Assessment

1. There are ______steps to solve the problem.

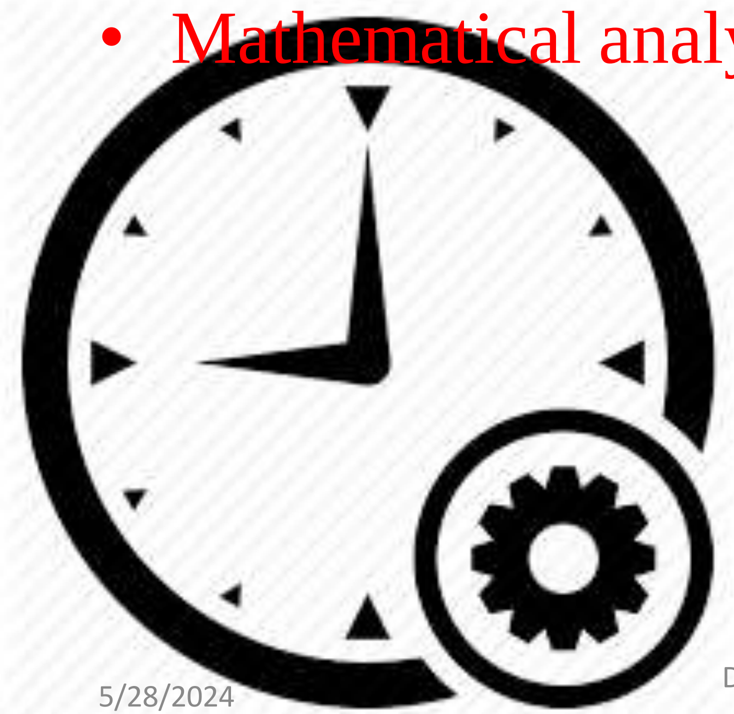
2. Two main measures for the efficiency of an algorithm are_________

3. Match the following

Arrange the Elements in order	-	Graph problems
N Queen Problem	-	String processing
Convex hull	-	Numerical Problems
Integral Calculus	-	Searching
Graph coloring	-	Combinatorial problem
Find a new string in existing one	-	Geometric problem
Find the given number	-	Sorting

Fundamentals of the Analysis of Algorithm Efficiency

- Analysis Framework
- Asymptotic Notations and its properties
- Mathematical analysis of Recursive algorithms
- Mathematical analysis of Non - Recursive algorithms



Analysis Framework

- Measuring Input size
- Units for measuring running time
- Orders of growth
- worst-case, best-case, average-case

Analysis Framework

Analyzing the efficiency of algorithm

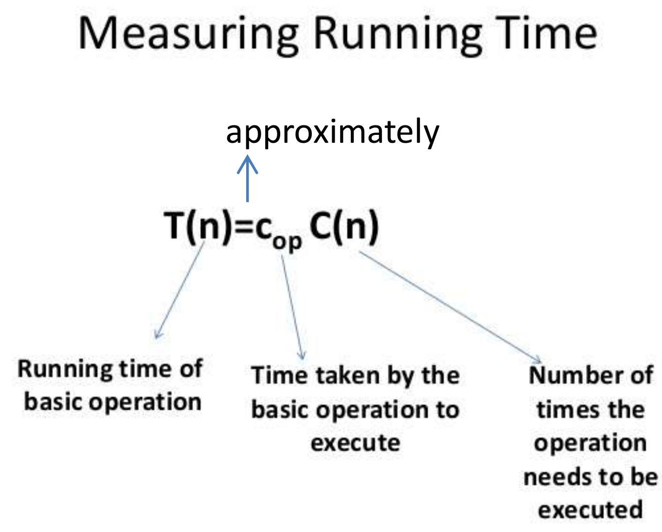
Time efficiency (fast) & Space efficiency (extra space)

- **Measuring an inputs size**
  - Algorithm efficiency (function input size n)- (Ex:searching)
  - N x N matrix multiplication → *n* (matrix order) , number of elements in matrix
  - Input size – algorithm's operation.
  - ***Example:*** spell-checking algorithm (characters, word)
  - Some application – size (no. of bits in the n's binary representation)

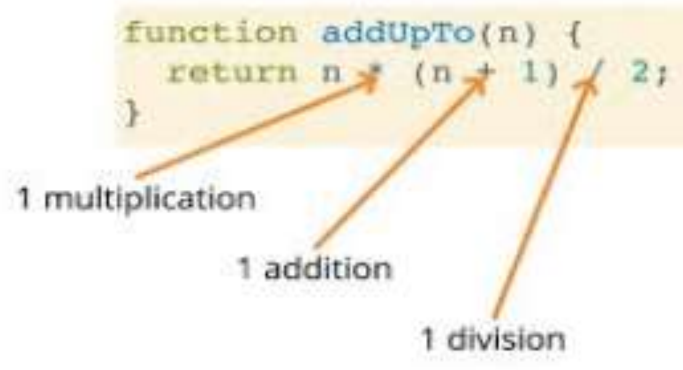
$$b = \log_2 n + 1$$

Analysis Framework

- **<u>Units for measuring running time</u>**
  - Units (seconds, milliseconds,…) → drawbacks →speed of computer, compiler– machine code, …
  - Units – count of basic operation executed
  - Ex: sorting – basic operation (key comparison) – *n* (input size)

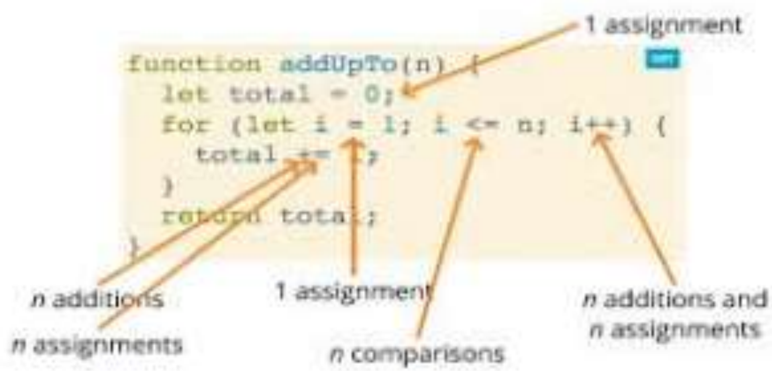


Analysis Framework - <u>Units for measuring running time</u>



3 simple operations, regardless of the size of *n*

This function will take 3 simple operations, regardless of the size of n. If we compare to the below function, we have a loop and it depends on the value of n.



Analysis Framework

- **<u>Units for measuring running time</u>**

<u>Example:</u>

```
for(i=0;i<n;i++)
{
    if(a[i]==k)
    {
      printf("\n Element found %d at position %d",a[i],i+1);
      exit(0);
    }
}
```

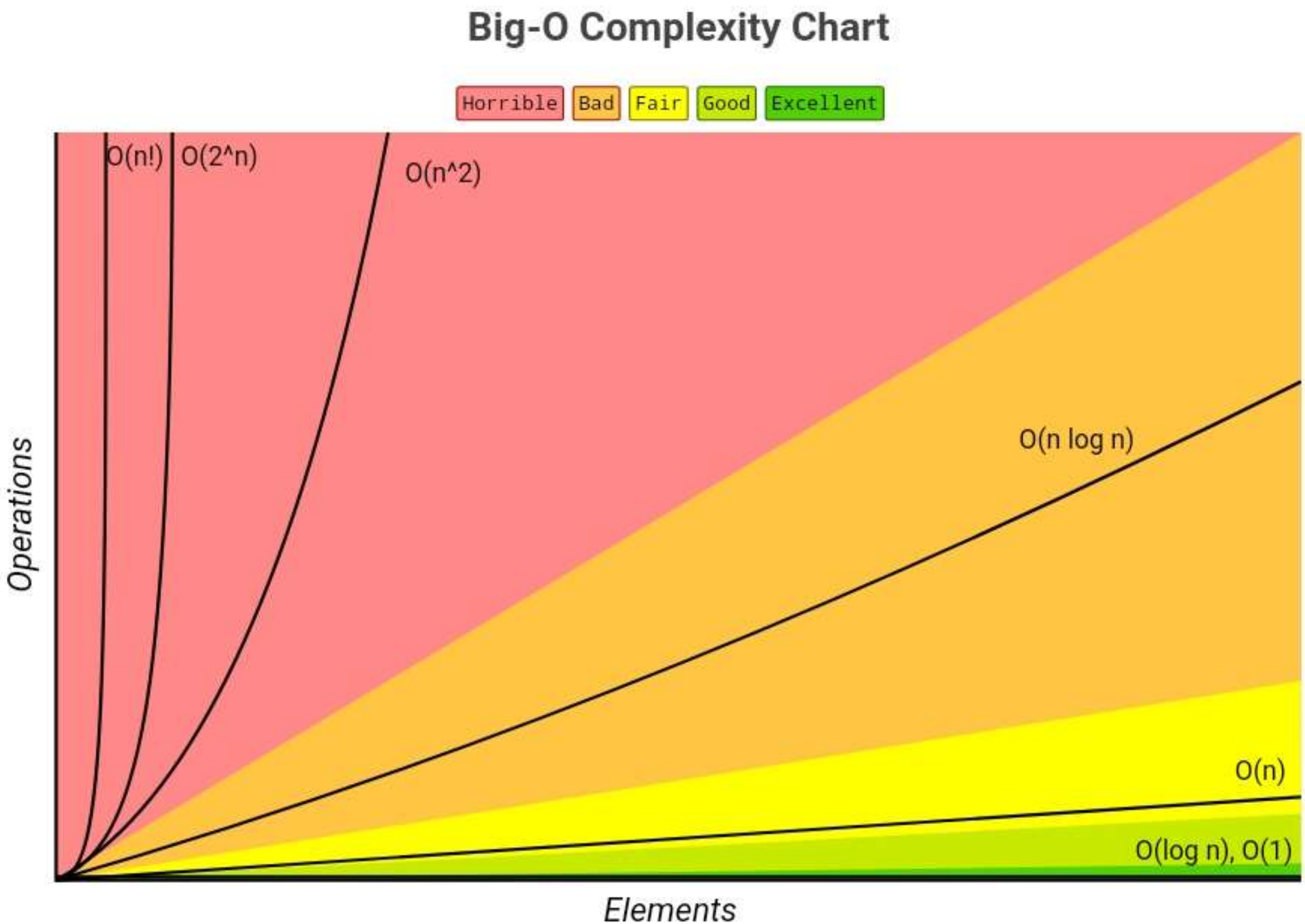
```
for (i = 0; i < n; ++i)
  {
    for (j = i + 1; j < n; ++j)
    {
      if (number[i] > number[j])
      {
        a = number[i];
        number[i] = number[j];
        number[j] = a;
      }
    }
```

Analysis Framework

- **Orders of growth**
  - Predicting the change in time and space of algorithm taken depending on the input size *n*
  - Measuring the performance of algorithm with respect to input size

TABLE 2.1 Values (some approximate) of several functions important for analysis of algorithms

n	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n	$n!$
10	3.3	10^1	$3.3 \cdot 10^1$	10^2	10^3	10^3	$3.6 \cdot 10^6$
10^2	6.6	10^2	$6.6 \cdot 10^2$	10^4	10^6	$1.3 \cdot 10^{30}$	$9.3 \cdot 10^{157}$
10^3	10	10^3	$1.0 \cdot 10^4$	10^6	10^9		
10^4	13	10^4	$1.3 \cdot 10^5$	10^8	10^{12}		
10^5	17	10^5	$1.7 \cdot 10^6$	10^{10}	10^{15}		
10^6	20	10^6	$2.0 \cdot 10^7$	10^{12}	10^{18}		


Big-O Complexity Chart
Horrible
Bad
Fair
Good
Excellent
O(n!)
O(2^n)
O(n^2)
O(n log n)
O(n)
O(log n), O(1)
Operations
Elements

Analysis Framework

- **<u>Worst-case, Best-case and Average-case efficiencies</u>**

```
ALGORITHM  SequentialSearch(A[0..n − 1], K)
    //Searches for a given value in a given array by sequential search
    //Input: An array A[0..n − 1] and a search key K
    //Output: The index of the first element in A that matches K
    //          or −1 if there are no matching elements
    i ← 0
    while i < n and A[i] ≠ K do
        i ← i + 1
    if i < n return i
    else return −1
```

Worst-case - $C_{worst}(n)=n$
Best-case - $C_{best}(n)=1$
Average-case - average no of steps

Analysis Framework

- **Worst-case, Best-case and Average-case efficiencies**

LInear Search Example

Amortized efficiency