n+1)}{2}\ in\ our\ case\ n = 2i\ so\ equation\ would\ be$$

$$= \frac{2i(2i+1)}{2}\ please\ this\ value\ in\ the\ eq1\ we\ get$$

$$T(I) = \frac{2i(2i+1)}{2} + 2i$$

Solve the above equation we get

$$T(I) = \frac{\cancel{2}i(2i+1)}{\cancel{2}} + 2i = 2i^2 + i + 2i = 2i^2 + 3i$$

Hence running time of inner loop is

$$T(I) = 2i^2 + 3i$$

Now calculate the outer-most loop i.e. from line no. 1 to so on.

$$T(n) = \sum_{i=1}^{n} T(I) = \sum_{i=1}^{n} (2i^2 + 3i)$$

$$T(n) = 2\sum_{i=1}^{n} i^2 + 3\sum_{i=1}^{n} i$$

$$T(n) = 2\sum_{i=1}^{n} i^2 + 3\frac{n(n+1)}{2}$$

$$T(n) = 2\frac{2n^3 + 3n^2 + n}{6} + 3\frac{n(n+1)}{2}$$

$$T(n) = \frac{4n^3 + 6n^2 + 2n}{6} + 3\,\frac{n(n+1)}{2} = \frac{4n^3 + 6n^2 + 2n}{6} + 3\,\frac{n^2 + n}{2}$$

$$T(n) = \frac{4n^3 + 6n^2 + 2n}{6} + \frac{3n^2 + 3n}{2}$$

Remember the LCM in above case it would be 6

So

$$T(n) = \frac{4n^3 + 6n^2 + 2n}{6} + \frac{9n^2 + 9n}{6} = \frac{4n^3 + 6n^2 + 2n + 9n^2 + 9n}{6}$$

$$T(n) = \frac{4n^3 + 15n^2 + 11n}{6}$$

Hence the highest value of n is n^3

Thus

$$\boldsymbol{\theta}(\boldsymbol{n^3})$$

- **Plane Sweep Algorithm on 2-dimentional Maxima**

See lecture slides Lecture-5

- **Asymptotic Notation**

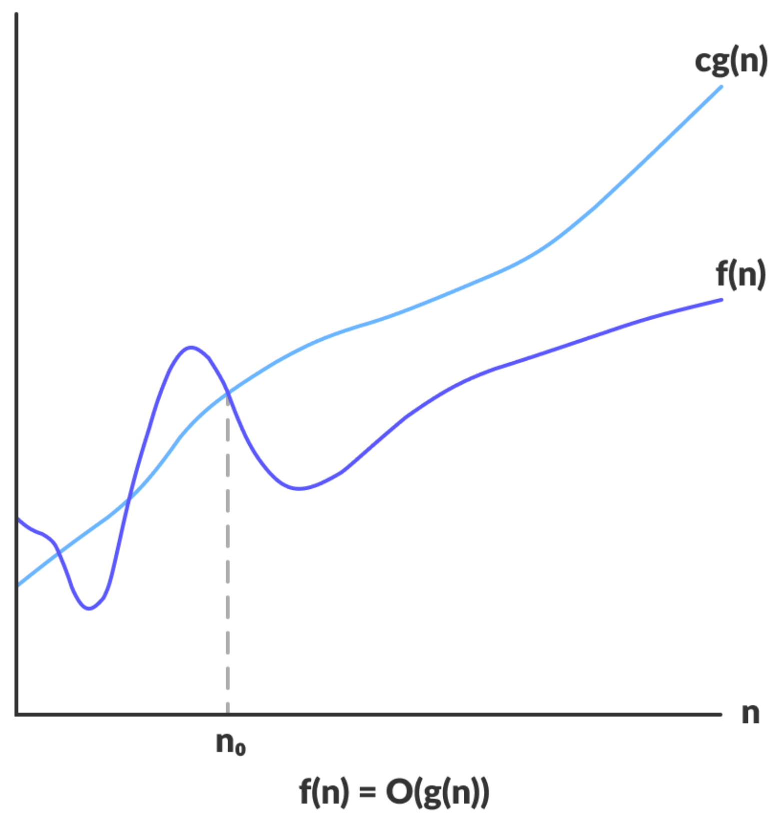
Asymptotic notations are the mathematical notations used to describe the running time of an algorithm when the input tends towards a particular value or a limiting value.

We can say equivalent of two functions coming/raising together but not meet (very near).

There are mainly three asymptotic notations:

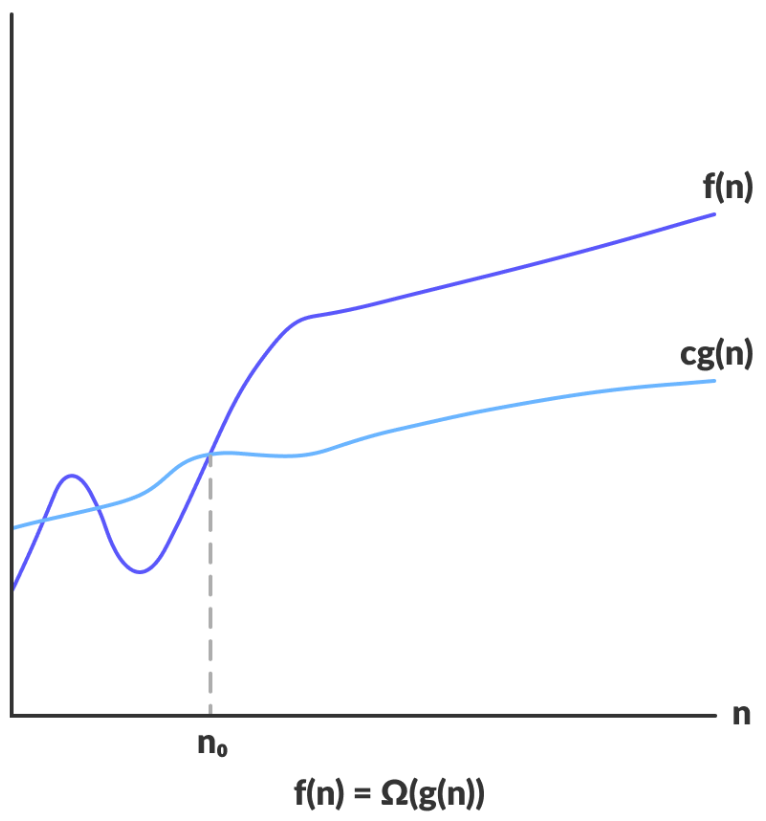
1. Big-O notation
2. Omega notation
3. Theta notation

1. **Big-O notation**



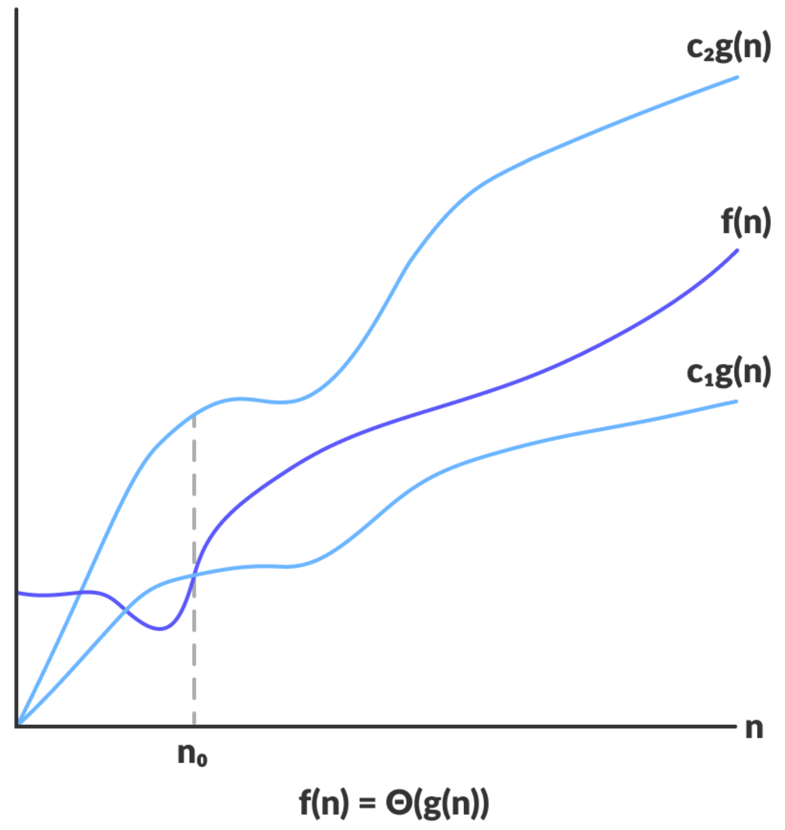
Big-O notation represents the upper bound of the running time of an algorithm. Thus, it gives the worst-case complexity of an algorithm.

2. **Omega Notation (Ω-notation)**



Omega notation represents the lower bound of the running time of an algorithm. Thus, it provides the best case complexity of an algorithm.

3. **Theta Notation (Θ-notation)**



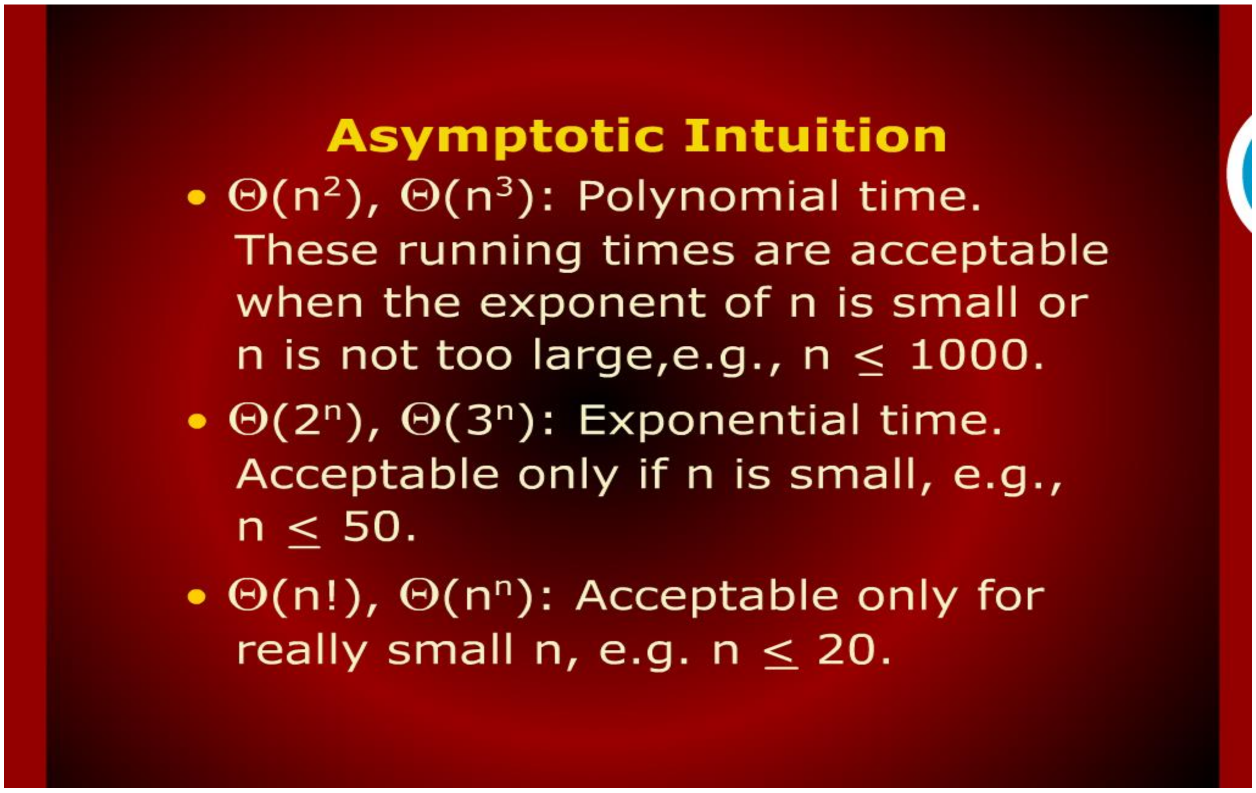
Theta notation encloses the function from above and below. Since it represents the upper and the lower bound of the running time of an algorithm, it is used for analyzing the average-case complexity of an algorithm.

Asymptotic Intuition

Here is a list of common asymptotic running times:

- $\Theta(1)$: Constant time; can't beat it!
- $\Theta(\log n)$: Inserting into a balanced binary tree; time to find an item in a sorted array of length n using binary search.
- $\Theta(n)$: About the fastest that an algorithm can run.

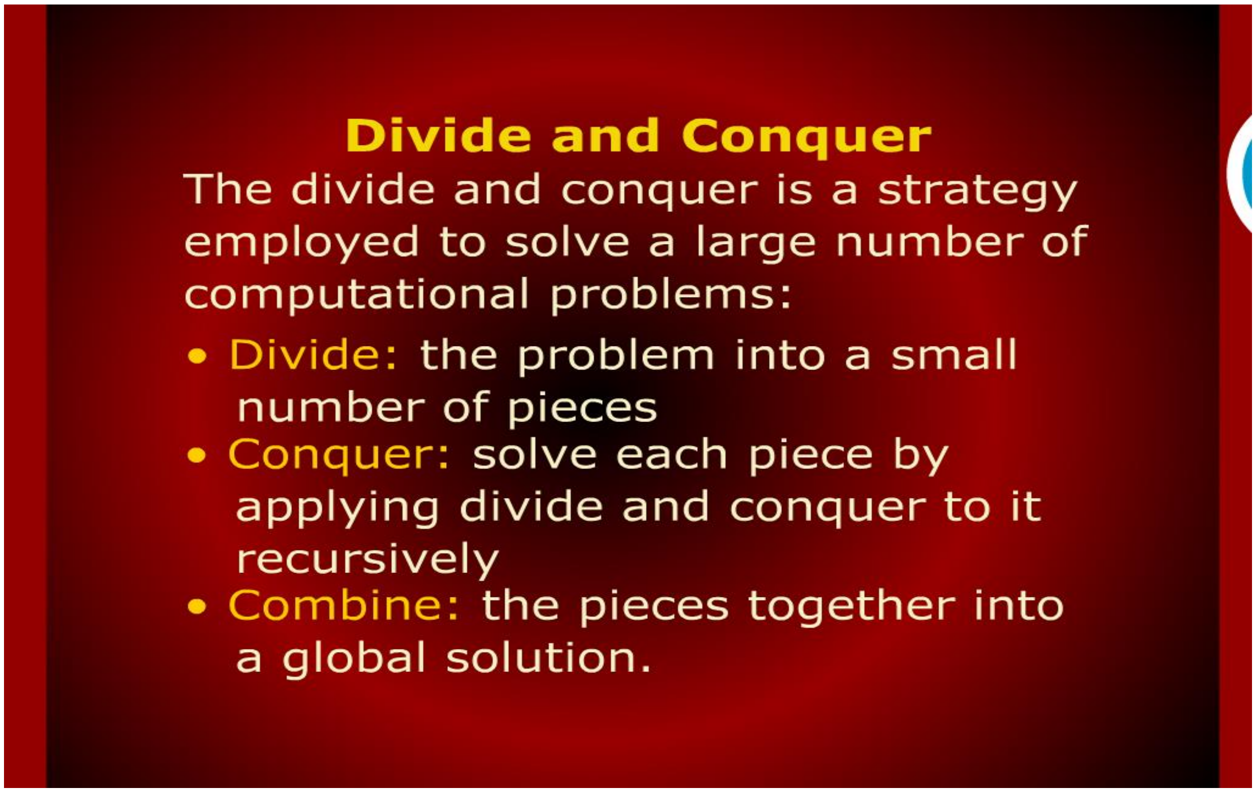
- $\Theta(n \log n)$: Best sorting algorithms.



Asymptotic Intuition

- $\Theta(n^2)$, $\Theta(n^3)$: Polynomial time. These running times are acceptable when the exponent of n is small or n is not too large,e.g., $n \leq 1000$.
- $\Theta(2^n)$, $\Theta(3^n)$: Exponential time. Acceptable only if n is small, e.g., $n \leq 50$.
- $\Theta(n!)$, $\Theta(n^n)$: Acceptable only for really small n, e.g. $n \leq 20$.

- **Divide and Conquer Algorithm.**



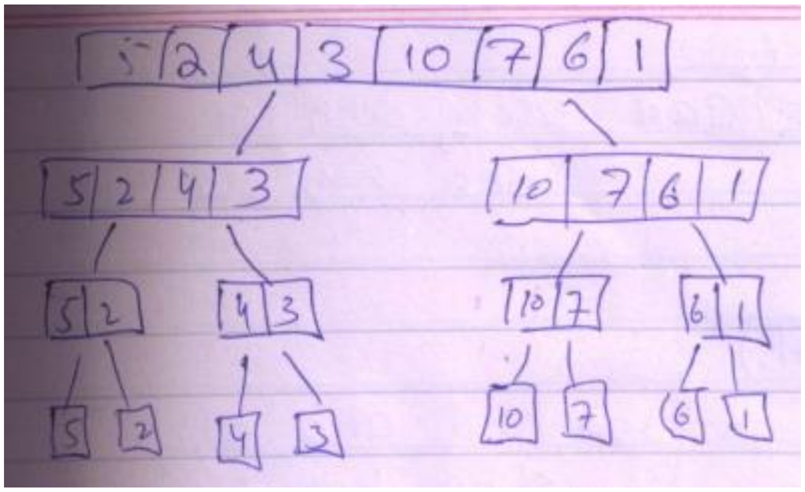
Divide and Conquer

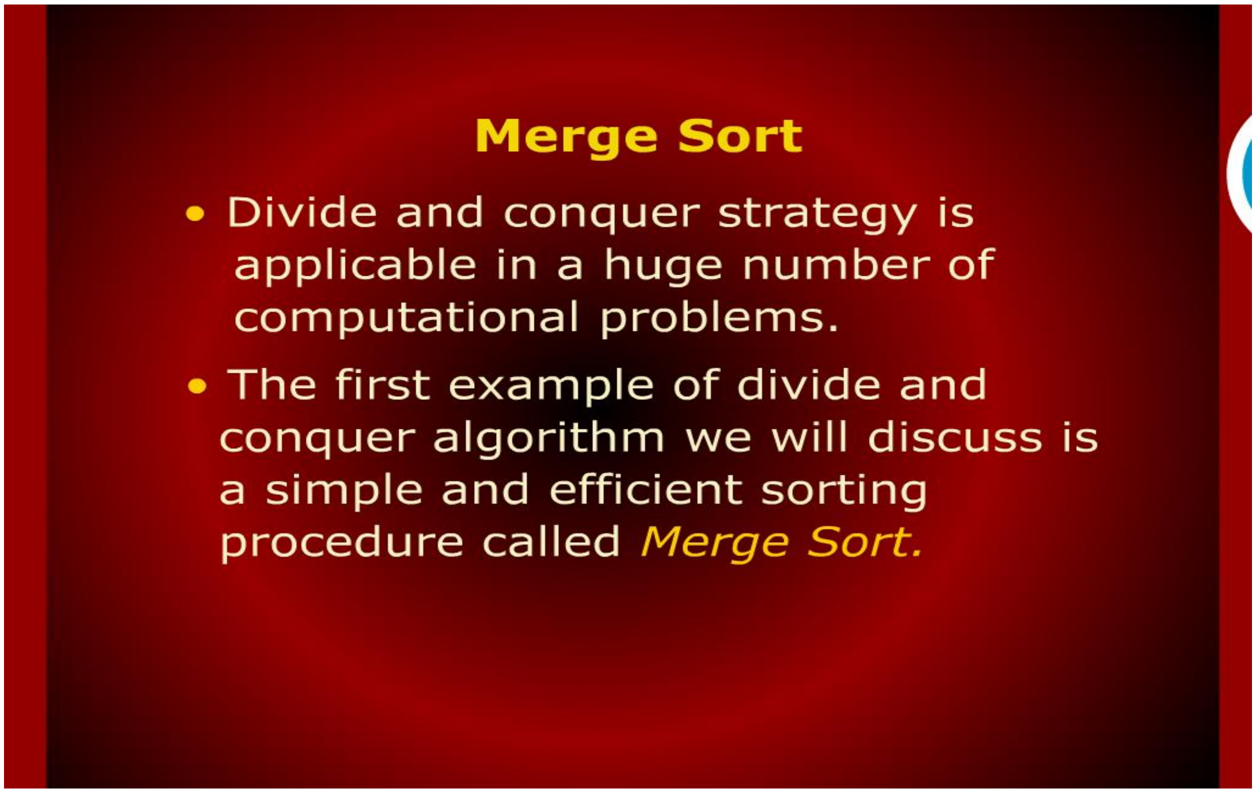
The divide and conquer is a strategy employed to solve a large number of computational problems:

- Divide: the problem into a small number of pieces
- Conquer: solve each piece by applying divide and conquer to it recursively
- Combine: the pieces together into a global solution.

- **MERGE SORT:**

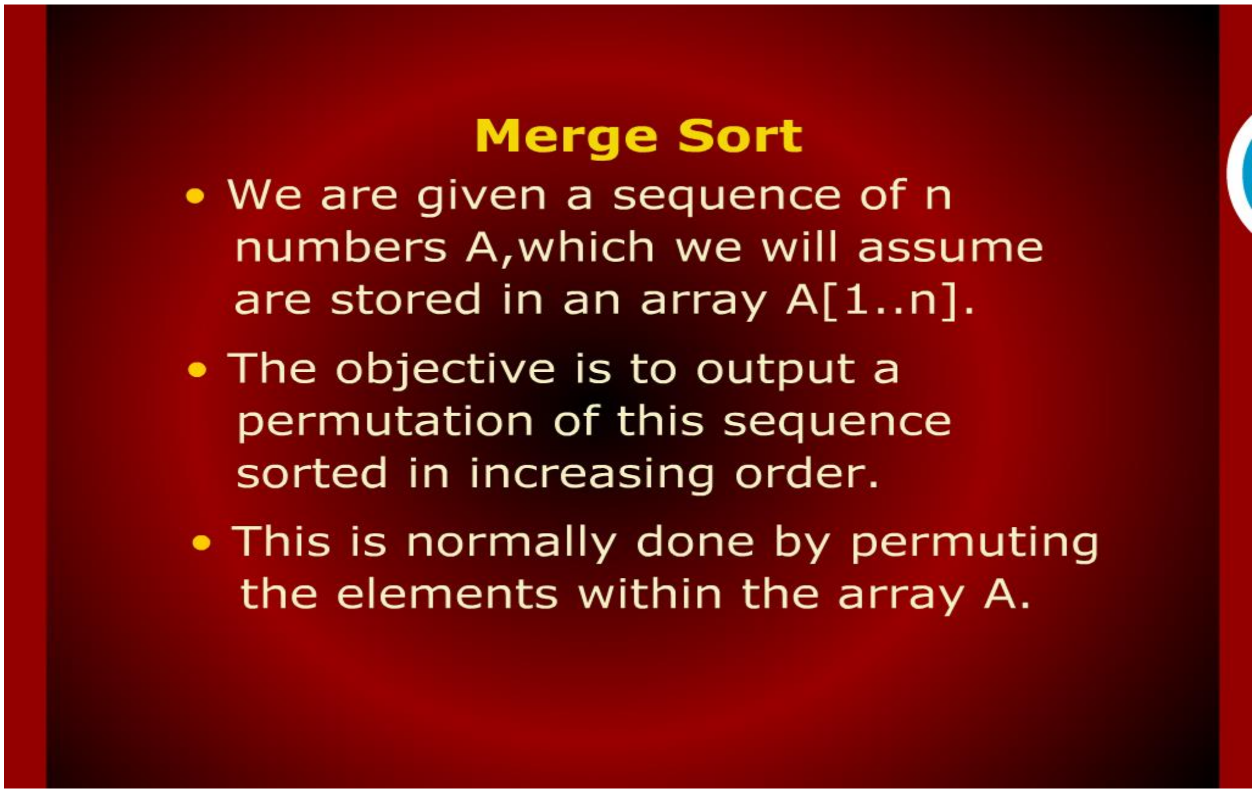
Which uses a recursive technique known as Divide-and-Conquer.





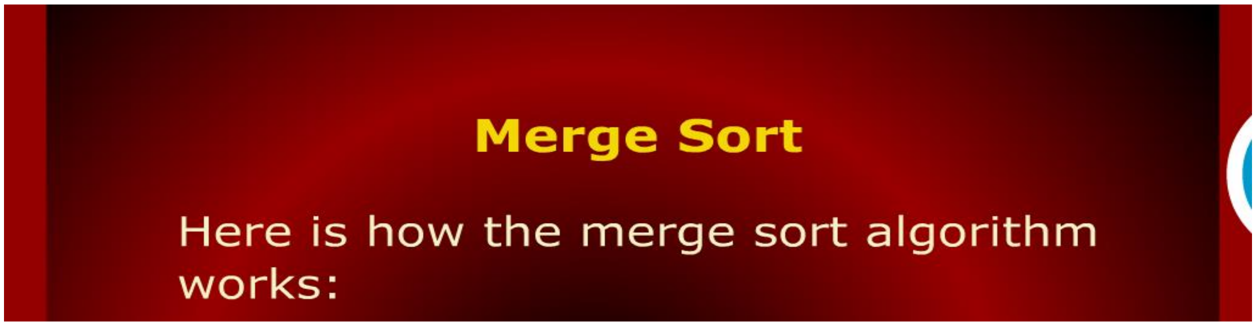
Merge Sort

- Divide and conquer strategy is applicable in a huge number of computational problems.
- The first example of divide and conquer algorithm we will discuss is a simple and efficient sorting procedure called *Merge Sort.*



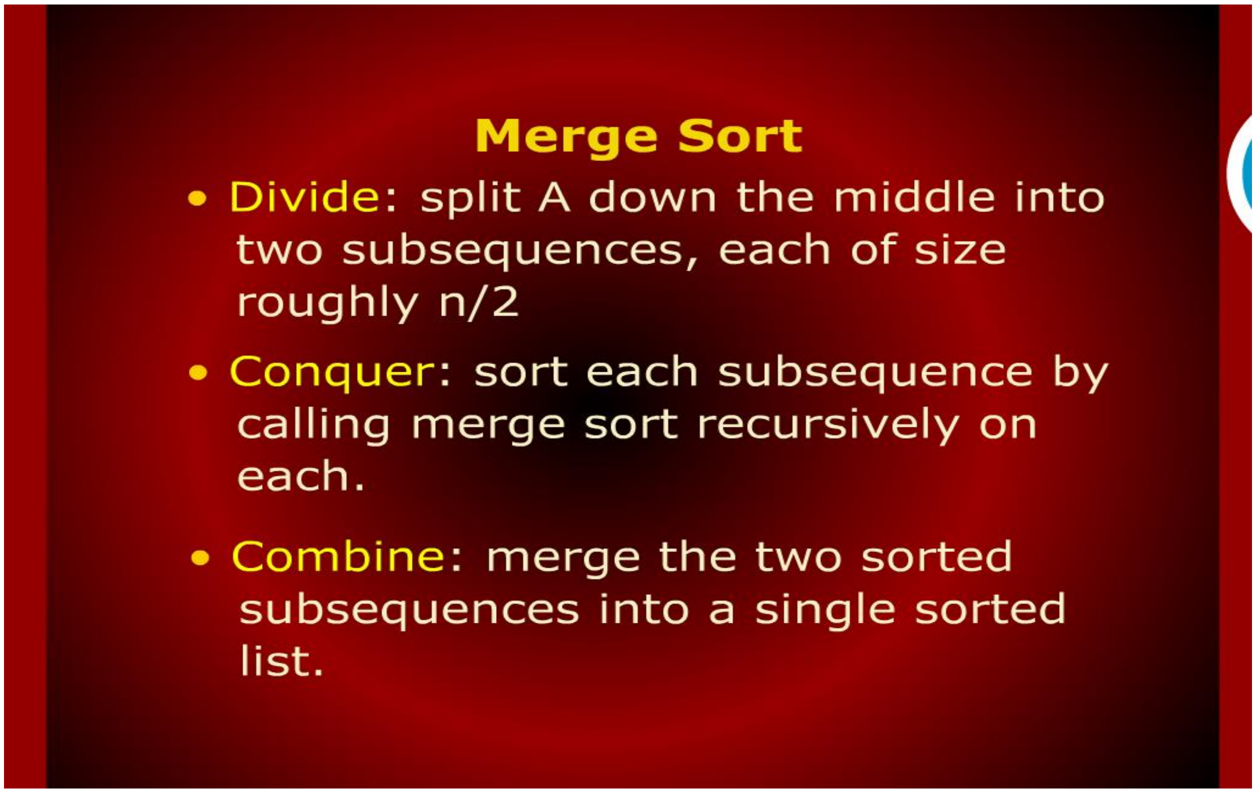
Merge Sort

- We are given a sequence of n numbers A,which we will assume are stored in an array A[1..n].
- The objective is to output a permutation of this sequence sorted in increasing order.
- This is normally done by permuting the elements within the array A.



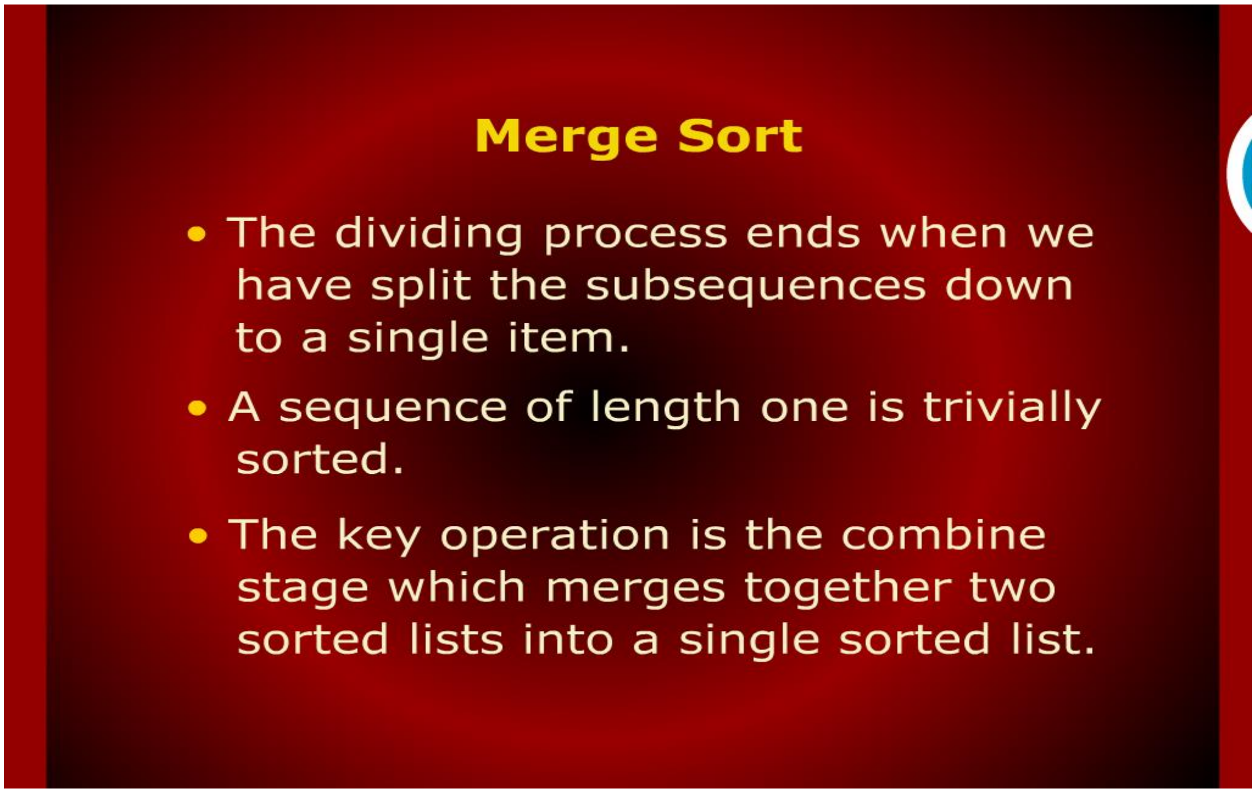
Merge Sort

Here is how the merge sort algorithm works:



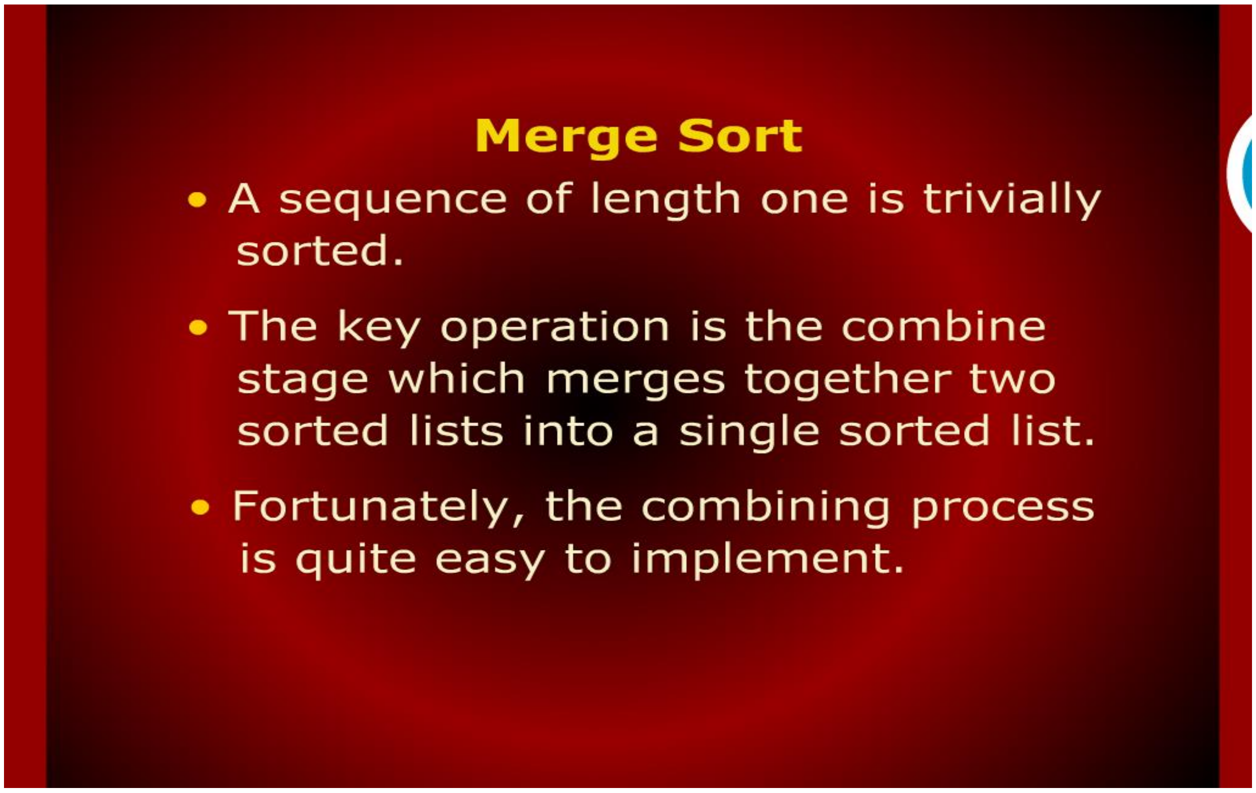
Merge Sort

- Divide: split A down the middle into two subsequences, each of size roughly n/2
- Conquer: sort each subsequence by calling merge sort recursively on each.
- Combine: merge the two sorted subsequences into a single sorted list.



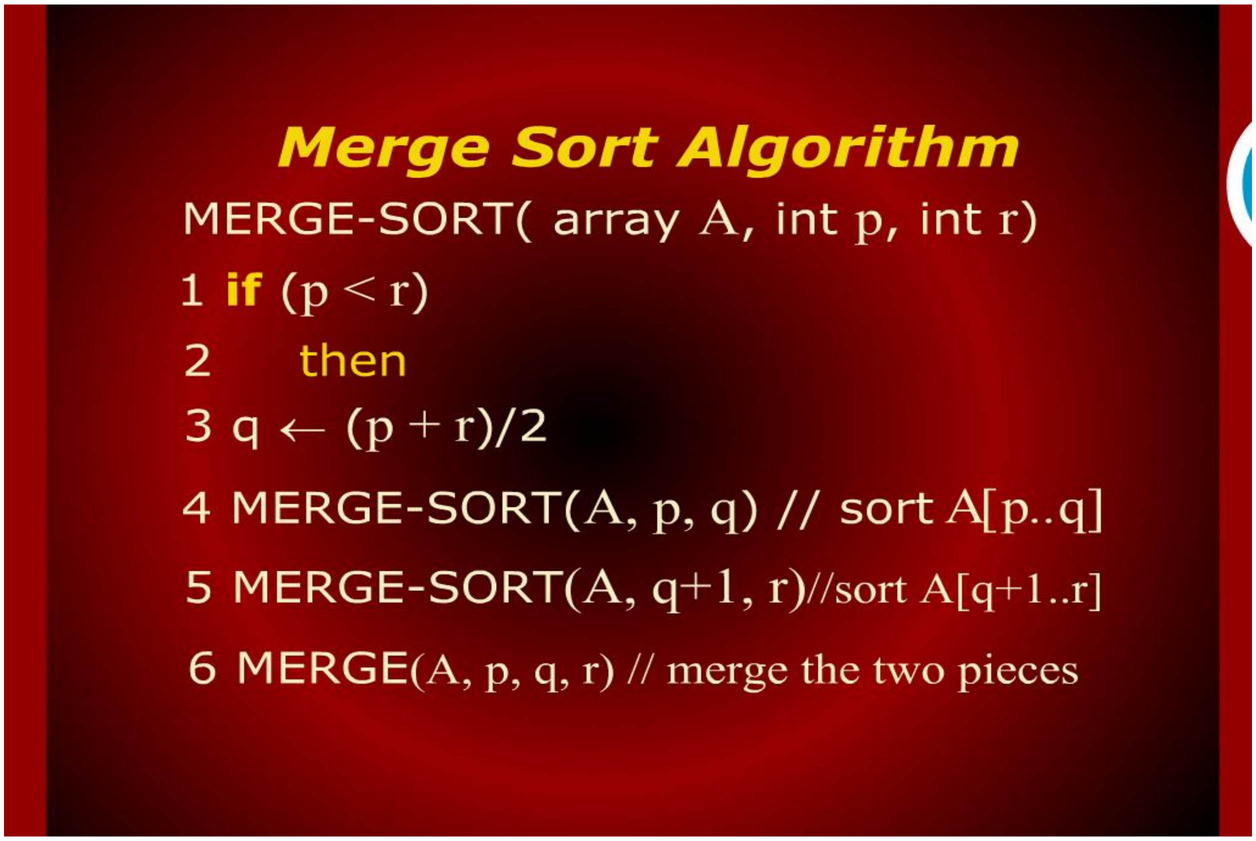
Merge Sort

- The dividing process ends when we have split the subsequences down to a single item.
- A sequence of length one is trivially sorted.
- The key operation is the combine stage which merges together two sorted lists into a single sorted list.



Merge Sort

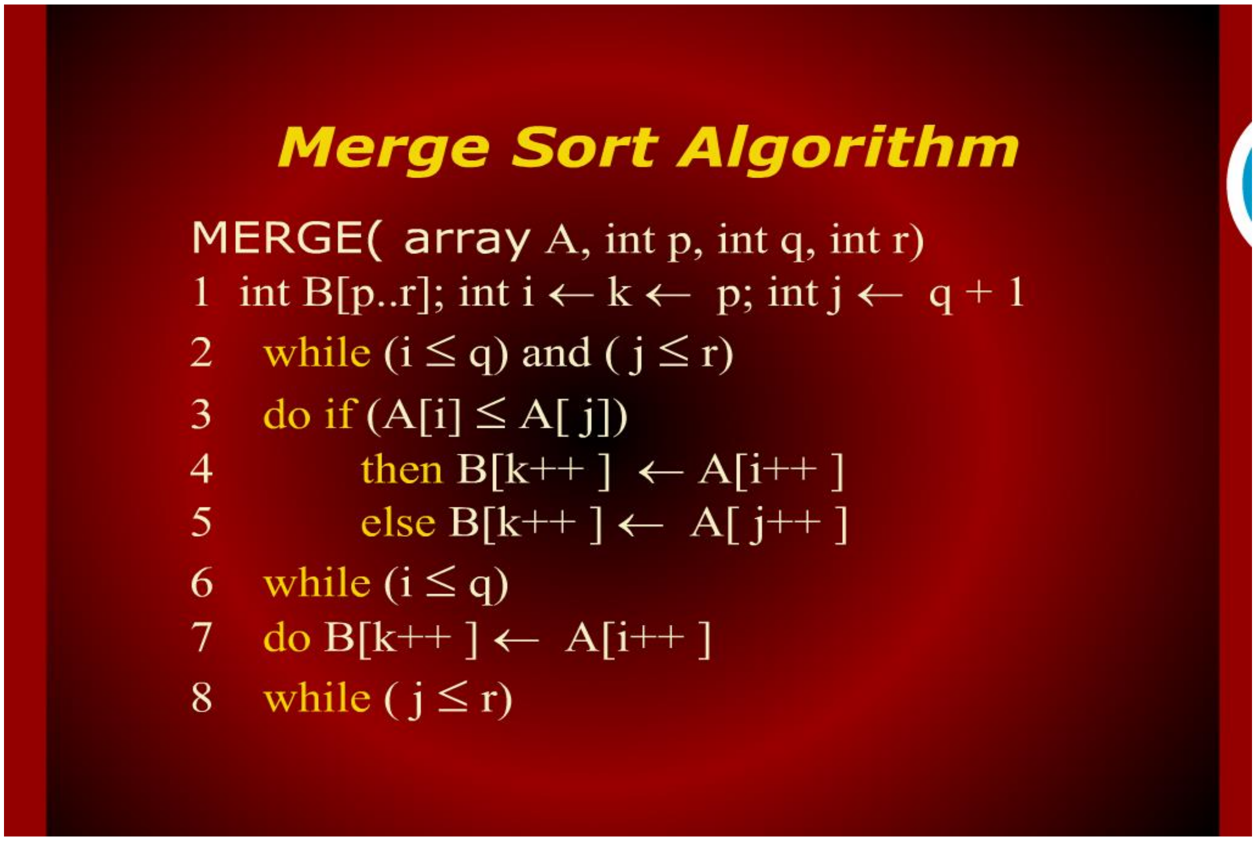
- A sequence of length one is trivially sorted.
- The key operation is the combine stage which merges together two sorted lists into a single sorted list.
- Fortunately, the combining process is quite easy to implement.



Merge Sort Algorithm

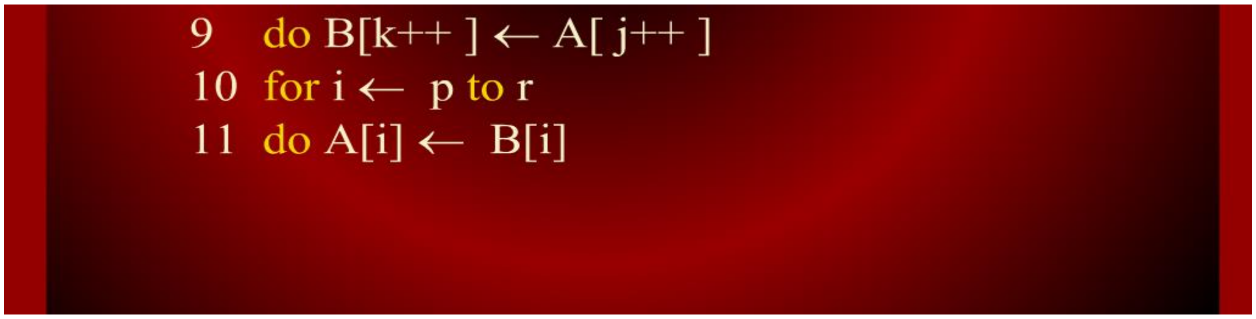
```
MERGE-SORT( array A, int p, int r)
1 if (p < r)
2     then
3 q ← (p + r)/2
4 MERGE-SORT(A, p, q) // sort A[p..q]
5 MERGE-SORT(A, q+1, r)//sort A[q+1..r]
6 MERGE(A, p, q, r) // merge the two pieces
```

Where p is beginning index of array list A and r is ending index of array list A



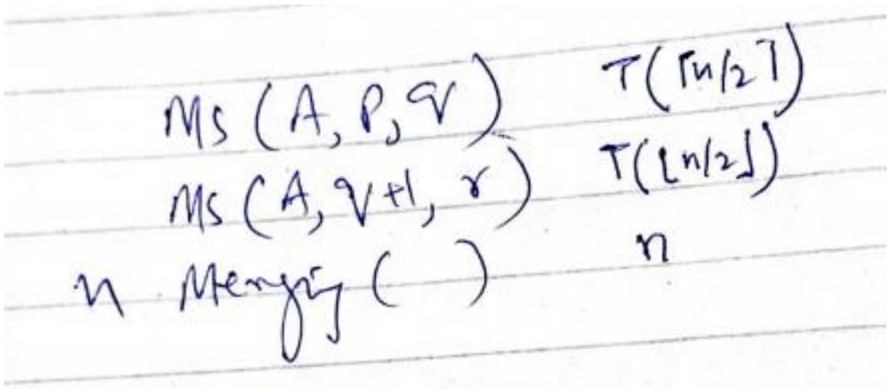
Merge Sort Algorithm

```
MERGE( array A, int p, int q, int r)
1  int B[p..r]; int i ← k ← p; int j ← q + 1
2    while (i ≤ q) and ( j ≤ r)
3    do if (A[i] ≤ A[ j])
4            then B[k++ ] ← A[i++ ]
5            else B[k++ ] ← A[ j++ ]
6    while (i ≤ q)
7    do B[k++ ] ← A[i++ ]
8    while ( j ≤ r)
```



```
9    do B[k++ ] ← A[ j++ ]
10   for i ← p to r
11   do A[i] ← B[i]
```

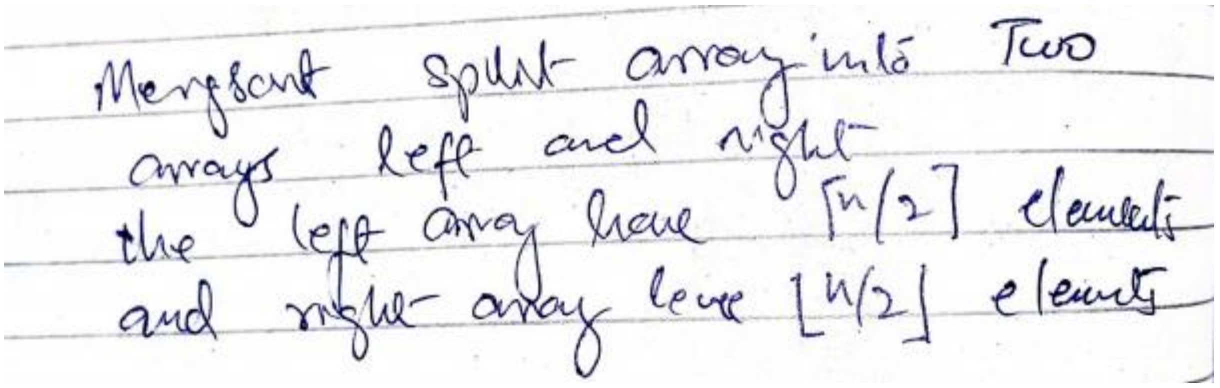
Time analysis



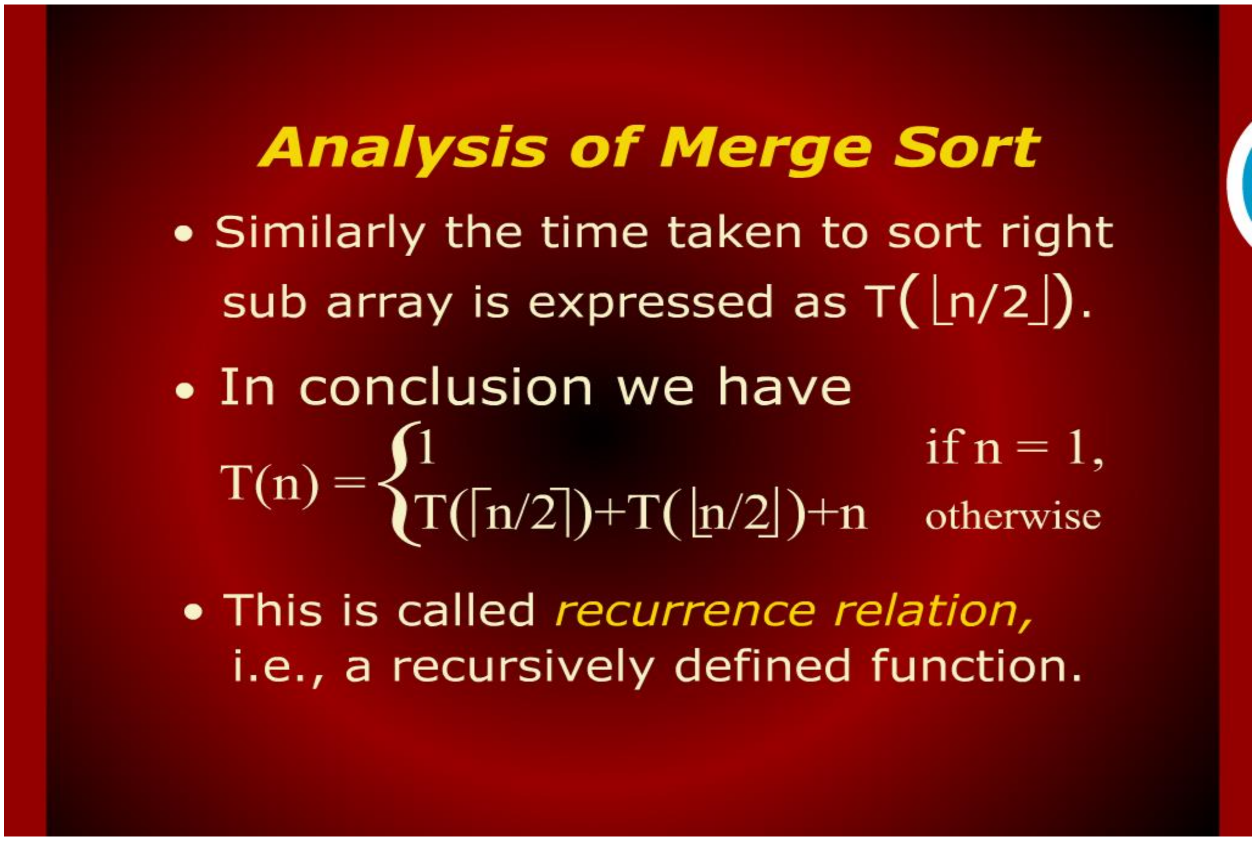
MS (A, P, q) $T(\lceil n/2 \rceil)$

MS (A, q+1, r) $T(\lfloor n/2 \rfloor)$

n Merging ( )   n



Mergsort split array into Two arrays left and right the left array have $\lceil n/2 \rceil$ elements and right array have $\lfloor n/2 \rfloor$ elements

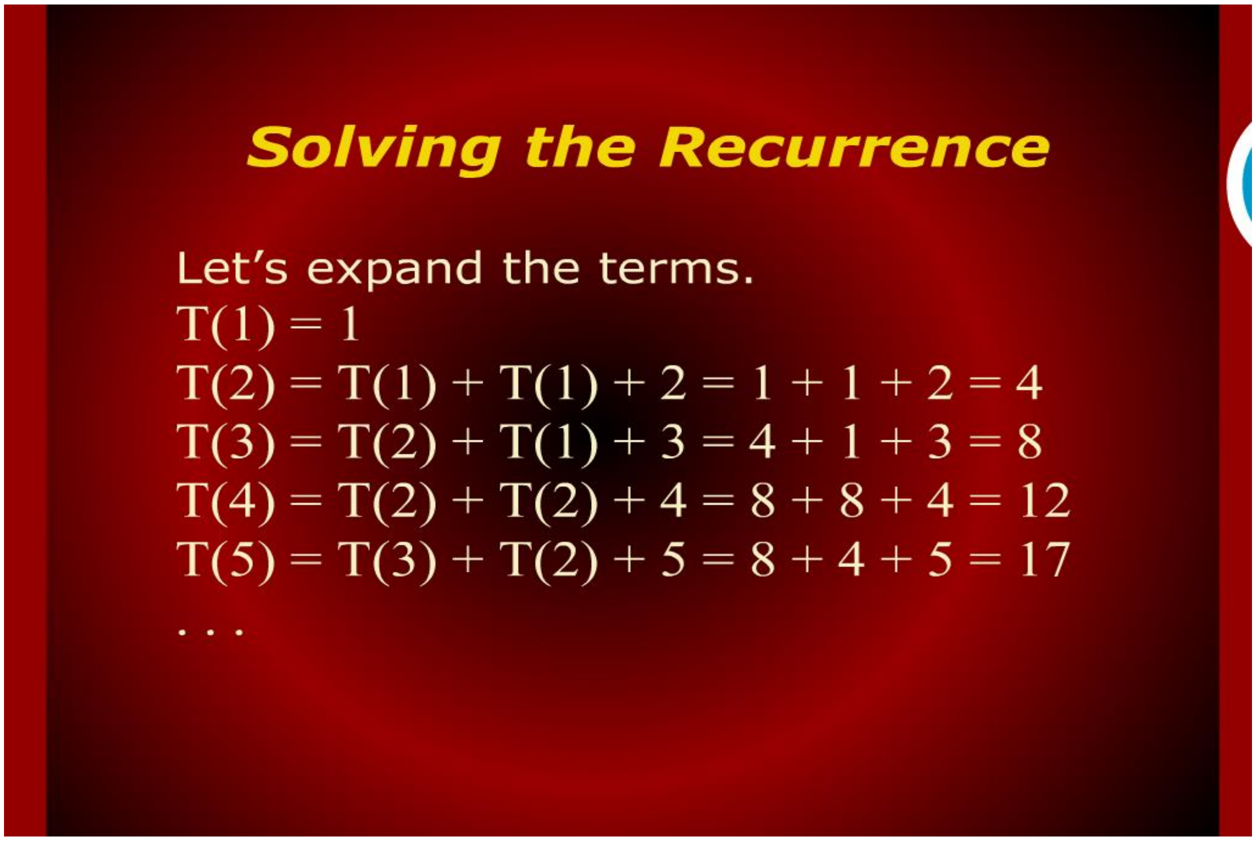


Analysis of Merge Sort

- Similarly the time taken to sort right sub array is expressed as $T(\lfloor n/2 \rfloor)$.
- In conclusion we have

$$T(n) = \begin{cases} 1 & \text{if } n = 1, \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n & \text{otherwise} \end{cases}$$

- This is called *recurrence relation,* i.e., a recursively defined function.



Solving the Recurrence

Let’s expand the terms.

$T(1) = 1$

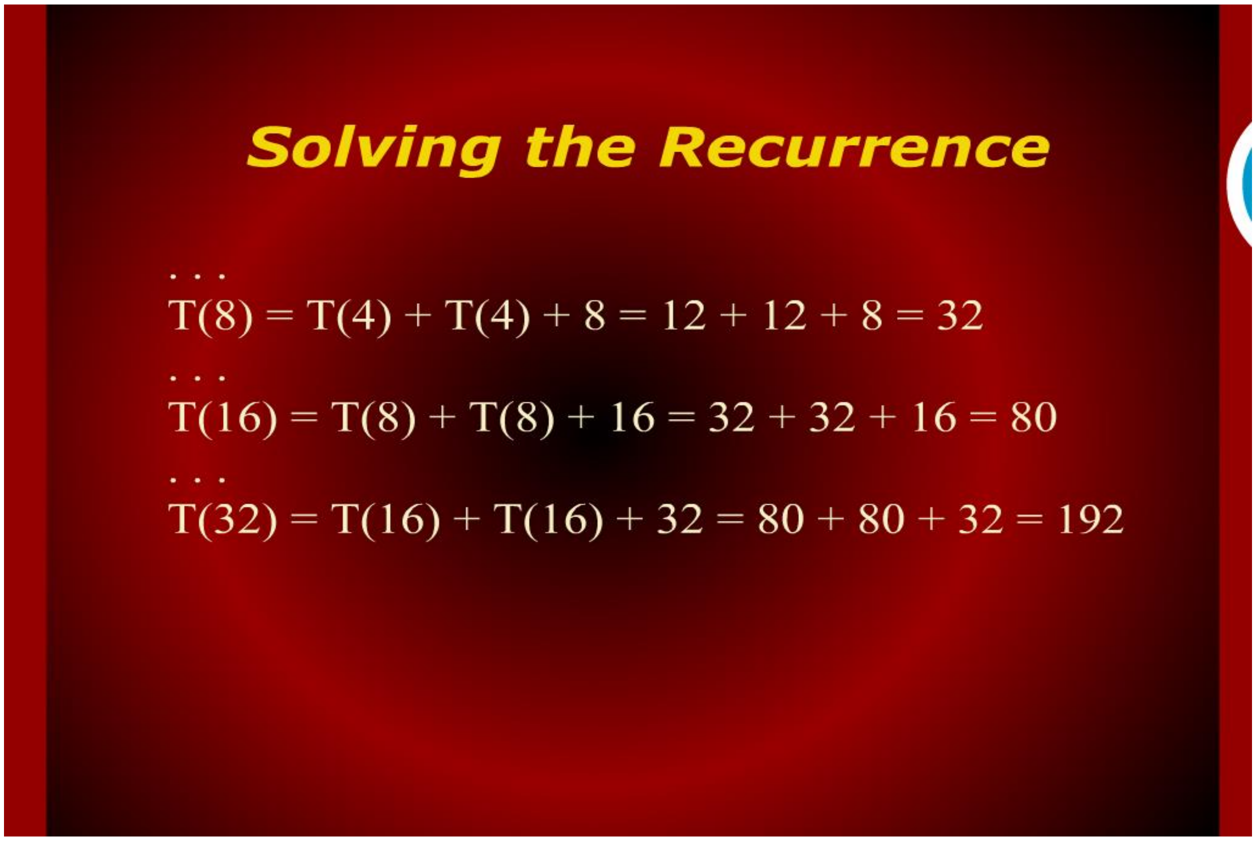
$T(2) = T(1) + T(1) + 2 = 1 + 1 + 2 = 4$

$T(3) = T(2) + T(1) + 3 = 4 + 1 + 3 = 8$

$T(4) = T(2) + T(2) + 4 = 8 + 8 + 4 = 12$

$T(5) = T(3) + T(2) + 5 = 8 + 4 + 5 = 17$

. . .



Solving the Recurrence

. . .

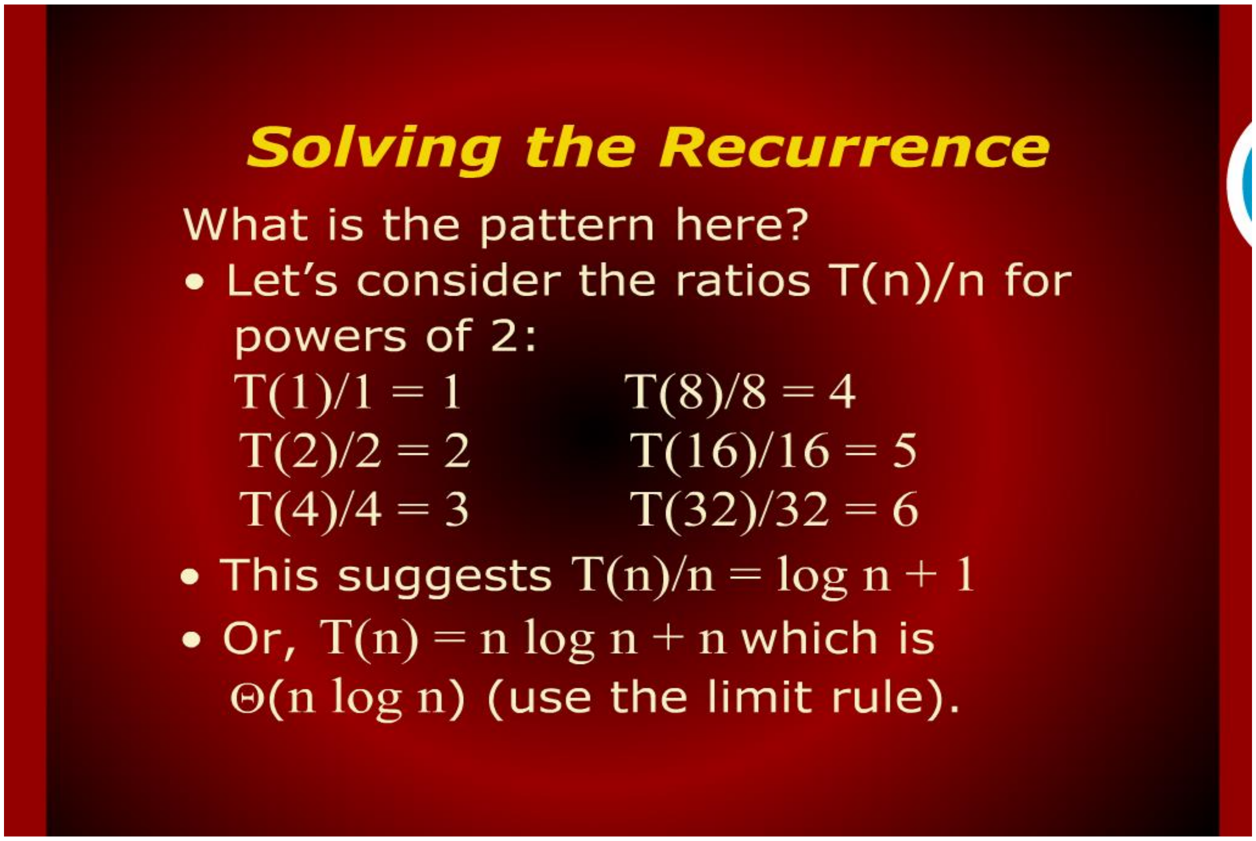
$T(8) = T(4) + T(4) + 8 = 12 + 12 + 8 = 32$

. . .

$T(16) = T(8) + T(8) + 16 = 32 + 32 + 16 = 80$

. . .

$T(32) = T(16) + T(16) + 32 = 80 + 80 + 32 = 192$



Solving the Recurrence

What is the pattern here?

- Let's consider the ratios T(n)/n for powers of 2:

$T(1)/1 = 1$	$T(8)/8 = 4$
$T(2)/2 = 2$	$T(16)/16 = 5$
$T(4)/4 = 3$	$T(32)/32 = 6$

- This suggests $T(n)/n = \log n + 1$
- Or, $T(n) = n \log n + n$ which is $\Theta(n \log n)$ (use the limit rule).