① What is Time Complexity?

old computer

data :- 1,00,000 element in an array

Algorithm, linear search for target that does not exist in an array

Time Taken :- 10 sec

M1 macbook (very fast)

— // —

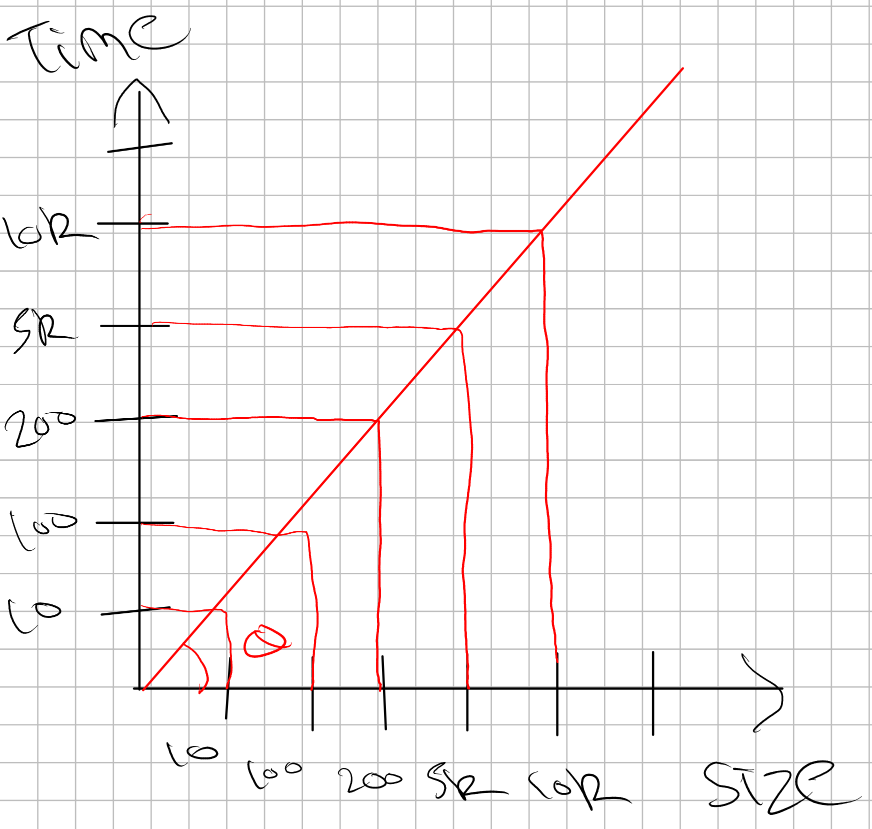
— // —

Time Taken :- 1 sec

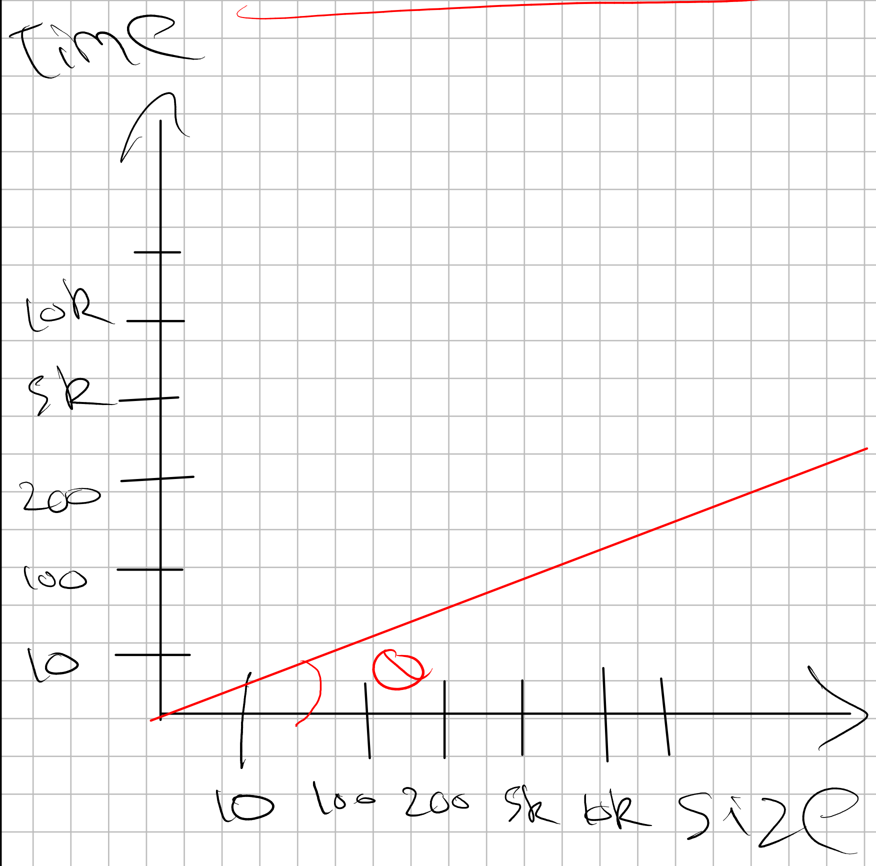
⊕ Both machines have same time complexity.

Time complexity != Time Taken

Old machine

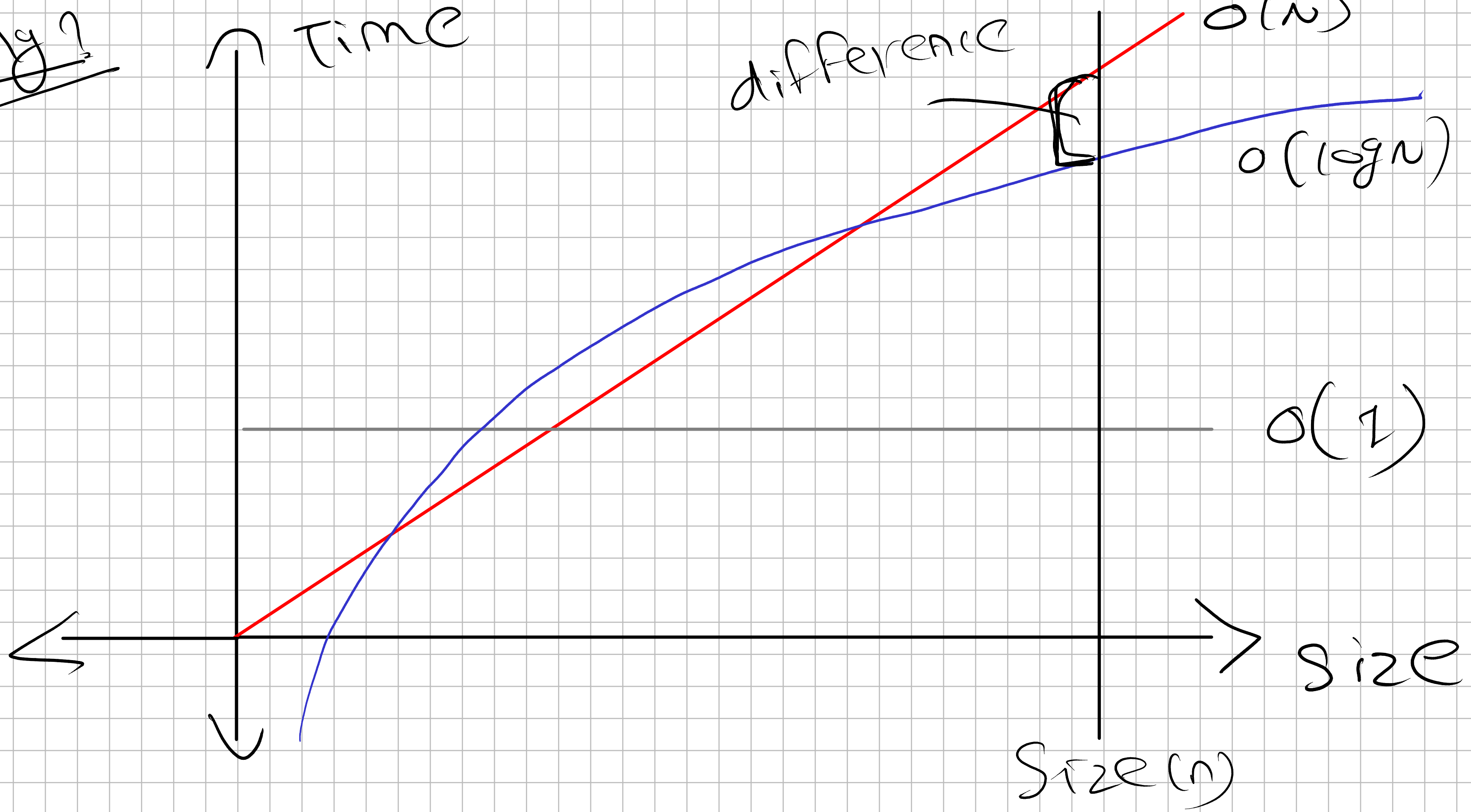


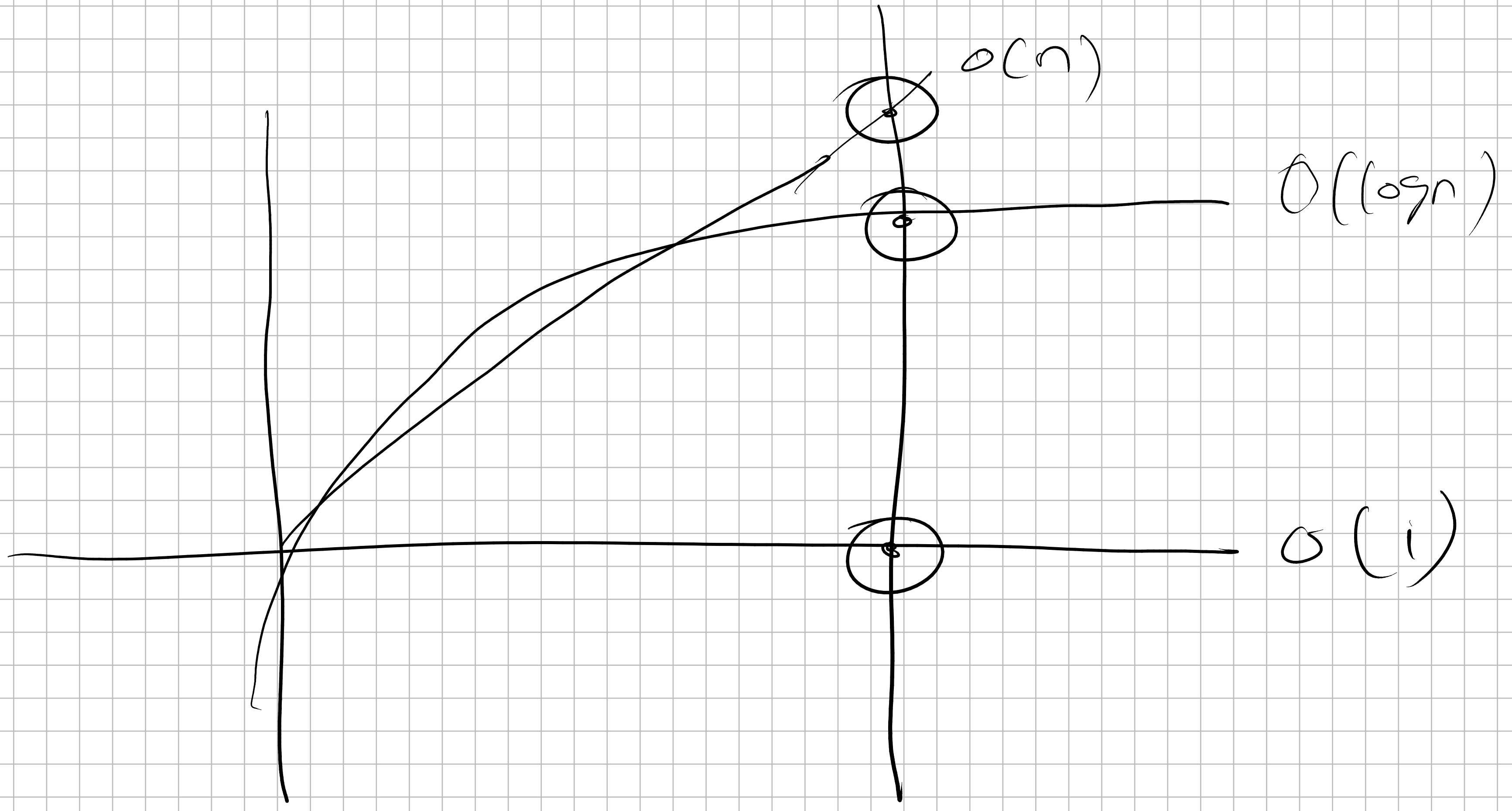
M1 Macbook



θ Function that gives us the relationship about how the time will grow as the input grows.

Why?





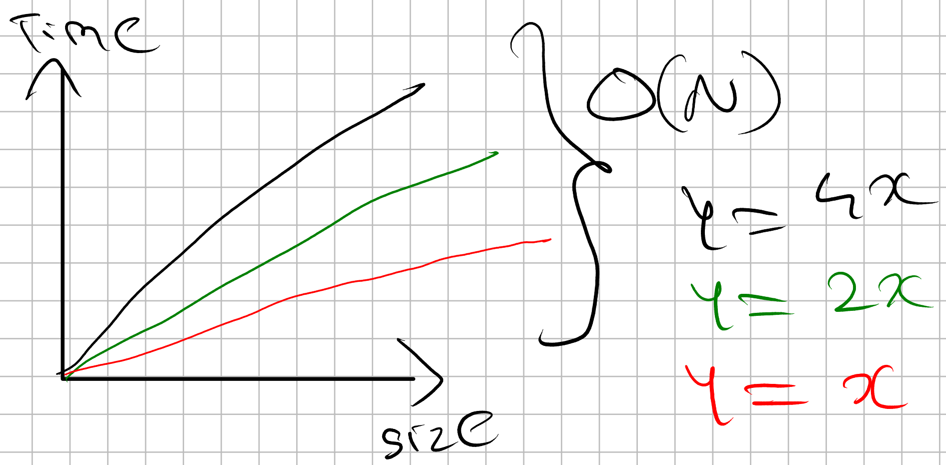
$O(1) < O(\log N) < O(N)$

what do we consider when thinking about complexity?

① Always look for worst case complexity.

② Always look at complexity for large | ∞ data.

③



- Even though value of actual time is different they are all growing linearly.
- We don't know the actual time.
- This is why, we ignore all constants.

④ $O(N^3 + \log N)$

- From Point no. ②

⇒ 1 mil ⇒ $((1\text{mil})^3 + \log(1\text{mil}))$

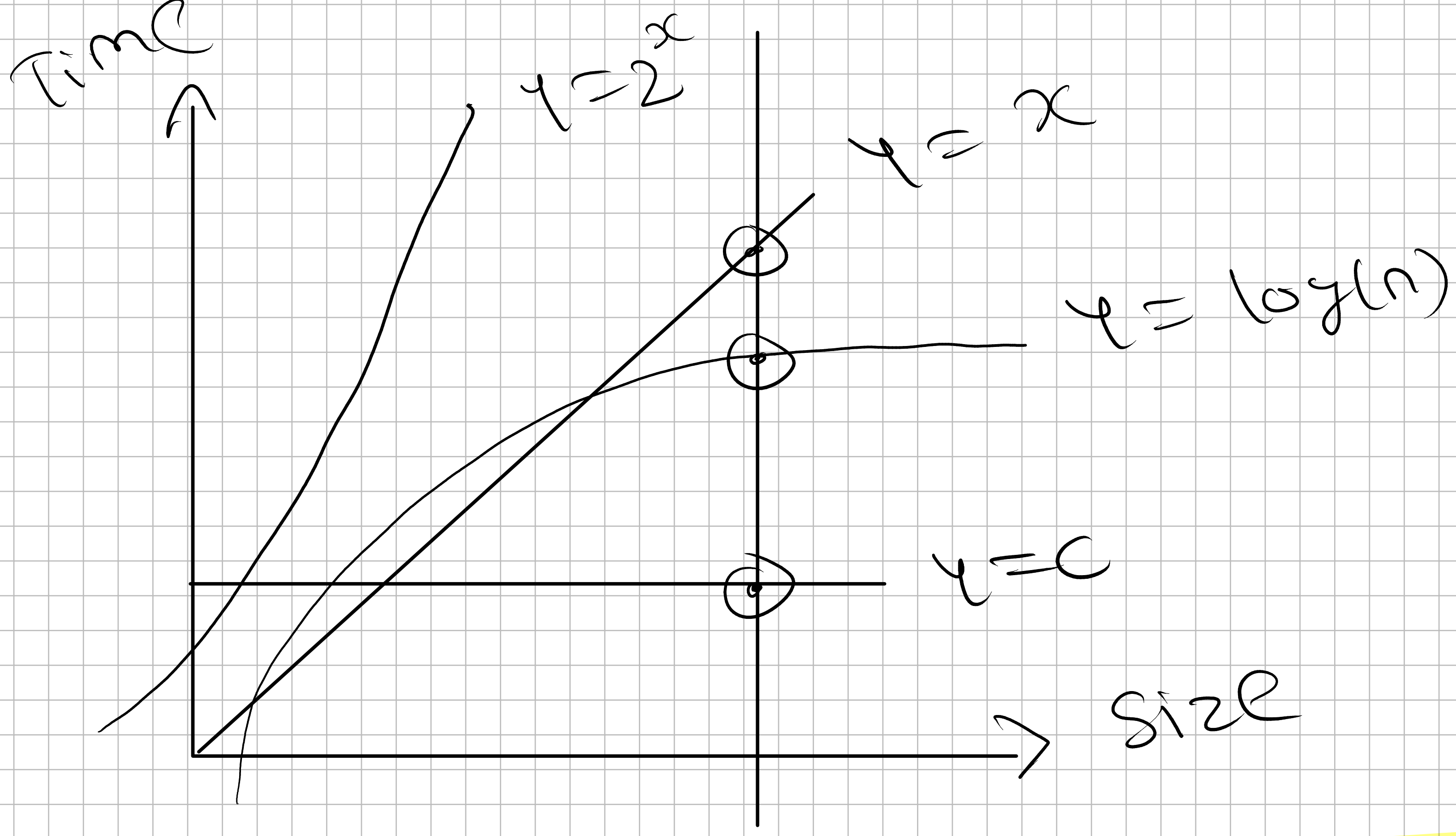
⇒ $(1\text{mil}^3 + \underbrace{6\text{ sec}})$ → very small hence ignore

Always ignore less dominating terms.

Ex:- $O(3N^3 + 4N^2 + 5N + 8)$

$= (N^3 + \cancel{N^2} + \cancel{N})$ → less dominating terms

$= O(N^3)$



$O(1) < O(\log(N)) < O(N) < O(2^n)$

$\downarrow$

$O(N \log N)$

Big-oh Notation

✱ word definition :-

① It represents the upper bound of running time of an algorith.

② It gives the worst-case complexity of an algorithm.

$O(N^3) \rightarrow$ upper bound

✱ maths :-

$$f(n) = O(g(n))$$

$$\boxed{\lim_{N\to\infty} \frac{f(n)}{g(n)} < \infty}$$

$$O(\underbrace{N^3}_{g(n)}) = O(\underbrace{6N^3 + 3N + 5}_{f(n)})$$

$$= \lim_{N\to\infty} \frac{6N^3+3N+5}{N^3} = 6 + \frac{3}{\infty} + \frac{5}{\infty}$$

$$= \lim_{N\to\infty} 6 + \frac{3}{N^2} + \frac{5}{N^3}$$

$$= 6 + 0 + 0$$

$$= \boxed{6 < \infty}$$

finite value

Big omega :- opposite of Big-oh

* It represents the lower bound of the running time of an algorithm. Thus it provides the best case time complexity of an algorithm.

$\Omega(n^3) \Rightarrow$ (Lower bound)

Maths :-

$$\boxed{\lim_{n \to \infty} \frac{f(n)}{g(n)} > 0}$$

Theta Notation:-

- Theta notation encloses the function from above & below.
- It represents the upper & lower bound.
- It is used for analyzing average-case complexity of an algorithm.

$\Theta(N^2) \Rightarrow$ Both upper bound & lower bound is $= N^2$

$$\boxed{0 < \lim_{N\to\infty} \frac{f(n)}{g(n)} < \infty}$$

Theta Notation:-

- Theta notation encloses the function from above & below.
- It represents the upper & lower bound.
- It is used for analyzing average-case complexity of an algorithm.

$\Theta(N^2) \Rightarrow$ Both upper bound & lower bound is $= N^2$

$$\boxed{0 < \lim_{N \to \infty} \frac{f(n)}{g(n)} < \infty}$$

little o notation

✱ This is also giving upper bound.

words :- loose up

Big oh	little o
	(stronger statement)
$f = O(g)$	$f = o(g)$
$f \le g$	$f < g$
	strictly slower

maths :-

$$\boxed{\lim_{n \to \infty} \frac{f(N)}{g(N)} = 0}$$

Ex :- $f = N^2$ $\qquad$ $g = N^3$

$$\lim_{N \to \infty} \frac{N^2}{N^3} \Rightarrow \lim_{N \to \infty} \frac{1}{N} = \underline{\underline{0}}$$

little omega :

Big Ω

$f = \Omega(g)$

$f \geq g$

little ω

$f = \omega(g)$

$f > g$

Maths :-

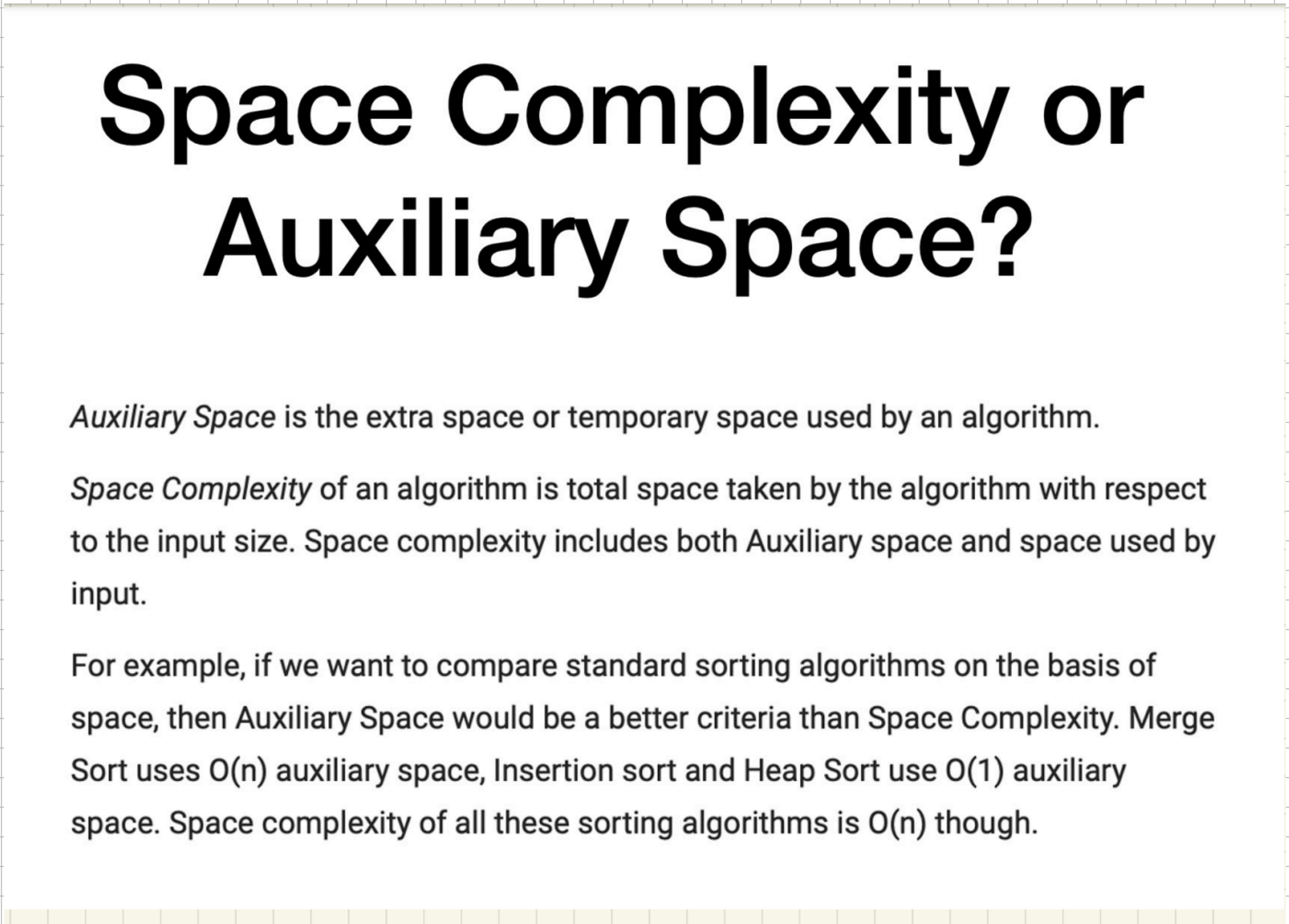
$$\boxed{\lim_{N \to \infty} \frac{f(N)}{g(N)} = \infty}$$

Ex :-

$$\lim_{N \to \infty} \frac{N^3}{N^2}$$

$$\Rightarrow \lim_{N \to \infty} N = \boxed{\infty}$$

Space complexity



Space Complexity or Auxiliary Space?

Auxiliary Space is the extra space or temporary space used by an algorithm.

Space Complexity of an algorithm is total space taken by the algorithm with respect to the input size. Space complexity includes both Auxiliary space and space used by input.

For example, if we want to compare standard sorting algorithms on the basis of space, then Auxiliary Space would be a better criteria than Space Complexity. Merge Sort uses O(n) auxiliary space, Insertion sort and Heap Sort use O(1) auxiliary space. Space complexity of all these sorting algorithms is O(n) though.

Q

```
for (i=1; i≤N;)
{
    for (j=1; j≤k; j++)
    {
        // some operation that takes time
              t
    }
    i = j+k
}
```

Inner loop: $O(kt)$ time

Ans: $O(kt + \underbrace{\text{times outer loop is running}}_{?})$

$j = 1, 1+k, 1+2k, 1+3k, 1+4k, --- \underbrace{1+xk}$

$1+xk \leq N$

$xk \leq N-1$

$$\boxed{x = \frac{N-1}{k}}$$

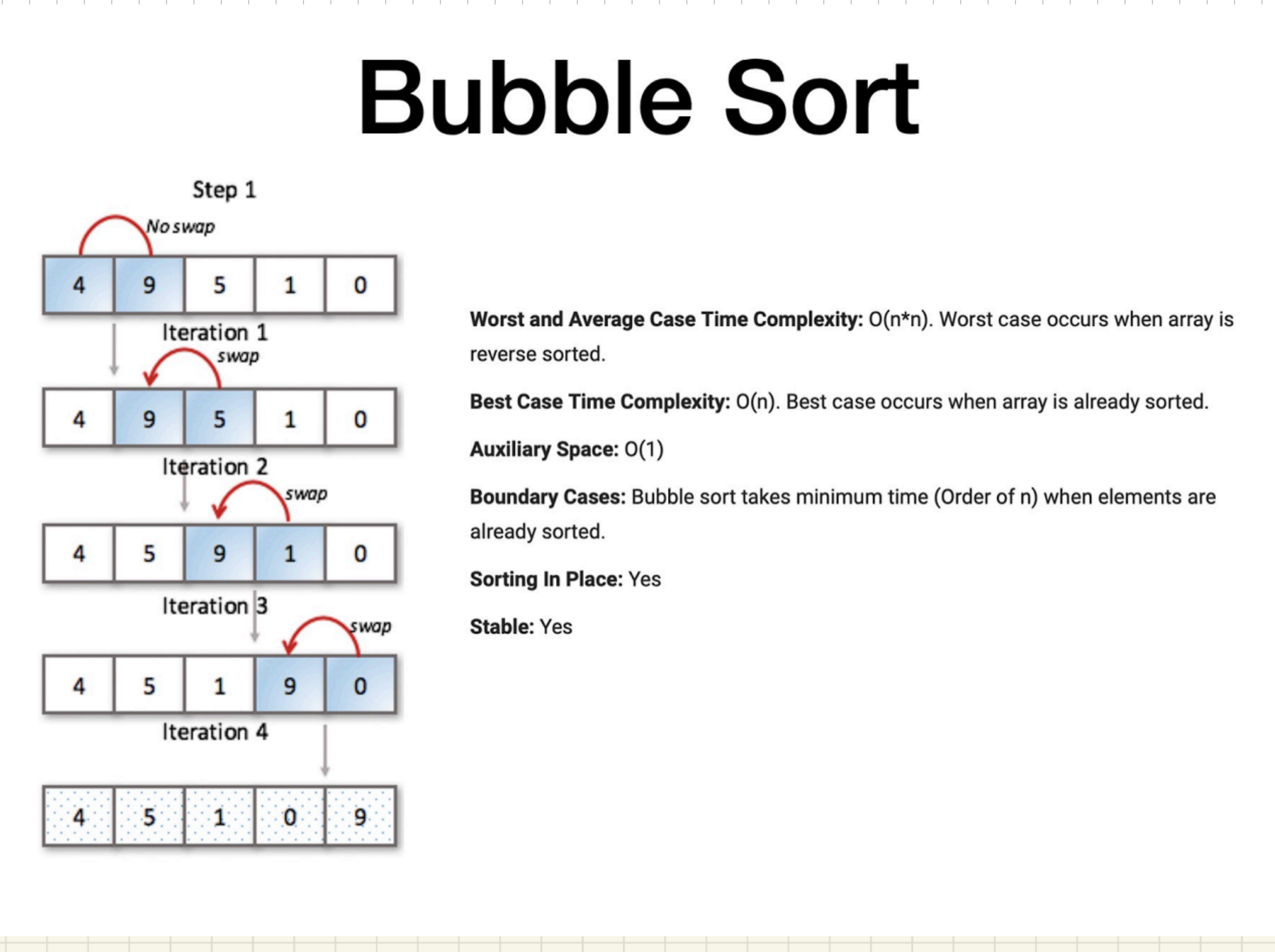
Times the outer loop is running

$O\left(kt + \frac{N-\cancel{1}}{k}\right)$

$= \underline{\underline{O(Nt)}}$

Ans

Bubble Sort



Bubble Sort

Worst and Average Case Time Complexity: O(n*n). Worst case occurs when array is reverse sorted.

Best Case Time Complexity: O(n). Best case occurs when array is already sorted.

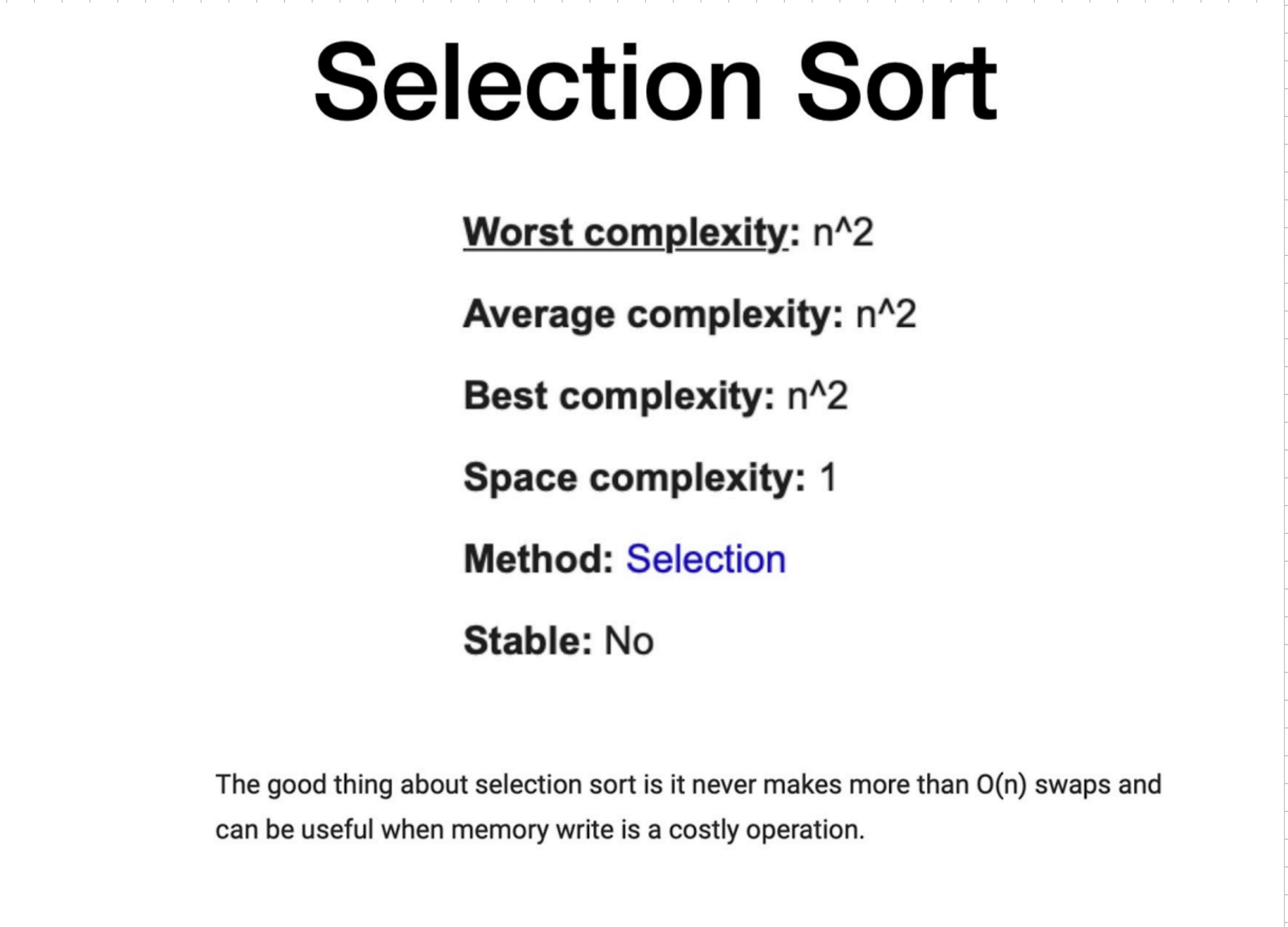
Auxiliary Space: O(1)

Boundary Cases: Bubble sort takes minimum time (Order of n) when elements are already sorted.

Sorting In Place: Yes

Stable: Yes

Selection Sort



Selection Sort

Worst complexity: n^2

Average complexity: n^2

Best complexity: n^2

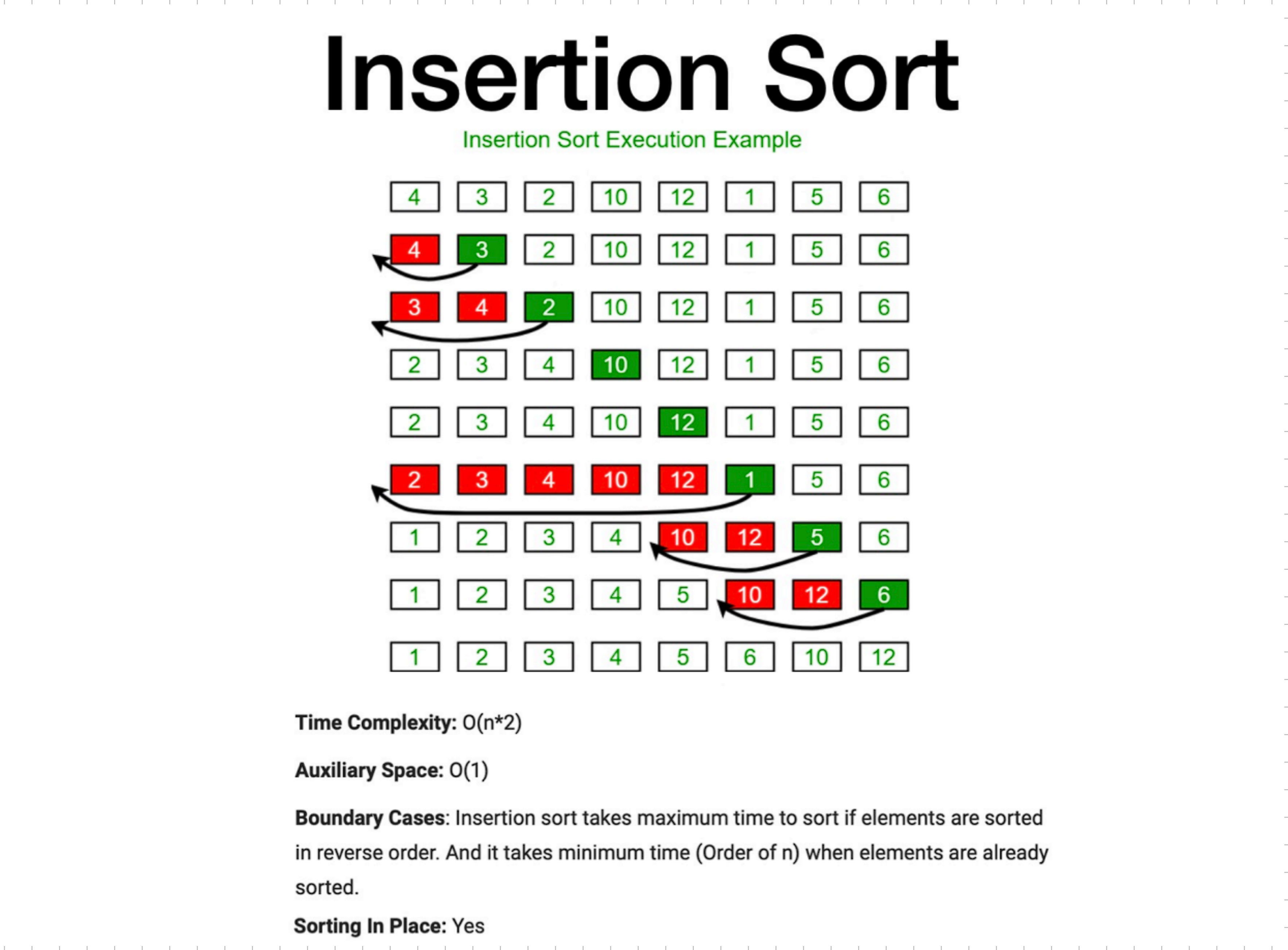
Space complexity: 1

Method: Selection

Stable: No

The good thing about selection sort is it never makes more than O(n) swaps and can be useful when memory write is a costly operation.

Insertion Sort



Insertion Sort

Insertion Sort Execution Example

4	3	2	10	12	1	5	6
4	3	2	10	12	1	5	6
3	4	2	10	12	1	5	6
2	3	4	10	12	1	5	6
2	3	4	10	12	1	5	6
2	3	4	10	12	1	5	6
1	2	3	4	10	12	5	6
1	2	3	4	5	10	12	6
1	2	3	4	5	6	10	12

Time Complexity: O(n*2)

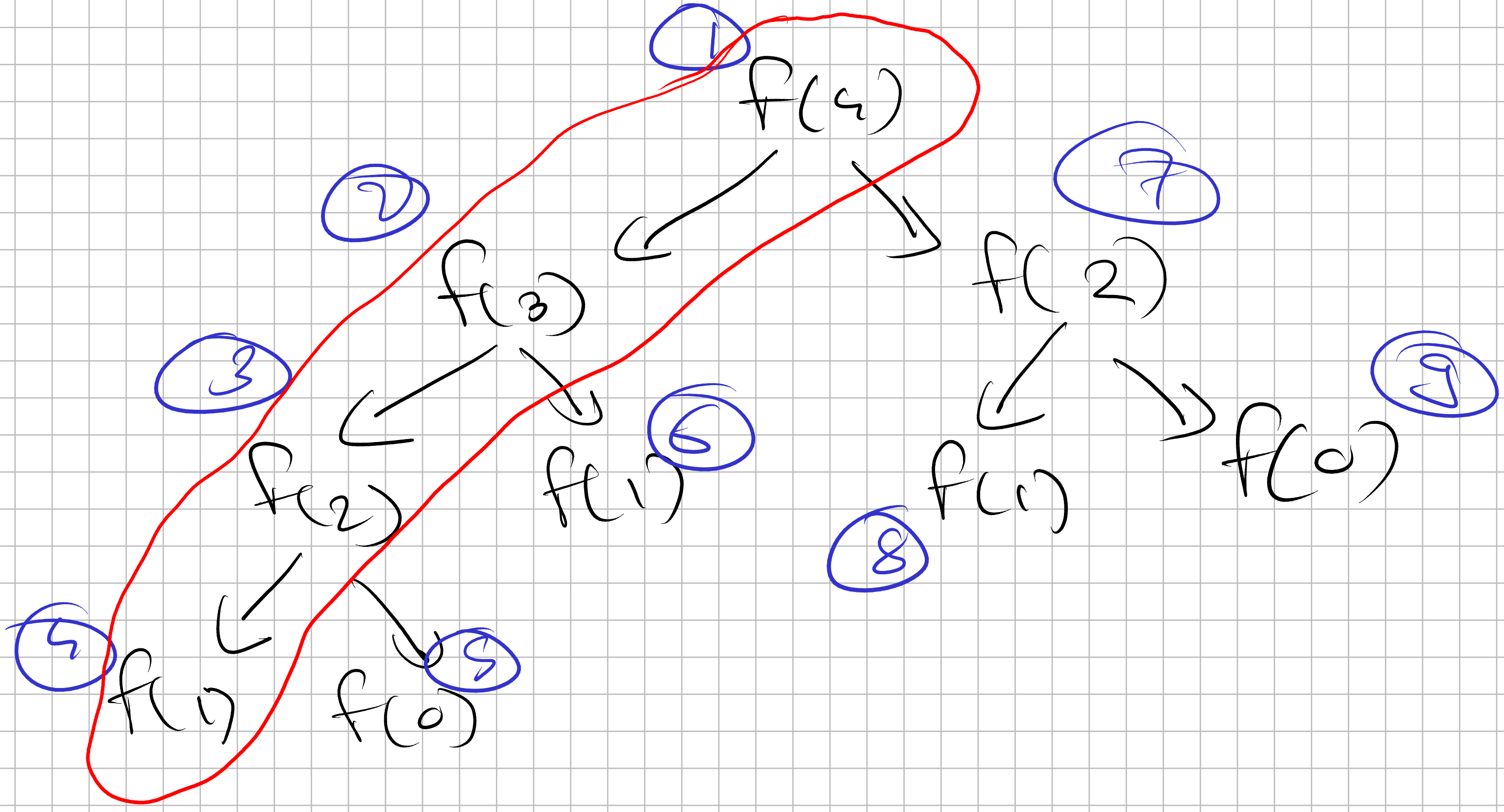
Auxiliary Space: O(1)

Boundary Cases: Insertion sort takes maximum time to sort if elements are sorted in reverse order. And it takes minimum time (Order of n) when elements are already sorted.

Sorting In Place: Yes

Recursive Algorithm

$O(N)$



Trick :- only call that are interlinked will be in the stack at same time.

Space complexity = Height of tree

Path