✓

Outline

- Introduction to Recurrence Equation
- Different methods to solve recurrence
- Divide and Conquer Technique
- Multiplying large Integers Problem
- Problem Solving using divide and conquer algorithm –
  - ✔ Binary Search
  - ✔ Sorting (Merge Sort, Quick Sort)
  - ✔ Matrix Multiplication

Recurrence Equation

Introduction

Many algorithms (divide and conquer) are recursive in nature.

When we analyze them, we get a recurrence relation for time complexity.

We get running time as a function of �� (input size) and we get the running time on inputs of smaller sizes.

A recurrence is a recursive description of a function, or a description of a function in terms of itself.

A recurrence relation recursively defines a sequence where the next term is a function of the previous terms.

Methods to Solve Recurrence

Substitution

Homogeneous (characteristic equation)

Inhomogeneous

Master method

Recurrence tree

Intelligent guess work

Change of variable

Range transformations

Substitution Method – Example 1

Time to solve the instance of size $n - 1$

Example

$$T(n) = T(n-1) + n$$

1:

1

- We make a guess for the solution and then we use mathematical induction to prove the g
  is correct or incorrect.

- Replacing n by $n-1$ and $n-2$, we can write following equations.

- Substituting equation 3 in 2 and equation 2 in 1 we have now, Time to solve the instance of size n

$$T(n-1) = T(n-2) + n - 1$$

$$T(n-2) = T(n-3) + n - 2$$

$$T(n) = T(n-3) + n - 2 + n - 1 + n$$

2 3

4

Substitution Method – Example 1

$$T(n) = T(n-3) + n - 2 + n - 1 + n$$

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

- From above, we can write the general form as,

- Suppose, if we take $k = n$ then,

$$T(n) = T(n-k) + (n-k+1) + (n-k+2) + \ \ldots + n$$

$$T(n) = T(n-n) + (n-n+1) + (n-n+2) + \ \ldots + n$$

$$T(n) = 0 + 1 + 2 + \ \ldots + n$$

Substitution Method – Example 2

$$\therefore t(n-1) = c2 + c2 + t(n-3)$$

$$t(n) = \begin{cases} c1 \; if \; n = 0 \\ c2 + t(n-1) \quad o/w \end{cases}$$

- Rewrite the equation, $t(n) = c2 + t(n-1)$
- Now, replace $\mathbf{n}$ by $\mathbf{n-1}$ and $\mathbf{n-2}$

$$t(n-1) = c2 + t(n-2)$$
$$t(n-2) = c2 + t(n-3)$$

- Substitute the values of $\mathbf{n-1}$ and $\mathbf{n-2}$

$$t(n) = c2 + c2 + c2 + t(n-3)$$

- In general,

$$t(n) = kc2 + t(n-k)$$

- Suppose if we take $k = n$ then,

$$t(n) = nc2 + t(n-n) = nc2 + t(0)$$
$$t(n) = nc2 + c1 = \boldsymbol{O(n)}$$

Substitution Method Exercises

Homogeneous Recurrence

Homogeneous Recurrence – Example 1 : Fibonacci Series

```
Function fibiter(n)
  i ← 1; j ← 0;
```

```
for k ← 1 to n do
    j ← i + j;
    i ← j - i;
    return j
```

Case 1

Homogeneous Recurrence – Example 1 : Fibonacci Series

Case 2

Homogeneous Recurrence – Example 1 : Fibonacci Series

```
Function fibrec(n)
    if n < 2 then return n
    else return fibrec (n - 1) + fibrec (n - 2)
```

Homogeneous Recurrence – Example 1 : Fibonacci Series

Homogeneous Recurrence – Example 1 : Fibonacci Series

Example 2 : Tower of Hanoi

tower 1 tower 2 tower 3

tower 1 tower 2 tower 3

Example 2 : Tower of Hanoi

Inhomogeneous equation

Example 2 : Tower of Hanoi

Homogeneous Recurrence Exercises

Master Theorem

Number of

Time to divide
& recombine

Time required to

sub-problems

solve a sub-problem

Master Theorem – Example 1

Merge sort

Master Theorem – Example 2

Binary Search

Master Theorem – Example 3

Master Theorem Exercises

Recurrence Tree Method

Recurrence Tree Method

The recursion tree for this recurrence is

Recurrence T~~ree Method~~

The recursion tree for this recurrence is

Recurrence Tree Method

The recursion tree for this recurrence is

Recurrence Tree Method - Exercises

Divide & Conquer (D&C)

Technique

Introduction

Many useful algorithms are recursive in structure: to solve a given problem, they call themselves recursively one or more times.

These algorithms typically follow a divide-and-conquer approach:

The divide-and-conquer approach involves three steps at each level of the recursion: 1. **Divide:** Break the problem into several sub problems that are similar to the original problem but smaller in size.

2. **Conquer:** Solve the sub problems recursively. If the sub problems sizes are small enough, just solve the sub problems in a straightforward manner.

3. **Combine:** Combine these solutions to create a solution to the original problem.

D&C Running Time Analysis

Binary Search

Introduction

Binary Search Example

1 3 7 9 11 32 52 74 90

Step 1:

1 2 3 4 5 6 7 8 9

1 3 7 9 11 32 52 74 90 Find approximate midpoint

Binary Search Example

Step 2: 1 2 3 4 5 6 7 8 9 **1 3 7 9 11 32 52 74 90**

Step 3:

1 2 3 4 5 6 7 8 9

1 3 7 9 11 32 52 74 90

Search for the target in the area before midpoint.

Binary Search Example

Step

4: 1 2 3 4 5 6 7 8 9 1 3 7 9 11 32 52 74 90

Find approximate

Step 5:

~~midpoint~~

1 2 3 4 5 6 7 8 9 1 3 7 9 11 32 52 74

90

Binary Search Example

Step

6: 1 2 3 4 5 6 7 8 9 1 3 7 9 11 32 52 74 90

Step 7:

1 2 3 4 5 6 7 8 9

1 3 7 9 11 32 52 74 90

Binary Search – Iterative Algorithm

```
Algorithm: Function biniter(T[1,...,n], x)
   if x > T[n] then return n+1
```

```
i ← 1;
j ← n;
while i < j do
    k ← (i + j ) ÷ 2
    if x ≤ T [k] then j ← k
    else i ← k + 1
return i
```

3

i

6

7

k

11

32

33

53

Binary Search – Recursive Algorithm

```
Algorithm: Function binsearch(T[1,…,n], x)
if n = 0 or x > T[n] then return n + 1 else
        return binrec(T[1,…,n], x)
     Function binrec(T[i,…,j], x)
        if i = j then return i
        k ← (i + j) ÷ 2
        if x ≤ T[k] then
           return binrec(T[i,…,k],x)
        else return binrec(T[k + 1,…,j], x)
```

Binary Search - Analysis

j

Binary Search – Examples

Multiplying Large Integers

Multiplying Large Integers – Introduction

Additional terms

Multiplying Large Integers – Example 1

Now we can compute the required product as follows:

Multiplying Large Integers – Analysis

Multiplying Large Integers – Example 2

Step 1:

Step 2:

Merge Sort

Introduction

- Merge Sort is an example of **divide and conquer algorithm.**
- It is based on the **idea of breaking down a list into several sub-lists** until each sub list consists of a **single element.**
- **Merging those sub lists** in a manner that results into a sorted list.
- **Procedure**

  Divide the unsorted list into N sub lists, each containing 1 element

  Take adjacent pairs of two singleton lists and merge them to form a list of 2 elements. N will now convert into N/2 lists of size 2

Repeat the process till a single sorted list of all the elements is obtained

Merge Sort – Example

Unsorted

~~Array~~

724 521 2 98 529 31 189 451

1 2 3 4 5 6 7 8

Step 1: Split the selected array

1 2 3 4 5 6 7 8 **724 521 2 98 529 31 189 451**

529 31 189 451 **1 2 3 4**

724 521 2 98 **1 2 3 4**

Merge Sort – Example

Select the left subarray and Split

1 2 3 4

[redacted]

Split

1 2 3 4

[redacted]

Select the right subarray and

1 2 529

31

1 2 724

521

1

Split

1

1

1

1

1 2 2 98

1

1 2 189

1

724

451 521 451

2

98

529

1

31

724 189

724

31 529 189 451 31 189

521 724 2 98 2 98 521

451 529

Merge

2 31 98 189 451 521 529 724

Merge Sort – Algorithm

```
Procedure: mergesort(T[1,…,n])
```

```
if n is sufficiently small then
insert(T)
else
            array
  U[1,…,1+n/2],V[1,…,1+n/2]
   U[1,…,n/2] ← T[1,…,n/2]
   V[1,…,n/2] ← T[n/2+1,…,n]
     mergesort(U[1,…,n/2])
     mergesort(V[1,…,n/2])
     merge(U, V, T)
```

```
merge(U[1,…,m+1],V[1,…,n+1],T[1,…,m+n]) i ← 1;
j ← 1;
U[m+1], V[n+1] ← ∞;
for k ← 1 to m + n do
   if U[i] < V[j]
   then T[k] ← U[i];  i ←
          i + 1;
   else T[k] ← V[j];
        j ← j + 1;
```

Procedure:

Merge Sort - Analysis

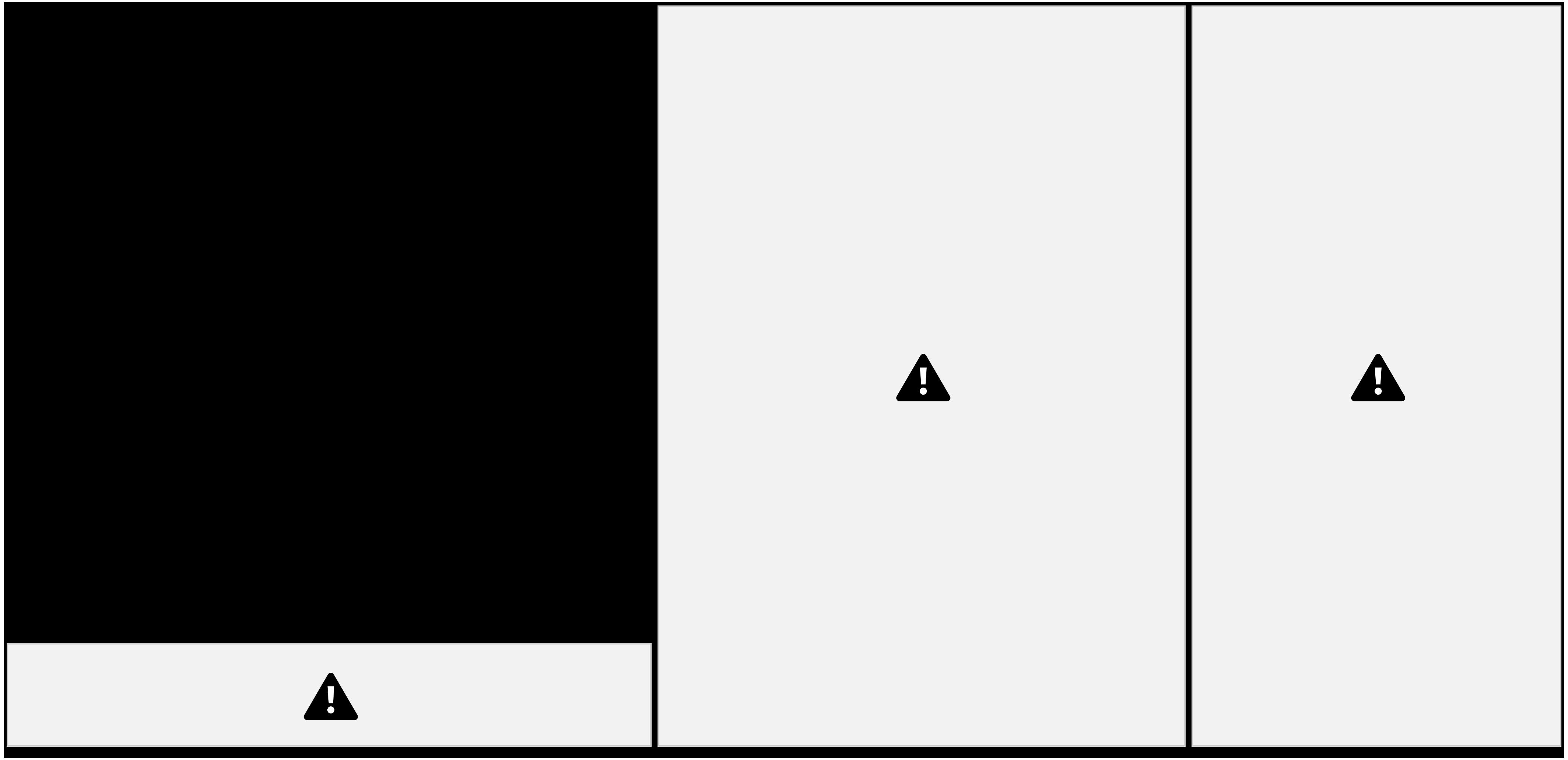
Strassen’s Algorithm for Matrix Multiplication

Matrix Multiplication

Matrix Multiplication

Strassen’s Algorithm for Matrix Multiplication

Strassen’s Algorithm for Matrix Multiplication [redacted]



Strassen’s Algorithm - Analysis

Quick Sort

Introduction

Quick sort chooses the first element as a pivot element, a **lower bound is the first index** and an **upper bound is the last index**.

The array is then **partitioned** on either side of the pivot.

Elements are moved so that, those **greater** than the pivot are shifted to its **right** whereas the others are shifted to its **left.**

Each Partition is **internally sorted recursively**.

Pivot
Elemen
t

B

0 1 2 3 4 5 6 7 42 23 74 8 9 99 87

11 65 58 94 36

U
B

L

Quick Sort - Example

8 9 0 1 2 3 4 5 6 7

```
Procedure pivot(T[i,…,j]; var l)

p ← T[i]
k ← i; l ← j+1
Repeat
k ← k+1 until T[k] > p
or k ≥ j
```

42 23 74 11 65 58 94 36 k

Swa
p

[redacted]

l

Swa

p

23 36 11 65 58 94 74 99 87 11 42

k l

LB = 0, UB = 9

p = 42

k = 0, l = 10

```
Repeat
l ← l-1 until T[l] ≤ p While k < l
do
 Swap T[k] and T[l]  Repeat k ←
k+1 until   T[k] > p
 Repeat l ← l-1 until   T[l] ≤ p
Swap T[i] and T[l]
```

42 23 74 11 65 58 94 36 99 87 36 74

k l

Quick Sort - Example

```
Procedure pivot(T[i,…,j]; var l)
```

89 0 1 2 3 4 5 6 7

```
p ← T[i]
```

[redacted] 42 [redacted]

[redacted]

k ← i; l ← j

j+1

L

Repeat

B

B

23 36

k ← k+1 until [redacted]

T[k] > p or k ≥ U

Repeat

l ← l-1 until T[l] ≤ p

kl

While k < l do T[l]

B

UB

Swap T[k] and L

Repeat k ← k+1 until T[k] > p

k l

Repeat l ← l-1 until T[l] ≤ p

11 23 36 42 65 58 94 74 99 87

Swap T[i] and T[l]

11 23 36 42 65 58 94 74 99 87

23 36

Quick Sort - Example

L

U

B

B

Procedure pivot(T[i,…,j]; var l) 89 0 1 2 3 4 5 6 7

p ← T[i]
k ← i; l ← j+1
Repeat
k ← k+1 until T[k] > p

or k ≥ j
11 23 36 42 [redacted]

p
[redacted]

Swa

T[k] > p

65 [redacted] 65

l

k

Repeat
l ← l-1 until T[l] ≤
p While k < l do
 Swap T[k] and T[l]
Repeat k ← k+1 until

T[l] ≤ p

58 65 94 74 99 87

B
UB

L

 Repeat l ← l-1 until

Swap T[i] and T[l]

Quick Sort - Example

Procedure

11 23 36 42 58 65 94 74 99 87

B

94

U

B

Swa

p [redacted]

pivot(T[i,…,j]; var l)

```
p ← T[i]
k ← i; l ← j+1
Repeat
k ← k+1 until T[k] > p
```

or k ≥ j Repeat

l ← l-1 until T[l] ≤ p

[redacted] k

Swa

p [redacted] 94

94 [redacted] k l

l

L

Swap T[k] and T[l]

Swa

L

B

U

B

```
While k < l do
  Repeat k ← k+1 until   T[k] > p
  Repeat l ← l-1 until   T[l] ≤ p
Swap T[i] and T[l]
```

p 74 87

87 74 94 99

k l

11 23 36 42 58 65 74 87 94 99

Quick Sort - Algorithm

```
Procedure: quicksort(T[i,…,j])
{Sorts subarray T[i,…,j] into
ascending order}
if j – i is sufficiently small
then insert (T[i,…,j])
else
pivot(T[i,…,j],l)
quicksort(T[i,…,l - 1])
quicksort(T[l+1,…,j]
```

```
Procedure: pivot(T[i,…,j]; var
l) p ← T[i]
k ← i
l ← j + 1
repeat k ← k+1 until T[k] > p
or k ≥ j repeat l ← l-1 until
T[l] ≤ p while k < l do
Swap T[k] and T[l]
 Repeat k ← k+1 until T[k] > p
Repeat l ← l-1 until T[l] ≤ p
Swap T[i] and T[l]
```

Quick Sort Algorithm – Analysis

Quick Sort Algorithm – Analysis

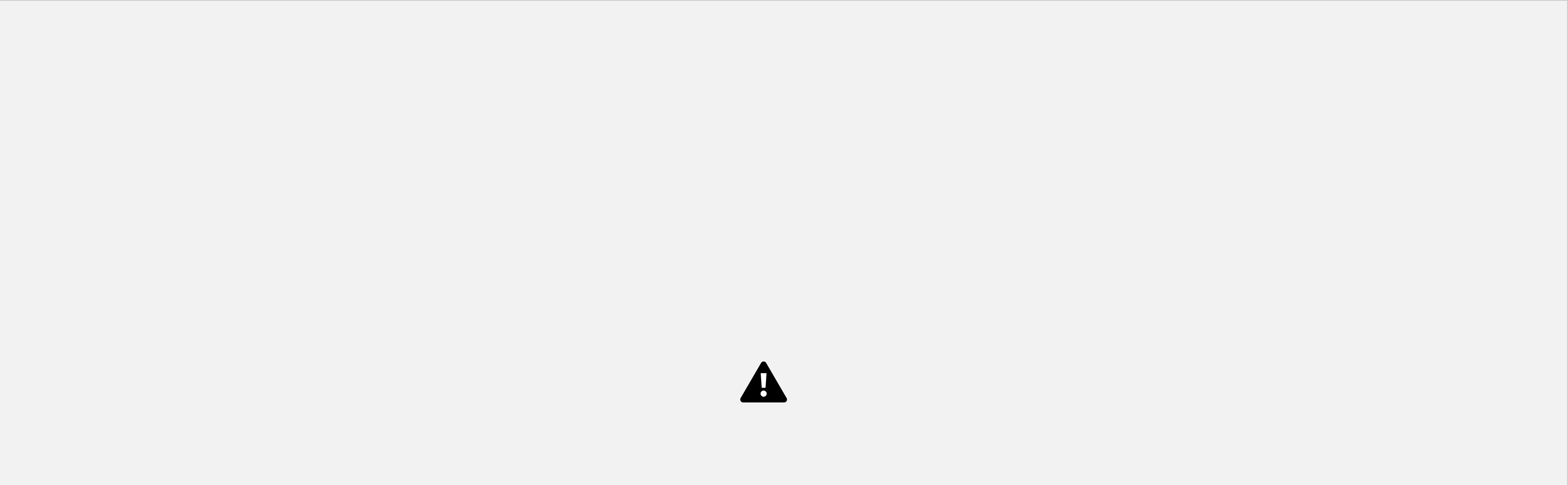
Quick Sort - Examples

Sort the following array in ascending order using quick sort algorithm.

1. 5, 3, 8, 9, 1, 7, 0, 2, 6, 4
2. 3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9
3. 9, 7, 5, 11, 12, 2, 14, 3, 10, 6

Exponentiation

Exponentiation – Sequential



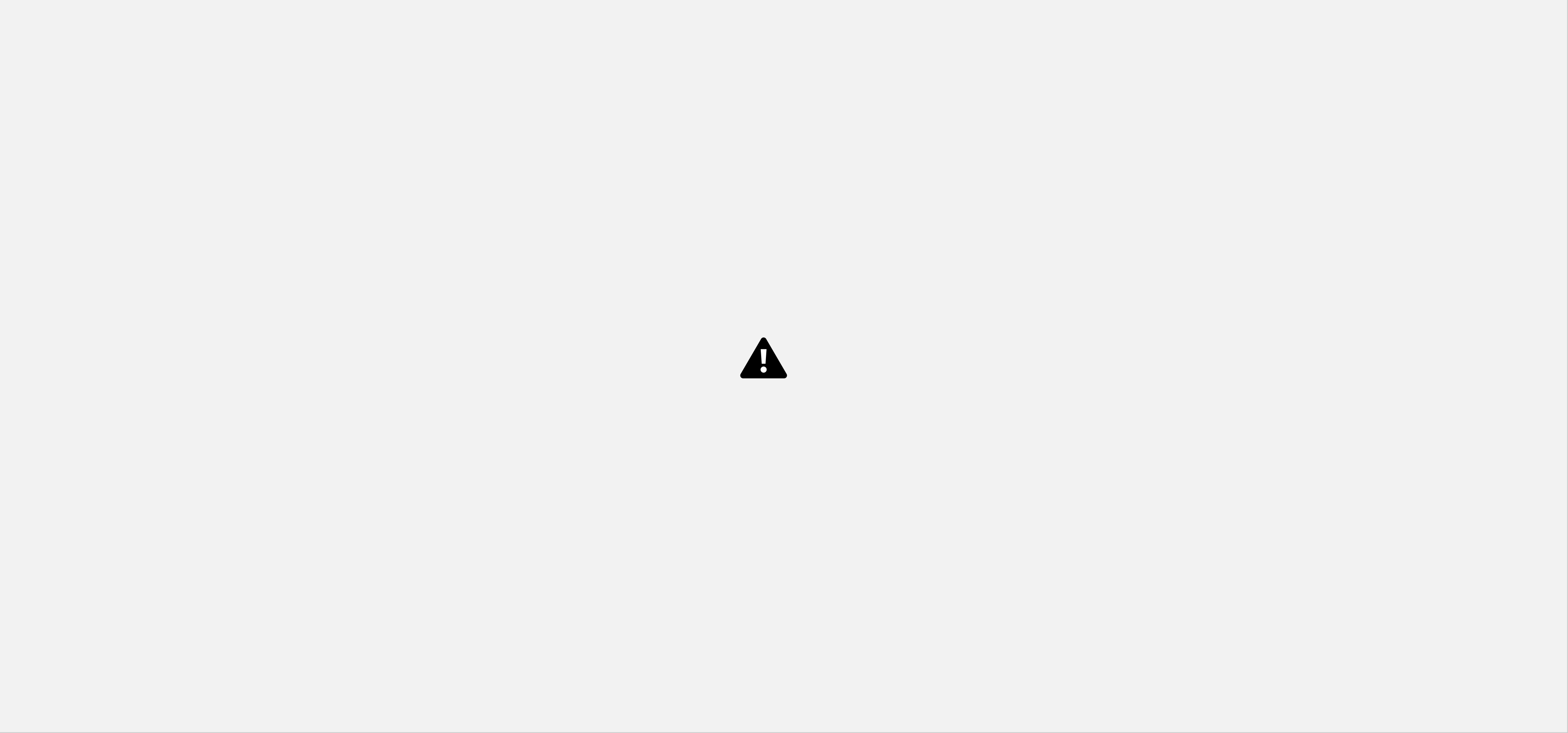
```
function exposeq(a, n)
  r ← a
  for i ← 1 to n - 1 do
    r ← a * r
  return r
```

Exponentiation – Sequential

10-10+1 10

Exponentiation - Sequential

Exponentiation – D & C



```
function expoDC(a, n)
   if n = 1 then return a
```

Exponentiation – D & C

Number of operations

the algorithm is given

Time taken by the algorithm is given by,

```
function expoDC(a, n)
    if n = 1 then return a
    if n is even then return [expoDC(a, n/2)]^2
    return a * expoDC(a, n - 1)
```

```
if n is even then return [expoDC(a, n/2)]^2
return a * expoDC(a, n - 1)
```

Exponentiation – Summary

Multiplication

Classic D&C

exposeq

expoDC

Thank You