Analysis of algorithms

The field of computer science, which studies the efficiency of algorithms is known as analysis of algorithm.

Effeciency of the program depends on many parameters -

- System Level
- Software Level
- compiler Level, ... etc.

Prof. knuth -

Complexity of an algorithm is nothing but the resources required for the execution of an algorithm.

Mostly we use * Space → Running Time (physically measur

* Time → Time complexity

Time complexity →

* A mathematical quantity & it is going to give you an idea on running time.

Time complexity is something that we would like to have even before writting the Program

Analysis of Algorithm →

The mathematical estimation of time complexity & it is not an emperical study.

We need a mathematical way for obtaining the Time complexity.

How to do ...

1) The time complexity can be estimated through the instruction count involved in the algorithm.

Instruction count of which input?

Ex- Sorting.

Instruction count of -

→ 10 (or) nos

→ 20 (or) nos

.

.

.

→ 1000 (or) nos

.

.

.

2) Input Size → Measures the quantum of input processed.

Input Size ⟷ Instruction count.

$n \longleftrightarrow f(n)$

Ex

Algorithm 1

$1000 \cdot n$

Algorithm 2

$n \cdot \log_2 n$

which one is better ?

Asymptotic Analysis

Study the behaviour of the function as $n \to \infty$

As n becomes larger & larger how the algorithms are behaving ?

When will $n \log n$ overtakes $1000n$?

$$n \to \infty \quad \frac{n \log n}{1000 \cdot n}$$

$$\log_2 n = 1000$$

$$n = 2^{1000}$$

$n \to f(n)$ (first set of n inputs, second set of n inputs)

is ?

$I_n = \{ i_1, i_2, i_3, \ldots, i_t \}$

t possible inputs of size n.

$Ic(i_1)$, $Ic(i_2)$ $\ldots$ $Ic(i_t)$

$Ic(i_1)$ → Number of instructions taken to process first set of ilps by the algorithm.

There could be variation in instruction count among various inputs of same size, so which one should I take?

Here comes the notation of worst-case Time complexity.

$\text{Max}\{ Ic(i_1), Ic(i_2), \ldots, Ic(i_t) \}$

There is another notation of Average-case complexity.

$$\frac{(Ic(i_1) + Ic(i_2) + \ldots + Ic(i_t))}{t}$$

Avg1

Seq 1 → 99, 98, 98, 99, ..., 100

IC(1), IC(2)

Avg2

Seq 2 → 3, 4, 3, 5, 6, 3, 3, ..., 100

Avg(Seq1) ≥ 98

Avg(Seq2) ≥ 5

Finally which instructions in the algorithm considered ?

(1) Perform the instruction count of every task.

(2) Usually it is sufficient to consider the dominant operations (costly operations) & just count them only.

Ex $C_{n \times n} = A_{n \times m} \times B_{m \times n}$

$$C_{ij} = \sum_{k=1}^{m} a_{ik} \cdot b_{kj}$$

would you count both addition & multiplication (or) only one of them?

If you count running time →

i_1	i_2	. . .	i_n
α_1	α_2	. . .	α_n
t_1	t_2		t_n

$\alpha_1 t_1 + \alpha_2 t_2 + \ldots + \alpha_n t_n$.

Summary

- Before even going to the first stroke of the keyboard I must have an idea on how good is the method that I have in my mind.

Time complexity gives you insights

→ Time complexity of ... on the running time. It will not give you the exact value of running time.

→ we are interested in finding out the functional relation which is relating the instruction count with the size of input.

$n \longrightarrow f(n)$

→ Worst-case complexity and average case complexity are the two different cases in Time-complexity. (One more is there, which is Best-case complexity - opposite to the worst case complexity).

→ usually the dominant operations in the algorithm will be counted in Time-complexity.

Faster Algorithm (n)

Faster CPU ?

A faster algorithm running on a slower machine will always win for large enough instances.

Ex

- Suppose algorithm S_1 sorts n keys in $2 \cdot n^2$ instructions.
- Suppose computer C_1 executes 1 billion instructions/sec.

When $n = 1$ million

$$\text{Time} = \frac{2 \times 1,000,000 \times 1,000,000}{1 \times 1,000,000,000}$$

$$= 2000 \text{ sec.}$$

- Suppose algorithm S_2 sorts n keys in $50 \cdot n \log_2 n$ instructions.

Suppose computer C_2 executes

- Suppose computer executes 10 million instructions/sec.

$n = 1$ million

$$\text{Time} = \frac{50 \times 1{,}000{,}000 \times \log_2 1{,}000{,}000}{10 \times 1{,}000{,}000}$$

$$= \frac{50 \times 1{,}000{,}000 \times 20^2}{10 \times 1{,}000{,}000}$$

$$= 100 \text{ sec.}$$

ASYMPTOTIC ANALYSIS

The study of how algorithms are behaving as their input size becomes larger & larger.

Most often we are interested in the rate of growth of time

or space requirements
larger & larger instances
of the problem.

The complexity of an algorithm M is the function T(n) which gives the running time and storage space requirement of the algorithm interms of the size 'n' of the input data.

In Asymptotic Analysis,

i) Ignore the machine dependent constants.

ii) Look at the growth of T(n) as $n \to \infty$.

There exist mainly three

types of asymptotic notation

i) Big-oh (O) notation

ii) Big-Omega (Ω) notation

iii) Theta (Θ) notation

Ex-

Consider an Inventory system, where

- 10,000 ms to read inventory from disk.
- 10 ms to process each transaction

n-tractions ?

$(10{,}000 + 10 \cdot n)$ ms

more important if n large.

1) Big-oh Notation

- Let 'n' be the size of input of the program
- Let T(n) be a function
  eg. Running Time.
- Let f(n) be another function - preferably simplest one.

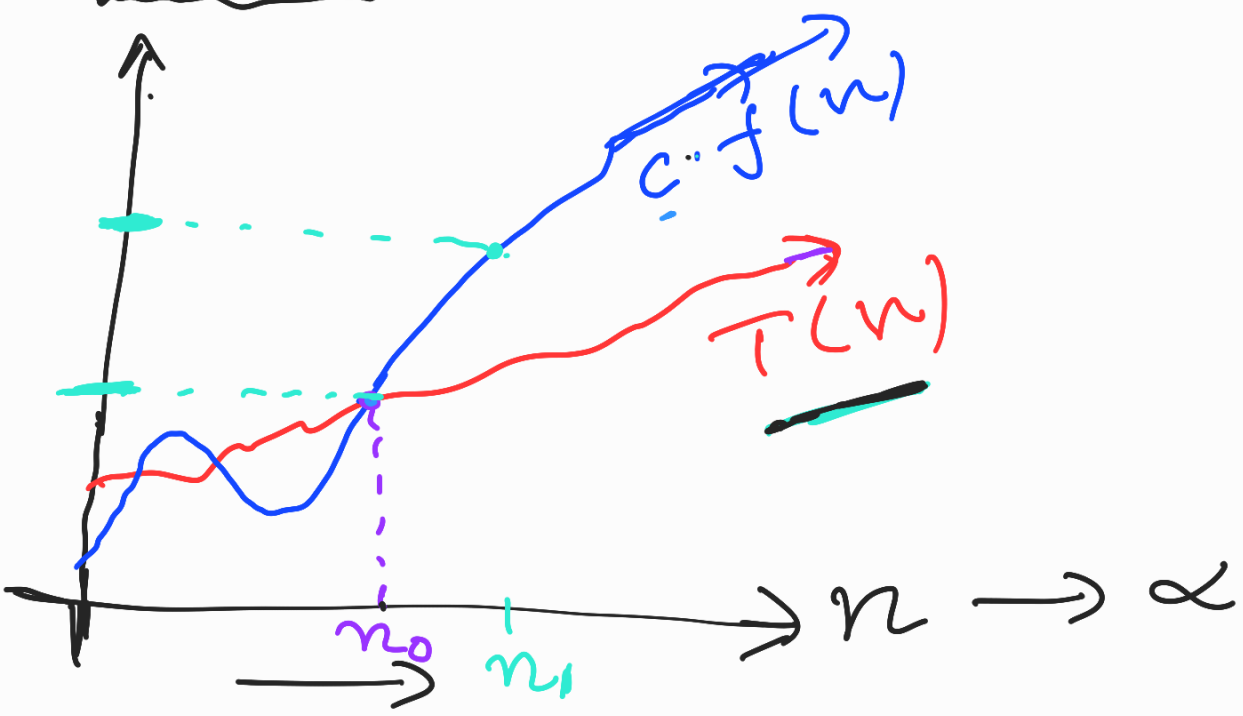
Big-oh notation gives the upper bounds.

Definition

O(f(n)) is the SET OF ALL functions T(n) such that:

There exist positive consts c & n_0 such that, for all $n \geq n_0$,

$$T(n) \leq c \cdot f(n).$$



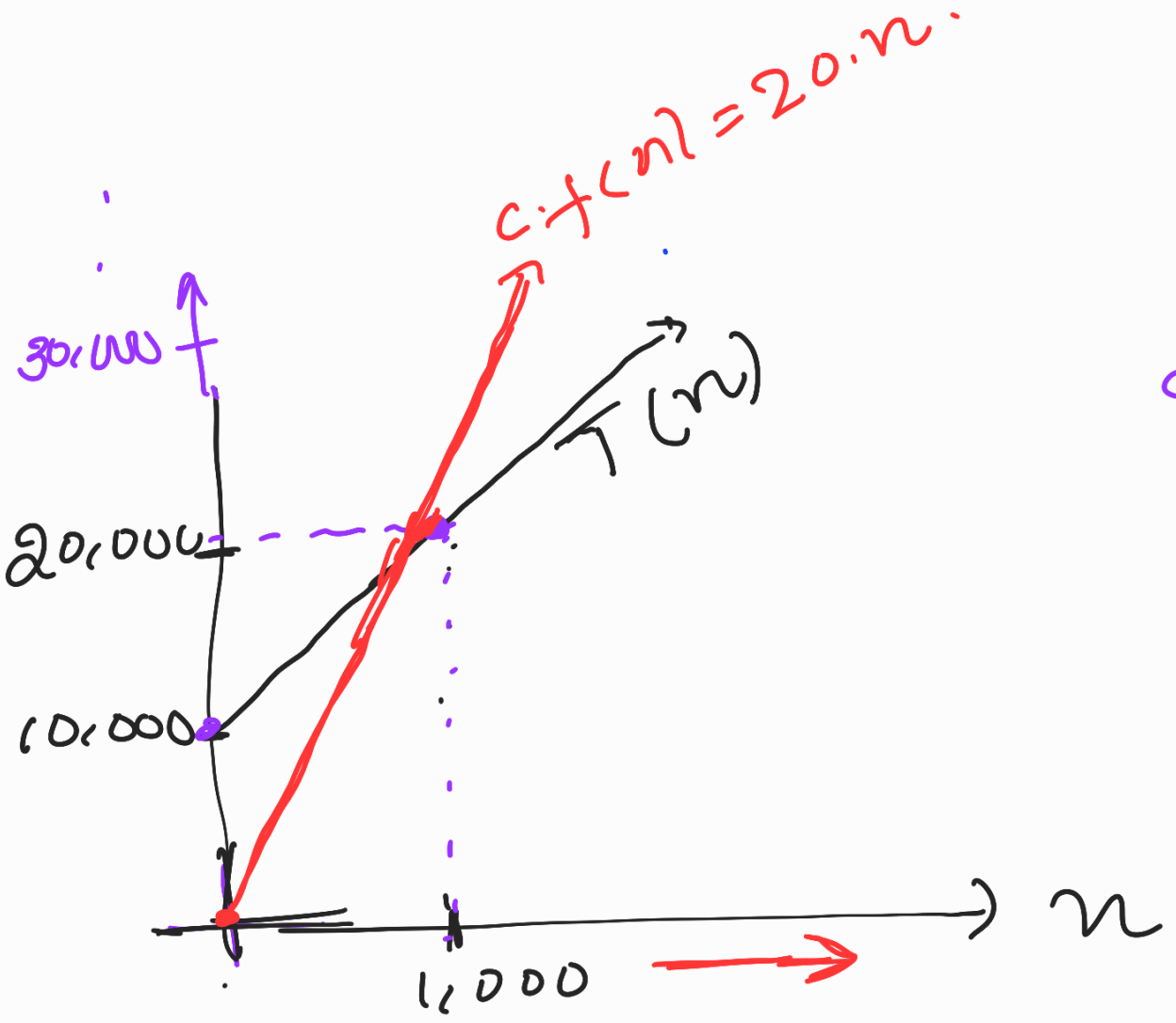
Ex: $T(n) = (10{,}000 + 10 \cdot n)$

Lets try $f(n) = n$

$T(n) \leq c \cdot f(n)$

$n \geq n_0$, c & n_0

Proof :



c = 20

For any $n \geq 1000$ & $c = 20$

$T(n) \leq c \cdot f(n)$.

Therefore

$T(n) \in O(f(n))$

$T(n) = O(f(n))$

Rules for finding Big-Oh

1) $f(n)$ is polynomial of degree d, then $f(n)$ is $O(n^d)$. i.e,

- Ignore the constant factors.
- Drop the lowest term.

Ex: $5n^3 + 6n^2 + 3 = f(n)$

$O(n^3)$

$2n^2 + n$

at $n = 5$

$2 \cdot 25 + 5$

at $n = 100$

$2 \cdot 10{,}000 + 100$

ii) Use the simplest form of function.

- Say $2n$ is $O(n)$ instead of $2n$ is $O(n^2)$ eventhough $2n \in O(n^2)$ is asymptotically correct.

iii) use the simplest expression of the class.

- Say $3n+5$ is $O(n)$ instead of $3n+5 \in O(3n)$

iv) If $T_1(n) = O(f(n))$ ✓

$T_2(n) = O(g(n))$ ✓

then

- $T_1(n) + T_2(n) = \max(O(f(n)), O(g(n)))$

- $T_1(n) * T_2(n) = O(f(n) * g(n))$

Examples

```
1. int search(int a[], int n,
              int k)
   { for(i=0; i<n; i++)
       { if(a[i]==k)
           return 1; }
     return 0;
   }
```

$T(n)$:

$i = 0;$ ⟶ 1 time

$i < n;$ ⟶ $n+1$ times

$i++;$ ⟶ n times.

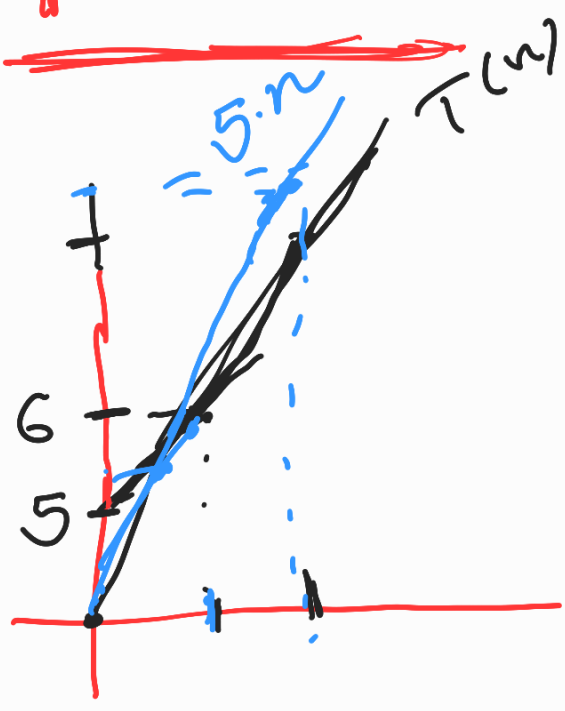
$if(a[i] == k)$ ⟶ n times

return 1/else 0 ⟶ 1 time.

$T(n) = 3n + 3$

$f(n) = ?$

$f(n) = n$



Proof: $c = 5$

$n_0 = 2$

for all $n_0 \geq 2$ & $c = 5$

$T(n) \leq c \cdot f(n)$

(2) $T(n) = n^3 + n^2 + n.$

$f(n) = n^3$

$c = ?$ $\quad n_0 = ?$

Set ~~$c = 1, \quad n_0 = 1$~~

for $c = 3, \quad n_0 \geq 1$

$T(n) \leq c \cdot f(n)$.

$T(n) \in O(n^4)$

Asymptotically correct.

24/8/21 $T(n) \leq c \cdot n^4$.

* Let $T(n) = 2n^2$

$f(n) = n^2$

Preferred one $\boxed{T(n) \in O(n^2)}$ ✓

$c = ?$ $\quad n_0 = ?$

$c = 3$ $\quad n_0 = 1$

$T(n) \in O(n^3)$ ✓

$T(n) \in O(n^4)$ ✓

$\vdots$

Function	Common Name
$O(1)$	constant
$O(\log n)$	Logarithmic
$O(\log^2 n)$	log-squared
$O(\sqrt{n})$	root
$O(n)$	Linear
$O(n \log n)$	n times log n
$O(n^2)$	Quadratic
$O(n^3)$	cubic
$O(n^4)$	Quartic
$O(2^n)$	Exponential
$O(e^n)$	

↓ $O(n!)$

$O(n^n)$

$O(n \log n)$ or faster ⇒ considered more Efficient

n^7 or slower time - considered useless

Warnings

→ Fallacious proof.

$n^2 \in O(n)$

proof: choose $c = n$.

then $n^2 \leq c \cdot n$

$\leq n \cdot n$

$\leq n^2$

Wrong → c must be constant

→ $e^{3n} \in O(e^n)$

Because constant factors doesnot matter.

$10^n \in O(2^n)$

→ Wrong

e^{3n} is bigger than e^n by a factor of e^{2n} & is not a constant.

10^n is bigger than 2^n by a factor of 5^n.

→ Big-oh notation doesn't always tell whole story.

$T_1(n) = n \cdot \log_2 n$

$T_2(n) = 100 \cdot n$

$T_1(n)$ dominates $T_2(n)$ asymptotically.

$\log_2 n$ is bigger than 100 if n is bigger than 2^{100}.

---

ii) Omega - Notation (Ω)

Def:

$\Omega(f(n))$ is the set of all functions that satisfy:

There exist positive constants c & n_0 such that for all $n \geq n_0$,

$$T(n) \geq c \cdot f(n)$$

Big-oh $T(n) \leq c \cdot f(n)$

If $T(n) \in O(f(n))$, then, $f(n) \in \Omega(T(n))$ & vice-versa

$2n \in \Omega(n)$, Because $n \in O(2n)$

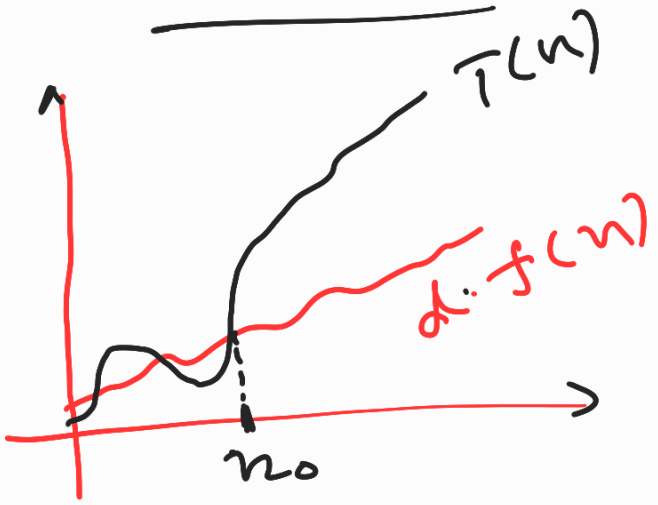
$n^2 \in \Omega(n)$, Because $n \in O(n^2)$

$n^2 \in \Omega(3n^2 + n\log n)$, because

$\Omega(3n^2) = \Omega(n^2)$

$3n^2 + n\log n \in O(n^2)$

Note: Omega gives a LOWER BOUND on a function.



Examples

$T(n) = 10n^2 + 4n + 2$

$T(n) = \Omega(n^2)$

$d = ?$ $\quad n_0 = ?$

$d = 9 \quad n_0 = 1$

$d = 10 \quad n_0 = 0$

$T(n) \in \Omega(n)$ ✓

$T(n) \in \Omega(1)$ ✓

$\in \Omega(2^n)$

→ $2n^2+5n-6 \notin \Omega(n)$

→ $2n^2+5n-6 \in \Omega(n^2)$ ✓

Observation

If $T(n) \in O(f(n))$ & $\in \Omega(g(n))$

then $T(n)$ is sandwitched between $\underbrace{c \cdot f(n)}_{\text{Max}}$ & $\underbrace{d \cdot g(n)}_{\text{Min}}$

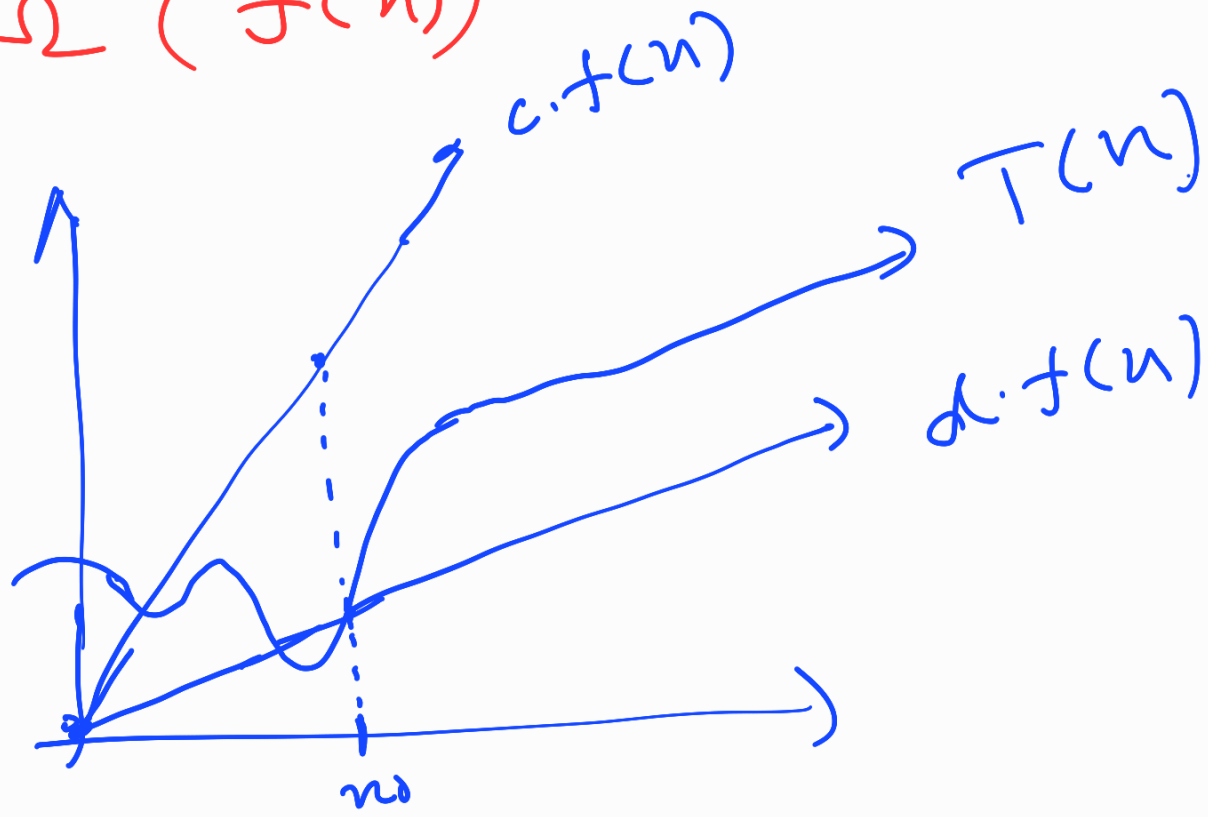
If $f(n)$ & $g(n)$ are the same functions then it leads to Θ notation.

Theta Notation

$\Theta(f(n))$ is the SET OF ALL FUNCTIONS $T(n)$

that are in $\Theta(f(n))$

& $\Omega(f(n))$



Example

→ $n^3 \in \Theta(3n^3 - n^2)$

$c = ?$ $\quad d = ?$ $\quad n = ?$

$n^3 \in O(3n^3 - n^2)$

$n^3 \in \Omega(3n^3 - n^2)$.

$c = 3 \quad n_0 = 1$

$d = \frac{1}{2}$.

$\Theta(n^2)$

→ $5n^2$ is in $O(n^2)$

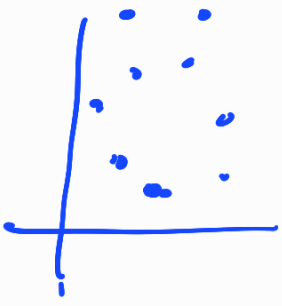
Proof: $c = 6$ $\quad d = 5$ $\quad n_0 \geq 1$

$$\underset{d \cdot f(n)}{5n^2} \leq \underset{T(n)}{5n^2} \leq \underset{c \cdot f(n)}{6 \cdot n^2}$$

→ $2n^2 + 5n - 6 \neq \Theta(2^n)$

→ $2n^2 + 5n - 6 \neq \Theta(n)$

→ Problem: Given a set of points P, find a pair closest to each other.



Approach:

calculate distance between every pair.

Return minimum.

$\frac{P \cdot (P-1)}{2}$ pairs.

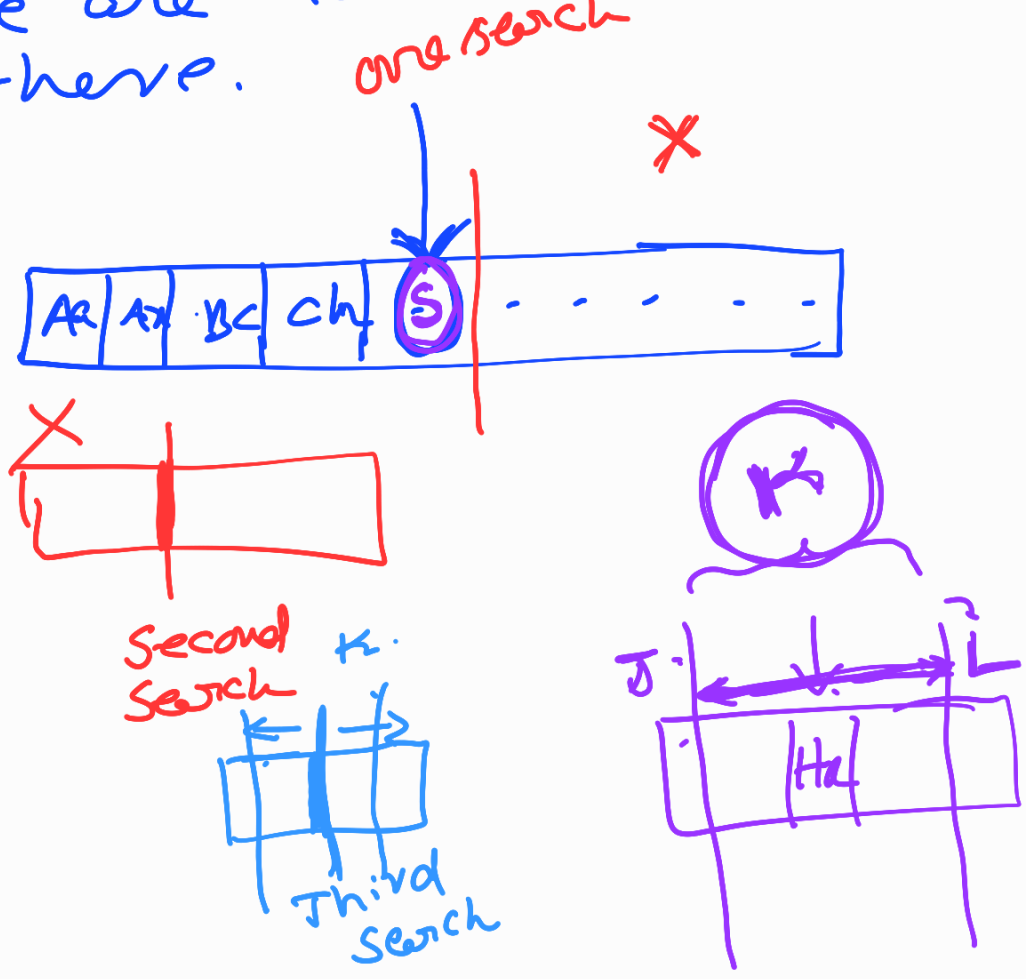
Each pair takes constant

time:

$$T(P) \in \Theta(P^2)$$

- Array contains n music albums sorted by title. You require a list of albums starting with letter H:

There are such a k albums are there.



Searches : $O(\log n)$

Total $T(n) \in O(\log n + k)$

$$T(n) = \omega^6 + \omega m + m^3$$

$$T(n) \in \Theta(\omega^6 + m^3)$$

Binary search

$$T(n) = 1 + T\left(\frac{n}{2}\right)$$

$$= 1 + 1 + T\left(\frac{n}{4}\right)$$

$$= 2 + T\left(\frac{n}{2^2}\right)$$

$$= 2 + 1 + T\left(\frac{n}{8}\right)$$

$$= 3 + T\left(\frac{n}{2^3}\right)$$

$$\vdots$$

$T(n) = T\left(\frac{n}{2}\right) + 1$

$$n = 2^k$$

$$k = \log_2 n$$

Master Theorem

Let $a \geq 1$ & $b > 1$ be constants. Let $f(n)$ be a function. $T(n)$ be defined

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

Then $T(n)$ has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a - \epsilon})$

for some constant $\epsilon > 0$,

then $T(n) = \Theta(n^{\log_b a})$

2. If $f(n) = \Theta(n^{\log_b a})$, then

$T(n) = \Theta(n^{\log_b a} \cdot \log n)$.

3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$
for some constant $\epsilon > 0$,
& if $a \cdot f(\frac{n}{b}) \leq c \cdot f(n)$
for some constant $c < 1$
& all sufficiently large n,
then $T(n) = \Theta(f(n))$

Examples

1) $T(n) = 9 \cdot T(\frac{n}{3}) + n$

$a = 9 \quad b = 3 \quad f(n) = n$

$n^{\log_b a} = n^{\log_3 9} = n^2$

$f(n) = O(n^{\log_b a - \epsilon})$

$n = O(n^{\log_3 9 - 1})$

$n = O(n^{2-1})$

$= O(n$

$T(n) = \theta(n^{\log_3 9}) = \underline{\theta(n^2)}$

2) $T(n) = T\left(\frac{2n}{3}\right) + 1$

$a = 1, \quad b = 3/2 \quad f(n) = 1$

$n^{\log_b a} = n^{\log_{\frac{3}{2}} 1} = n^0 = 1$

$f(n) = n^{\log_b a} = 1$

$T(n) = \theta(1 \cdot \log n)$

$= \theta(\log n)$

3) $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n \log n$

$b = 2 \quad f(n) = n \log n$

$a = 2$ b —

$n^{\log_b a} = n^{\log_2 2} = n^1 = n$

$n \log n$ is asymptotically greater than n

Master Theorem can not be applied.

4)

$T(n) = T\left(\frac{n}{2}\right) + 1$

$a = 1 \quad b = 2 \quad f(n) = 1$

$n^{\log_b a} = n^{\log_2 1} = n^0 = 1$

case 2

$T(n) = \theta(n^{\log_b a} \cdot \log n)$

$= \theta(1 \cdot \log n)$

$= \underline{\theta(\log n)}$