Set up the recurrance for no of multiplications and solve the order of growth

$$M(n) = M(n-1) + 1 \qquad n > 1$$
$$M(1) = 0$$

$$= [M(n-2)+1] + 1$$

$$= [M(n-3)+1] + 1 + 1$$

$$= M(n-i) + i \qquad n - i = 1$$
$$i = n-1$$

$$= 0 + n - 1$$

$$M(n) = n-1$$ order of growth $\in \Theta(n)$ iff class = linear

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

Write a recursive algo to find n^{th} Fibonacci no

Algorithm Fib (n)

// I/P :- int n (+ve)

// O/P :- n^{th} Fib no

if n = 0 return 0

if n = 1 return 1

else return Fib (n-2) + Fib (n-1)

I/P size :- Magnitude of n $n = 2^b$

Basic op :- +

The Basic op is independent of input type

Set up recurrance rel^n for this problem

$$F(n) = F(n-1) + F(n-2) \qquad F(0)=0,\ F(1)=1$$

Note:

Any recurrance of type

$$ax(n) + bx(n-1) + cx(n-2) = f(n)$$

if $f(n)$ is equated to zero for all values of n then it is homogeneous (if $f(n)=0$)

If not, non-homogeneous

2nd order linear recurrance with constant coefficients (degree of var is 1)

a, b, c are real constants

because $n, n-1, n-2$ is the parameter of linear var x

Consider a homogeneous case

$$ax(n) + bx(n-1) + cx(n-2) = 0$$

Sol^n is obtained using 3 formula

which formula to be used is based on the roots of Quadratic eq^n which is of form $ar^2 + br + c = 0$ (characteristic eq^n)

Case 1:- If roots are real & distinct i.e. $b^2 - 4ac > 0$

$$x(n) = \alpha r_1^n + \beta r_2^n$$

α & β → real constants
r_1, r_2 → root of QE

Case 2:- If roots are real and equal $b^2 - 4ac = 0$

$$x(n) = \alpha r^n + \beta n r^n$$

α & β → real constant
r → root

Case 3:- If roots are complex $b^2 - 4ac < 0$ with roots $(u \pm iv)$

$$x(n) = \gamma^n [\alpha \cos\theta + \beta \sin\theta]$$

α, β → real const
$\gamma = \sqrt{u^2 + v^2}$
$\theta = a\tan(v/u)$

Eg:- ~~$x(n) - 6x(n-1) + 9x0$~~

$x(n) - 6x(n-1) + 9x(n-2) = 0$, $x(0)=0$, $x(1)=3$

$a=1$
$b=-6$
$c=9$

characteristic eq^n : $r^2 - 6r + 9 = 0$

$r = 3$ real & equal

$x(n) = \alpha 3^n + \beta n 3^n$

$x(0) = \alpha 3^0 + \beta(0) 3^0$

$x(0) = \alpha$ $(\because x(0) = 0)$

$\alpha = 0$

$x(1) = \alpha 3^0 + \beta 3$

$x(1) = 1$

$3 = \beta 3$

$\beta = \frac{3}{3}$ $\quad \beta = 1$

$x(n) = n 3^n$

Now solving for Fibonacci recurrance rel^n

↓

i.e. explicit formula to find n^{th} fib no.

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

$F(0) = 0$
$F(1) = 1$

$F(n) - F(n-1) - F(n-2) = 0$

$a=1$
$b=-1$
$c=-1$

~~ax^2~~ $r^2 - r - 1 = 0$

$r_1 = 1.61803$
$r_2 = -0.61803$

$\alpha r_1^n + \beta r_2^n$

$\alpha r^n + \beta n r^n$

$b^2 - 4ac = 1 - 4(1)(-1) = 5$

$r_1 = \frac{-b + \sqrt{5}}{2a}$ $\quad r_2 = \frac{-b - \sqrt{5}}{2a}$

$\neq \frac{-b \pm \sqrt{5}}{2}$ Now $b = -1$ $\quad a = 1$

$\frac{1+\sqrt{5}}{2}$, $\frac{1-\sqrt{5}}{2}$

$F(n) = \alpha \left[\frac{1+\sqrt{5}}{2}\right]^n + \beta \left[\frac{1-\sqrt{5}}{2}\right]^n$

Let $n = 0$

$0 = \alpha \left[\frac{1+\sqrt{5}}{2}\right]^0 + \beta \left[\frac{1-\sqrt{5}}{2}\right]^0$

$= \alpha [1.6180]$

$0 = \alpha + \beta$ $\quad \alpha = -\beta$

Let $n = 1$

$1 = \alpha \left[\frac{1+\sqrt{5}}{2}\right] + \beta \left[\frac{1-\sqrt{5}}{2}\right]$

$= \alpha \left[\frac{1+\sqrt{5}}{2}\right] - \alpha \left[\frac{1-\sqrt{5}}{2}\right]$

$= \alpha + \frac{\alpha\sqrt{5}}{2} - \alpha + \frac{\alpha\sqrt{5}}{2}$

$1 = \frac{\alpha\sqrt{5}}{\alpha}$ α

$\alpha = \frac{1}{\sqrt{5}}$ $\quad \beta = -\frac{1}{\sqrt{5}}$

$F(n) = \alpha r_1^n + \beta r_2^n$

$F(n) = \frac{1}{\sqrt{5}} \left[\frac{1+\sqrt{5}}{2}\right]^n - \frac{1}{\sqrt{5}} \left[\frac{1-\sqrt{5}}{2}\right]^n$

$$F(n) = \frac{1}{\sqrt{5}}\left[\phi^n - \hat{\phi}^n\right]$$

where $\phi = \frac{1+\sqrt{5}}{2}$

$\hat{\phi} = \frac{1-\sqrt{5}}{2}$

$\phi = 1.618$

$\hat{\phi} = -0.618$

As $n \to \infty$, $\hat{\phi}$ tends to 0

$\therefore \frac{1}{\sqrt{5}}[\phi^n]$

$\phi = 1.618$ ⟷ called <u>golden ratio</u>

explicit formula to find nth fib no.

$$F(n) = \frac{1}{\sqrt{5}}[\phi^n]$$

<u>Golden ratio</u>

1) Fibonacci seq has role in plant study

Flower
Leaf
Seeds
} pattern follow fib sequence. 1 1 2 3 5

2) Entire body structure is based on Golden ratio
So its called <u>Divine Ratio</u>.

3) $a/b = \frac{a+b}{a}$ when $a > b$ then a & b are in golden ratio.

4) In cryptography we use Fib

<u>Mystery Algorithm</u>

$\frac{a}{b} = \frac{a+b}{a}$

No of +1 in fibonacci

$$A(n) = A(n-1) + A(n-2) + 1 \qquad A(0) = 0, \; A(1) = 0$$

In iterative → growth is linear

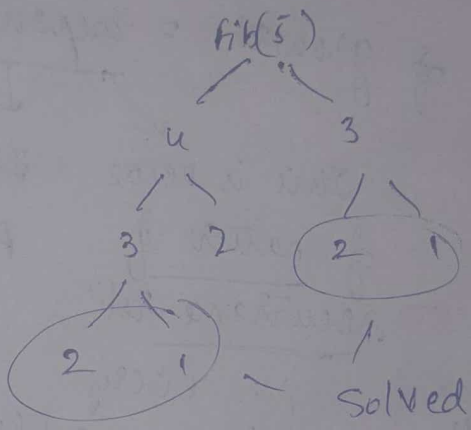
In recursion → it is exponential ($c\phi^n$)

So nature of recurrance

, recursion calls 2 more

tree of recursive calls is made

Here smaller subproblem gets computed again and again



← Solved again & again.

$$A(n) - A(n-1) - A(n-2) = 1$$

$$\Rightarrow r^2 - r - 1 = 0$$

↳ Non homogeneous linear eq^n

$$[A(n)+1] - [A(n-1)+1] - [A(n-2)+1] = 0$$

(+1) (-1) (-1) ↘ from RHS

Let $A(n)+1 = B(n)$

$$B(n) - B(n-1) - B(n-2) = 0$$

$a = 1 \quad b = -1 \quad c = -1$

$$r^2 - r - 1 = 0 \qquad \frac{+1 \pm \sqrt{1+4}}{2} = \frac{1+\sqrt{5}}{2} \; \& \; \frac{1-\sqrt{5}}{2}$$

$$r_1, r_2 = \frac{1 \pm \sqrt{5}}{2}$$

$$\alpha = \frac{1}{\sqrt{5}} \qquad \beta = \frac{-1}{\sqrt{5}}$$

$$B(n) = \frac{1}{\sqrt{5}} \left[\phi^{n+1} - \hat{\phi}^{n+1} \right]$$ ← using this

$$A(n) = \frac{1}{\sqrt{5}} \left[\phi^{n+1} - \hat{\phi}^{n+1} \right] - 1$$

Now $A(n)+1 = B(n)$

$A(0)+1 = B(0)$

$B(0) = 1$

$B(1) = 1$

$A(n) \in \Theta\ \phi^{(n+1)}$

order of growth = Exponential → No of + grows exponentially.

$n = \text{mag}^n = 2^b$

$A(n) \in \Theta\ \phi^{2b+1}$

↓ if measured wrt i/p size it becomes double exponential ∴ C

This is because of nature of recurrance rel? i.e. every recurrance calls 2 other recursions. Also n is reduced slightly i.e. n-1 & n-2.

This indicates that Fib recursive algo is very inefficient

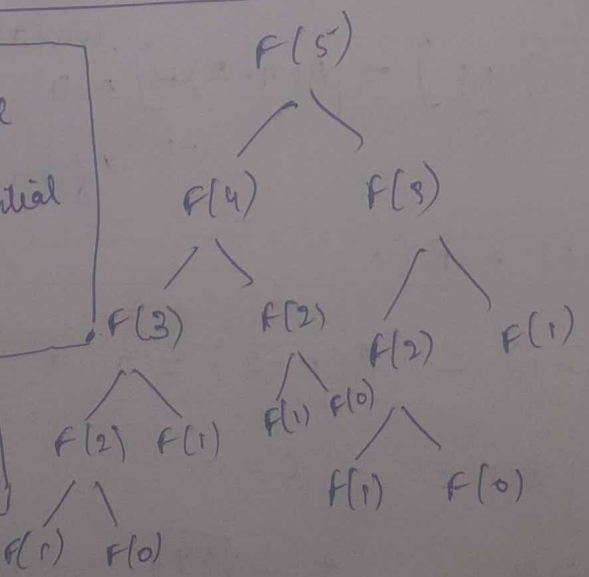
Tree of recursive func

Note:-

0.6^n, 1^n } They are not exponential

linear

log graph



Same subproblem is getting solved again & again

i.e. unnecessary computation.

Empirical Analysis → Theoritical Analysis does not yield correct order of growth. Also diff to compute

1)
2) Decide the metric < time / basic operation count

time < start time / end time } using time() function

i.e. ~~starttime()~~

$= \frac{\text{Starttime} - \text{Endtime}}{\text{no of clock ticks}}$ ←

```
Starttime = time()
algo()
Endtime = time()
```

3) Decide on input size (type of i/p)

4) Implement the algorithm

5) Generate the data → we need to examine for Asymptotic values of i/p size

→ Pseudorandom nos — rand() % n

from 1 to n-1

They may be same or not unique everytime → we need to give a starting point

← srand(time(NULL))

seed i.e. starting value

if not everytime same no is generated

Linear congruential Method

Algorithm Random(seed, m, a, b)

$r_0 \leftarrow$ seed

for $i \leftarrow 1$ to m do

$r_{i+1} \leftarrow (a * r_{i-1} + b) \bmod m$

end for

here
seed = starting value
m = range of generated nos from seed to (m-1)

6) Examine data and find order of growth and efficiency class.

Next method of algo analysis is Visualization → using graphics elements

(static) sect 2.7 → using applet viewer
Algorithm

dynamic visualization

↓

using images to convey useful info about algo

i.e. giving visual illustration of behaviour of alg.

→ series of still images to visualize algorithm

→ Animation → Movie like presentation

for learning & research.

discover hidden characters of algo.

2 disc of same colour never came in contact

in Tower of Hanoi

if odd disc → one colour

even disc → other colour

:)

Brute Force Technique → used as baseline for to derive different design strategy with ↑ efficiency

- Based on problem definition.
- Direct.

Eg:- $a^n = a \times a \times a \cdots \times a$ (n times) [Definition based]

Strengths of brute force design

→ Simplicity

→ Wide range of applicability – Any given problem Brute force sol^n always exists.

→ It yeilds reasonably good algorithm for several important problems. → like searching, sorting ~~for smaller i/p~~ with no limit on i/p size.

→ yeilds some efficient algorithm for important computational problems

Eg:- Sum of all elements → minimum is that we should scan all elements of array
Max of elements in array } linear → highest class in this case.

Weakness of brute

→ It rarely yeilds efficient algo
Only for some inp it is efficient

→ Algorithms which are derived with brute force technique are sometimes unacceptable (: very slow)

---

① Sorting problems

→ Selection sort.
→ Bubble sort } many are there using Brute force These 2 are examples.

Eg:

(15)	6	7	3	(1)	18	2
1	(6)	7	3	15	18	(2)
1	2	(7)	(3)	15	18	6
1	2	3	(7)	15	18	(6)

1	2	3	6	(15)	18	(7)
1	2	3	6	7	(18)	(15)
1	2	3	6	7	15	(18)

Algorithm SelectionSort(A[0 ... n-1])
// I/P :- Array A of n elements
// O/P :- Sorted array A in ascending order

```
for i ← 0 to n-2 do
    min ← i
    for j ← i+1 to n-1 do
        if A[j] < A[min]
            min ← j
    end for
    if (min != i)
    swap(A[i], A[min])
end for
```

$i \le n-2$
$i < n-1$

In above eg
0 1 2 3 4 5 6
7 elements
i varies from 0 to 5
so (n-2)

→ to reduce no of swaps

stop

Analysis

I/P size :- No of elements 'n'

Basic op :- comparision

It does not depend on type of input only on size

Even in sorted array

(1) 2 3 6 7 15 18

we need to check all

Again check all

Same for all I/P of same size

No diff cases need not be investigated

But we can reduce unnecessary swapping

By adding cond" if (min != i)

But basic op is same becoz swap is outside the inner loop. So Basic op count i.e. no of comparision is same for all cases.

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1$$

$$= \sum_{i=0}^{n-2} n-1-i-1+1$$

$$= \sum_{i=0}^{n-2} n-1-i$$

$$= n-1 \sum_{i=0}^{n-2} 1 - \sum_{i=0}^{n-2} i$$

$$= n-1 (n-2-0+1) - \frac{(n-2)^2}{2}$$

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

$$= (n-1)\left[n-1-\frac{n-2}{2}\right] = (n-1)\left[\frac{2n-2-n+2}{2}\right]$$

$$= \frac{n(n-1)}{2}$$

$90(98)/2$

} Basic op count shld match it in emperical analysis

$C(n) \in \Theta(n^2)$

Efficiency class = Quadratic :-)

Linear bcoz it is outside inner loop

inner loop prop of selection sort

For No of Swaps

worst case - Descending order

everytime swapping happens

$\therefore S(n) \in \Theta(n)$

best case - Ascending order

No of swaps = 0

$S(n) = 0$

This differentiates the selection sort from other i.e.

worst swap = linear

best = zero

Bcoz of that cond^n min != i

2 properties

→ Inplace ✓ → Bcoz no auxilliary array is used

→ +vely differentiates

→ Stability

Eg:- 2 | 5 3 | 1

i j

i < j

i' < j' → After sort

2' 2'' 1

1 2'' 2' Not stable

Even if cond^n min != i is there are not before sort

∴ in comparision we use A[j] < A[min]

② Bubble sort → max element is placed at proper pos at every iteration

Algorithm Bubblesort(A[0 · · · · n-1])
// I/P:- Array of n elements
// O/P:- Sorted array A in ascending order

```
for i ← 0 to n-2 do          → i < n-1 ; i ≤ n-2
    for j ← 0 to n-i-2 do    → j < n-i-1 ; j ≤ n-i-2
        if (A[j+1] < A[j])
            swap (A[j+1], A[j])
    end for
end for
stop.
```

very inferior sort when compared to many element -ary sorts

Analysis

I/P size :- No of elements (n)

Basic op :- Comparision

No different cases

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=0}^{n-i-2} 1$$

$$= \sum_{i=0}^{n-2} n-i-2-0+1$$

$$= \sum_{i=0}^{n-2} n-i-1$$

$$= \sum_{i=0}^{n-2} (n-1) - \sum_{i=0}^{n-2} i$$

$$= \frac{n(n-1)}{2}$$ (same as selection sort)

eff class. ~~growth~~ = Quadratic

$C(n) \in \Theta(n^2)$

No of swaps = Quadratic

Becoz inside both loops

in Descending order

$$S(n) = \frac{n(n-1)}{2}$$

n-1 for each i

total i's are n

$\therefore \frac{n(n-1)}{2}$

in Ascending order

$C(n) = 0$

If array is sorted or partially sorted then can we reduce eff class to linear? by removing unnecessary comparision?

If no swapping takes place in first iteration then the array is already sorted

If not sorted then atleast one sort takes place.

Modified / Better Bubble sort

```
for i ← 0 to n-2 do
    flag ← 0
    for j ← 0 to n-i-2 do
        if (A[j+1] < A[j])
            swap (A[j+1], A[j])
            flag ← 1
    end for
    if (flag = 0) stop
end for
```

Inside first loop.
After 1st iteration if no swaps then array is sorted so break
if after second iteration of no swaps then break.

$$\sum_{j=0}^{n-i-2} 1 = n-i-2+1$$

For $i=0$
i.e. 1st iteration

$= n-0-2+1$

$= n-1$

$\therefore$ $n-1$ times inner loop gets executed

No of comp $= n-1$

$C(n) \in \Theta(n)$ linear

$\therefore C(n)_{best} \in \Theta(n)$ [sorted]

$C(n)_{worst} \in \Theta(n^2)$ (worst)

To generate data

Avg = Quadratic → Practically

Best = Linear (Ascending)

Worst = Quadratic (Descending)

```
for i ← 0 to N do
    A[i] ← N + i + 10 → descending
           N - i + 10 → descending
```

Say N = 100

A[0] = 110

A[1] = 111

Inplace ✓

Stable ✓

becoz swap happens only for $<$ not $\leq$ so stable

$2_1\ 2_2\ 1 \leq 2_1\ 1\ 2_2$

$1\ 2_1\ 2_2$

What is the efficiency of brute force algo to compute a^n as a func of n and as a func of no of bits in binary representation of n

Sol^n :-

```
An ← a
for i ← 2 to n
    do
        An ← An * a
end for
```

$$M(n) = \sum_{i=2}^{n} 1$$

$$= n-1$$

$M(n) \in \theta(n)$ As a function of n its linear

$b = \log_2 n + 1$

$n = 2^b$

$M(n) \in \theta(2^b)$ → As a function of b (magnitude) it is exponential

---

Design a brute force algo to evaluate a polynomial

$P(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_2 x^1 + a_0$

Here degree = n

no of coeff = $n+1$

~~for~~ ALGORITHM Poly(P[0....n], x)

// compute the polynomial at a given Point x

// Input :- Array consisting of n+1 coefficients of polynomial of degree n, & point x
// O/P :- value of polynomial

Stored from lowest to highest
p[0] = a0
p[1] = a1
} could be reverse also.

```
p0 ← 0.0
for i ← n down to 0 do
    px ← 1
    for j ← 1 to i do
        px ← px * x
    end for   (→ we have x^i stored in px)
    poly ← poly + p[i] * px
                  (prev + coeff x x^i)
end for
return poly
Stop
```

computing from highest power to lowest power

Input size :- degree of polynomial
Basic op :- Multiplication
Basic op count :-

$$M(n) = \sum_{i=0}^{n} \sum_{j=1}^{i} 1$$

$$= \sum_{i=0}^{n} (i - 1 + 1)$$

$$= \sum_{i=0}^{n} i = \frac{n(n+1)}{2}$$

$$M(n) \in \Theta(n^2)$$

1 2 3
$3 \times \frac{4^2}{2}$

Efficiency class = Quadratic
Inefficiency is that we are computing x^i as though there is no relation between x^i & x^{i-1}
if 0 to n then $x^i \leftarrow x^{i-1} * x$
n to 0 then $x^{i-1} \leftarrow x^i / x$ → but ÷ by zero must be checked

∴ Modified algorithm

poly ← 0.0
for i ← 0 to n do
px ← 1

Now from lowest to highest

Horner's rule & problem reduction

M(n) = n

```
Algorithm Poly [p[ ], x]
    px ← 1
    poly ← p[0]
    for i ← 1 to n do
        px ← px * x
        poly ← poly + p[i] * px
    end for
    return poly
    stop
```

$$M(n) = \sum_{i=1}^{n} 2 = 2 \sum_{i=1}^{n} 1$$

$$= n - 1 + 1$$

$$= 2n = 2n$$

$M(n) \in \Theta(n)$ → linear

Largest degree possible in computer

i/p size find

100
1000
10,000
20,000

Apply selection sort to sort EXAMPLE
— " — bubble sort — " — " —

EXAMPLE
AXEMPLE
AEXMPLE
AEEMPLX
AEELPMX

AEELMPXX
AEELMPX

Is it possible to implement selection sort with linked list with the same efficiency class as array.

Assum LL to be single circular

```
for (ip = l → link; ip != l; ip = ip → link)
    temp ← ip
    for (jp = ip → link; jp != l → link; jp = jp → link)
        if (jp → info < mp → info)
            mp ← jp
    end for
    if (mp != ip)
    swap (mp → info, ip → info);
end for
```

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

∴ Quadratic

Yes same eff class as array

You have a row of binary digits arranged randomly. Arrange them in such an order such that all zeros precede all 1's (or vice versa)

Eg:- 01101001

Arrange like 00001111

The only constraint in arranging them is that you are allowed to interchange positions only if they are not identical. Find the order of growth of no of moves (i.e. swaps)

Sol:- Apply Bubble sort to compare adjacent elements to swap iff they are not identical

Bubble sort

0 1 0 1 1 0 0 1

Selection sort

0 1 1 0 1 1 0 0 1

Linear search → Brute force method of search

String matching

Application :-
Eg:- Google search engine
Spell checking
Forensic
Plagiarism

Text :- n characters

Pattern :- m characters $m \leq n$

Text : $t_0 t_1 t_2 \cdots t_i t_{i+1} \cdots t_{i+j} \cdots t_{i+m-1} \cdots t_{n-1}$

Pattern : $P_0 P_1 P_2 \cdots P_i \cdots P_{i+j} \cdots P_{m-1}$

Text : $t_0\ t_1\ t_2 \ldots t_i\ t_{i+j}\ t_{i+j}\ t_{i+m-1} \ldots t_{n-1}$

Pattern : $p_0\ p_1\ p_2 \ldots p_i\ p_j \ldots \ldots p_{m-1}$

Allign m characters of pattern with first m char of text
if all char pairs are matched return index of text (Now o)
if any mismatching pair is encountered
shift the pattern to right by one position & resume char comparision. repeatedly

Here we are returning 1st occurrance
if we shld search all occurance then repeat algo.

Text : 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
NOBODY — NOTICED — HIM n=18

Pattern: NOT m=3

Allign
character pair comparision
Mismatch pair is encountered so shift pattern by 1 position

```
NOBODY — . . .
 NOT
```

Mismatch shift

```
NOBODY — . . .
  NOT
```

Mismatch

```
NOBODY
 NOT
```

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
NOBODY — NOTICED — HIM

No of comp: 3

```
NOT
 NOT
  NOT
   NOT
    NOT
     NOT
      NOT
       NOT
        3 NOT
```

∴ ① Allignment from 0 to n-m

② Consider inner loop max no of char comparision is m min 1 1 to m

worst case or all pattern occurrance is to be checked or pattern does not exist

Eg: NOK → Last allignment

Total no of allignments = $n-m+1$ Here = 16
(∵ 0 to n-m)

It must ~~the~~ be at pos m-n

Here m = 18, n = 3 } last allig = 15
i.e. m-n
after that pattern goes out of text

Suppose in worst case = $(n-m+1)\,m$
in every allignment m comparision
i.e. every first n (m-1) are matching only mth char is not matching

Here = 16 × 3 = 48

Algorithm StringMatching [T[0...n-1], P[0...m-1])

// Searches for pattern P in the text T
// I/P :- Text T of n characters and pattern P of m characters
// O/P :- Returns 1st occurrance of pattern in Text if successful else returns -1

```
for i ← 0 to n-m do
    j ← 0
    while j < m and P[j] = T[i+j]
        j ← j+1
    end while
    if (j = m) return i
end for
return -1
```

I/P size :- is more than 1 parameter
i.e. m & n
pattern & text length.

Basic operation :- character pair comparision

The basic op is different for diff I/P type of the same size

Best case :- The search pattern exists in the 1st allignment or 1st indent of text

$C_{best}(n) \in \Theta(m)$

No of comparisions = m

Worst case :- successful if pattern is in last allignment (i.e. last m character)
unsuccessful if pattern does not exist

$C_{worst}(n) \in \Theta(n-m+1)(m)$ → first m-1 are not matching on every loop, only last one char is to be

$C_{worst}(n) \in \Theta(nm - m^2 + m)$

$$C_{worst}(n) \in \sum_{i=0}^{n-m} \sum_{0}^{m-1} 1$$

$$= \sum_{i=0}^{n-m} (m-i+1)$$

$$= m \sum_{i=0}^{n-m} 1$$

$$= m\,[n-m-0+1]$$

$$= m(n-m+1) = nm - m^2 + m$$

exact no of comparision

$\therefore C_{worst}^{succ/unsucc}(n) \in \Theta(nm)$

as n & m tends to ∞.

In natural language text we expect more shifts to happen with less comparisions

i.e In every alignment usually first m-1 will not match always eg NOBODY NOT NOT → 1st comparision only shift happens

Average case efficiency is much better than worst case efficiency.

Empirically, Average case is linear (proved)

$$C_{avg}(n) \in \Theta(n+m)$$

reduced to $C_{avg}(n) \in \Theta(n)$ [i.e. linear in terms of n]

Space time tradeoffs
i.e ↑ space is used ↓ time } or vice versa

using this we can do string matching.

For smaller instances we use Brute force instead of adding complexities ↓ i.e. avg is linear
∴ smaller instance of text (n) length

→ Horsepool
→ } Based on space-time tradeoff :)

Find the number of character comparisions while searching for GANDHI in the text

THERE_IS_MORE_TO_LIFE_THAN_INCREASING_ITS_SPEED

Here $m = 6$ $n = 47$

Search pattern does not exist

∴ Worst case = $(n - m + 1)$

No of alignment = $(47 - 6 + 1)$

= 42

INCREASING ITS

GA → 2 comp

for 41 alignments → 1 comp = 41

1 alignment → 2 comparision + 2

= 43 comparisions

Find the No of character comparision while searching for

P: ABABC T: BAABABABCCA

$m = 5$

$n = 11$

Pattern exists in

0 1 2 3 4 5 6 7 8 9 10

B A A B A B A B C C A

total alignments = 5

1st 1

2nd 2

3rd 5

4th 1

5th 5

14 comparisions → n = 11 Avg case (linear)

Find the number of comparisions made for Brute force string matching for searching for following patterns in the text of one thousand zeroes (i) 00001
(ii) 10000
(iii) 01010

n = 1000
m = 5 (in all 3)

(i) 00001

Worst case alligment = (1000 − 5 + 1) → or by direct formula for worst case (n − m + 1) m
= 996 = 996 × 5
no of comp in each alligment = 5 = 4980

(ii) 10000

worst case alligment = 996 = 996
no of comp in each = 1

(iii) 01010

996 × 2 = 1992

Avg case depends on text length

Data generation

Text :- All zeroes / ones
Worst case :- $(m-1)_{0's}$ m^{th} 1 / $(m-1)_{1's}$ m^{th} 0
Best case :- m zeroes / m ones
Worst case :- Random.

```
for i ← 0 to n do
    T[i] ← 0
```

Worst case

```
for i ← 0 to m-1 do
    P[i] ← 0
end for
P[i] ← 1   // last mth char is 1
```

Best case

```
for i ← 0 to m do
    P[i] ← 0
```

n	Ar	Basic op
100	10	$nm - m^2 + m$
	20	910
	⋮	
	100	
1000	m	
	10	
	20	
	⋮	
	1000	

Worst case

91 × 10

$P[m-1] = 1$ i.e. last char

10,000 m

Average case

```
for i ← 0 to n
    p[i] ← i-1+2        → even i = 0
                          odd i = 1
(or) p[i] ← rand()%2
```

---

consider the problem of counting in a given text the no of substrings starting with a and ends with b

Eg:- C A B A A X B Y A

Here 4 ← A B, A B A A X B, A A X B, A X B

```
Algo countstring [t[0...n-1], a, b)
{
    count ← 1
    for i ← 0 to n-1
        j ← i
        while (t[j] != B && j > n)
            if [t[j] = a)
                count ++
            j ← j+1
        end while
        i ← i+j
    end for
    return count
}
```

X complexity :- C

Found a b
For a B found in string no of A's to left of B

```
Scnt ← 0   Acnt ← 0
for i ← 0 to n-1 do
    if T[i] = 'A'   Acnt ++
    else T[i] = 'B'   Scnt ← Scnt + Acnt
end for
```

Eff class = linear

(or)

```
Scnt ← 0
for i ← 0 to n-2 do
    if [T[i] = A)
        for j ← i+1 to n-1 do
            if [T[j] = B)
                Scnt ← Scnt + 1
        end for
end for
```

For an A found in string find no of B's to right of A

Eff class = Quadratic

$$c(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1$$

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

Exhaustive Search

Some problems are combinatorial

No of solutions search of are in terms of

factorial ($n!$) or exponential (2^n)

Permutations, Combinations — Power set

combinatorial problems – Problems which involves combinatorial objects

↳ They are computationally very hard

Exhaustive search – Brute force methods to solve combinatorial problems.

↓

Name beco2

we generate all possible sol^n without leaving any one

i.e. exhaustively generate sol^n.

Among them we have to select best.

∴ They are optimization problems

↳

we need to maximize or minimize the characteristic fun^n

Eg:- maximize profits

minimize loss.

Among many sol^n

2^n or $n!$ in combinatorial problem

we need to select

∴ They are → Exhaustive search

Combinatorial problems

↓

optimization problems

↓

All possible sol^n

↓

feasible sol^n → sol^n which satisfies the constraints of problems

optimal sol^n

→ Travelling sales man problem
→ Knapsack problem
→ Assignment problem
} combinatorial problems.

Travelling sales man problem (TSP)

Salesman should start from one city and travels only once all cities before coming back to starting city with minimum distance.

Here characteristic fun^c is distance
constraint is visit each city only once

So suitable d.s is undirected weighted graph

∵ no constraints for d^o

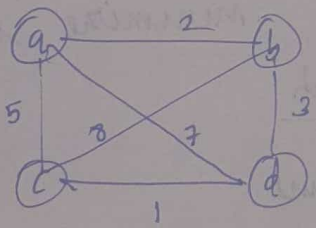
distance is weight.

using
adjacency matrix
or
adjacency linked lists

Lets consider worst case
complete graph

best case { Connected graph → min no of edges $n-1$

worst case { → max no of edges $\frac{n(n-1)}{2}$ Complete graph



→ complete

So matrix is suitable

Euler ckt :- all edges only once

Hamiltarion ckt :- all vertices only once and comes to start point

If sparsed i.e min no of edges then lists is suitable

↓ feasible sol^n of this problem

↓ In this problem we need to find shortest Hamiltarion cycle.

Lets consider starting point as a we examine exhaustively all the tours.

$a\,^{2}b\,^{8}c\,^{1}d\,^{7}a = 18$ dist

$a\,^{2}b\,^{3}d\,^{1}c\,^{5}a = (11)$

$a\,^{5}c\,^{1}b\,^{3}d\,^{7}a = 23$

$a\,^{5}c\,^{1}d\,^{3}b\,^{2}a = (11)$

$a\,^{7}d\,^{3}b\,^{8}c\,^{5}a = 23$

$a\,^{7}d\,^{1}c\,^{3}b\,^{2}a = 18$

all are feasible sol^{n}

Here optimal is dist = 11

Here $n = 4$.

starting at a we can generate maximum of 6 tours.

We have generated by excluding start and end (a) $\therefore (n-1)!$

$3! = 6$

Here 3 pairs differ only by d^{n}

abcda } = 18 abdca = 11

adcba } = 18 acdba = 11

$\therefore$ We can cut down the sol^{n} to half $\frac{(n-1)!}{2}$

But as $n \to \infty$ the growth of no of sol^{n} is in terms of factorial

Becoz it is undirected graph

So differ only by d^{n}

among all we need to find optimal one

For 1 sol^{n} the no of +^{n} is $(n-1)$ (linear)

Eg: a b c d a

1 (+) 8 (+) 1 (+) 7

$\therefore$ Efficiency class is in terms of factorial

$\therefore$ For $(n-1)!$ we need

$\approx n(n-1)!$:- no of +^{n}

$\therefore$ $C(n) \in \Omega(n!)$

greater than $n!$ ($\because n(n-1)!$) $\therefore$ Not efficient

$\therefore$ This shows how Brute force yeilds very inefficient sol^{n}

combinatorial problems

NP-hard problems :- Non deterministic polynomial problems

computationally hard → i.e we can't solve in polynomial type like $n, n^{2}, n^{3} \ldots$

Exhaustive search works for only very small values of n.

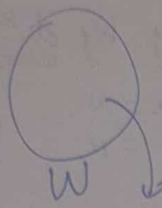
Knapsack problem

w_1 w_2 - - - - w_n

v_1 v_2 - - - - v_n

n items associated with weights $w_1, w_2 \ldots w_n$ and value $v_1, v_2, v_3 \ldots v_n$

Now we have Knapsack of weight W



we need to find the most valuable subset of items in weight W of Knapsack to fit into Knapsack.

put most valuable subset into it.

Now Brute force method i.e. exhaustive methods.

Generate all subsets of items

0 item
1 items (n)
2 items :

Eg:-

Item	Wt	value
1	7	42$
2	3	12$
3	4	40$
4	5	25$

W = 10
Knapsack

Manually we find that optimal is item 3 & 4

W = 9
Value = 65$

feasibility :- max weight can't exceed 10

Now generate all sol? (feasible)

		value
$\{\phi\}$	–	0
$\{1\}$	–	42$
$\{2\}$	–	12$
$\{3\}$	–	40$
$\{4\}$	–	25$
$\{1,2\}$	–	54$
$\{1,3\}$	X	not feasible
$\{1,4\}$	X	
$\{1,2,3\}$	X	
$\{1,3,4\}$	X	
$\{1,2,4\}$	X	
$\{2,3,4\}$	X	
$\{2,3\}$	52$	
$\{3,4\}$	65$.	
$\{2,4\}$	37$	
$\{1,2,3,4\}$	X	

a feasible

optimal – $\{3,4\}$

with value 65

$\in \Omega(2^n)$

Becoz we need to perform many +^ & comparisions

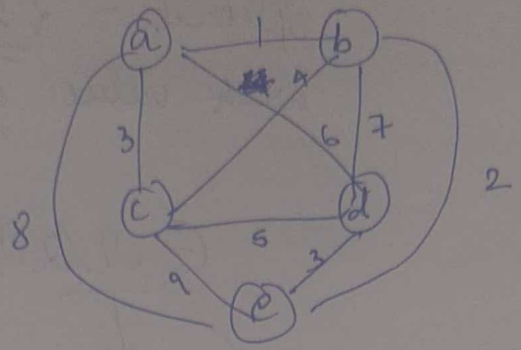
if all 2^ are feasible to obtain optimal from them we need 2^ comparisions.

∴ Exponential.

Item	wt	value
1	2	12$
2	1	10$
3	3	20$
4	2	15$

W = 5

Assignment Problem



complete graph

Assignment problem

n - people
n - job
} each person only one job
one job by only one person.

→ cost of executing j^{th} job by i^{th} person is a known fact

→ Problem is to find optimal sol^n by minimizing the cost

Cost matrix

	J_1	J_2	J_3	J_4
P_1	9	2	7	8
P_2	6	4	3	7
P_3	5	8	1	8
P_4	7	6	9	4

Feasible sol^n means here constraint is one person one job only.

Job 1 to 1

<1 2 3 4> = 9+4+1+4 = 18

<1 2 4 3> = 30

<1, 3, 2, 4> = —

<1, 3 4 2> = —

<1, 4 2 3> = —

<1 4 3 2> = —

1st set

Job 1 to 2

<2 1 3 4> = —

<2 1 4 3> = —

<2 3 1 4> = —

<2 3 4 1> = —

<2 4 1 3> = —

<2 4 3 1> = —

2nd set

3rd set 3 :

4th set 4 :

Totally 24 assignments

i.e $n!$ assignments

among $n!$ we shld find optimal

$\in \Omega(n!)$

Factorial → we need to add in each $n!$ & compare among $n!$ so Big omega

---

For all problems Brute force sol^n always exist

Enough for smaller instances of combinatorial problems → no need to apply more sophisticated strategy for smaller instances.

if we are able to acheive same efficiency for small instances using Brute force

---

Divide & Conquer

1. Divide the problem into smaller problems (preferably of same size)
2. Solve each subproblem individually (by recursion)
3. If required combine the soln of smaller instance to obtain soln of large instance (original)

Eg: ① n = 1 to 100 sum

S(50) S(50)

combine for original problem.

But for this Brute force is ideal.

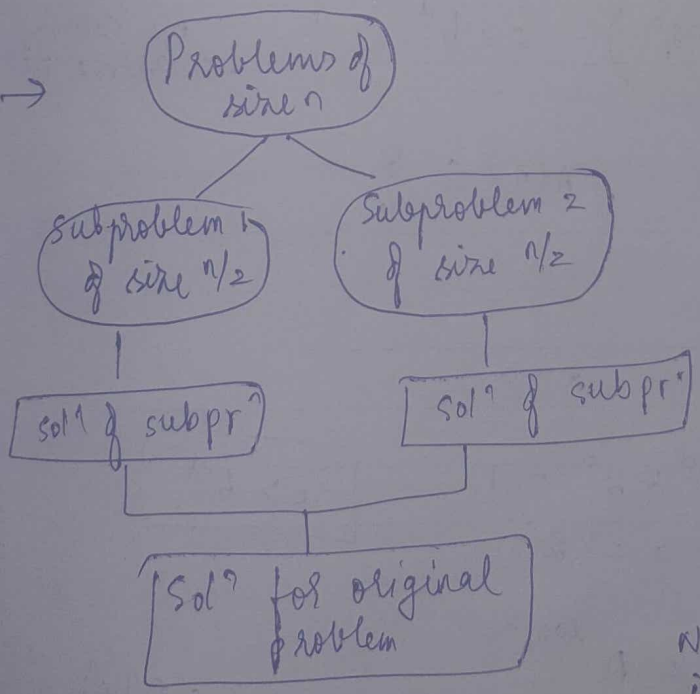
→ Divide & conquer is ideally suitable for parallel computation

makes sense only with parallelization

Eg ② a^n — $a^{n/2}$. $a^{n/2}$ → suitable if parallel computation is done.

combine

If only one person Then brute force is ideal



Diagrammatic representation

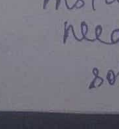
→ Give problem of size n

→ Divide into b instances each of size n/b

→ a is the no of instances to be solved among b instances

Not all instances needs to be solved

Eg:- Binary search



General recurrance relation of Divide & conquer

$$T(n) = a\,T(n/b) + f(n)$$

- $T(n)$: time complexity
- a : no of instances to be solved among b instances
- $T(n/b)$: time required to solve instance of size n/b
- $f(n)$: time required in splitting and merging

$a \geq 1$ → atleast one instance shld be solved

$b > 1$

no of instances shld be greater than 1

depends on
- → a
- → b
- → growth of $f(n)$

n^0, n^2, n^3

$f(n) = s(n) + m(n)$

- $s(n)$: time required to split
- $m(n)$: merge

Master Theorem :- Used to find eff class after setting recurrance rel' even without solving recurrance rel'

↓ Assumes that $f(n)$ growth is polynomial

$T(n) = a\,T(n/b) + f(n)$

if $f(n) \in \Theta(n^d)$ where $d \geq 0$ (if $d=0$ const, $d=1$ linear)

Then

$$T(n) \in \begin{cases} \Theta(n^d) & a < b^d \\ \Theta(n^d \log_b n) & a = b^d \\ \Theta(n^{\log_b a}) & a > b^d \end{cases}$$

gives only order of growth

Not exact growth ↓ by solving recurrance rel'

Eg:- $A(n) = 2A(n/2) + 1$; $A(n) \in ?$

Here $a = 2$

$b = 2$

$f(n) = 1 \quad d = 0$

$b^d = 2^0 = 1$

$a > b^d$

$a = 2$

$= n^{\log_2 2}$

$= n^1$

$\therefore A(n) \in \Theta(n)$

Eff class = linear

eneral recurrance relation

$$T(n) = a\,T(n/b) + f(n)$$

$a \geq 1$ → atleast one instance shld be solved

$b > 1$ no of instances shld be greater than 1

- a : no of instances to be solved among b instances
- $T(n/b)$: time required to solve instance of size n/b
- $f(n)$: time required in splitting and merging

$$f(n) = s(n) + m(n)$$

$s(n)$: time required to split; $m(n)$: merg

$n^{\log_b a}$

n^2

n^3

Theorem :- Used to find eff class after setting recurrance rel^n even if growth is polynomial without solving recurrance rel^n

$T(n/b) + f(n)$

$f(n) \in \Theta(n^d)$ where $d \geq 0$ (if $d=0$ const, $d=1$ linear)

$$T(n) \in \begin{cases} \Theta(n^d) & a < b^d \\ \Theta(n^d \log_2 n) & a = b^d \\ \Theta\left(n^{\log_b a}\right) & a > b^d \end{cases}$$

gives only or
of growth
Not exact gr
by solv
recurrance

2) $A(n) = 2A(n/2) + n$

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

$b^d = 2^1 = 2$ $\boxed{a = b^d}$

$n^1 \log_2 n$

$\therefore A(n) \in \Theta(n \log_2 n)$

linearithmetic

3) $A(n) = 2A(n/2) + n^2$

$a = 2 \quad b = 2 \quad f(n) = n^2 \quad d = 2$

$b^d = 2^2 = 4 \quad a < b^d$

$A(n) \in \Theta(n^2)$

Quadratic

4) $A(n) = 2A(n/4) + n^2$

$a = 2 \quad b = 4 \quad f(n) = n^2 \quad d = 2$

$b^d = 4^2 = 16 \quad a < b^d$

$A(n) \in \Theta(n^2)$

Quadratic

5) $A(n) = 4A(n/4) + n^2$

n^2

6) $A(n) = 4A(n/4) + n$

$n \log n$

Binary search — $C_{best}(n) = 1$ (middle element)

$\in \Theta(n)$

Now for worst case $\boxed{T(n) = T(n/2) + 1}$

$a = 1 \quad b = 2$

$f(n) = 1$

↓

no merging

splitting can be done in const time by using index of middle element

We divide problem to 2 instances and solve only 1

$d = 0 \quad b^d = 1 \quad a = b^d$

$n^0 \log_2 n = \log_2 n$

$\therefore C_{worst}(n) \in \Theta(\log_2 n)$

Ω = lower bound
↳ count can't be less than that

O → upper bound
↳ count can't go beyond this

Eg: $C(n) \in \Omega(\log n)$

$C(n)$ cannot be less than $\log n$

Exact count by solving

$$T(n) = T(n/2) + 1 \qquad n \geq 1 \qquad C(1) = 1$$

let $n = 2^k$

$$T(2^k) = T(2^{k-1}) + 1$$

$$= \left(T(2^{k-2}) + 1\right) + 1$$

$$= T(2^{k-i}) + i$$

$2^{k-i} = 1$

$k - i = 0$

$k = i$

$\Rightarrow$

$$= 1 + k$$

$n = 2^k$

$k = \log_2 n$

$$= 1 + \log_2 n$$

$\therefore C_{best}(n) = \Theta(1)$

$C^{\text{successful}}_{worst}(n) = \log_2 n + 1$

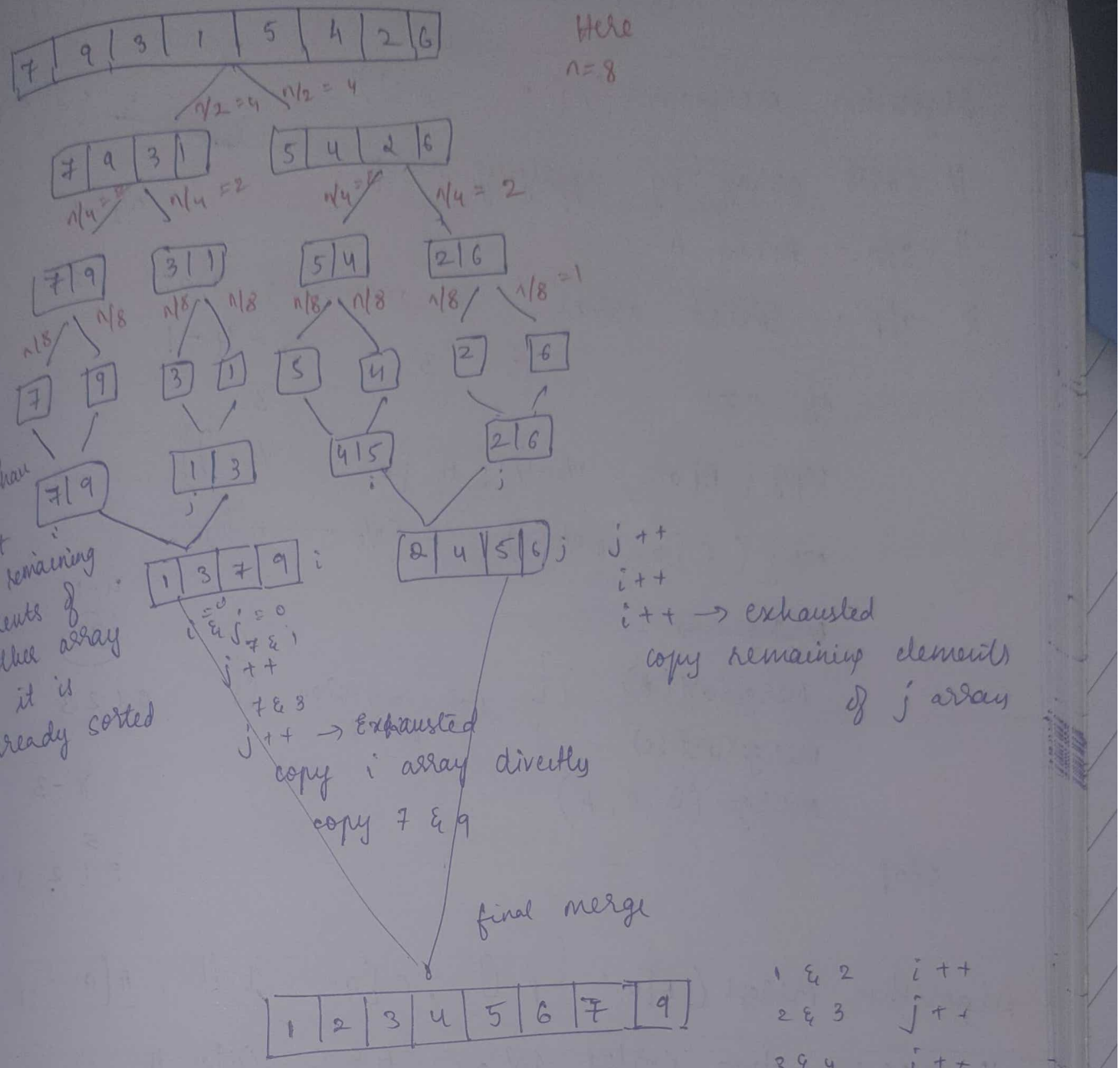
$C^{\text{unsuccessful}}_{worst}(n) = \log_2 n + 1$

→ for both its same only.

$\log_2(n+1)$

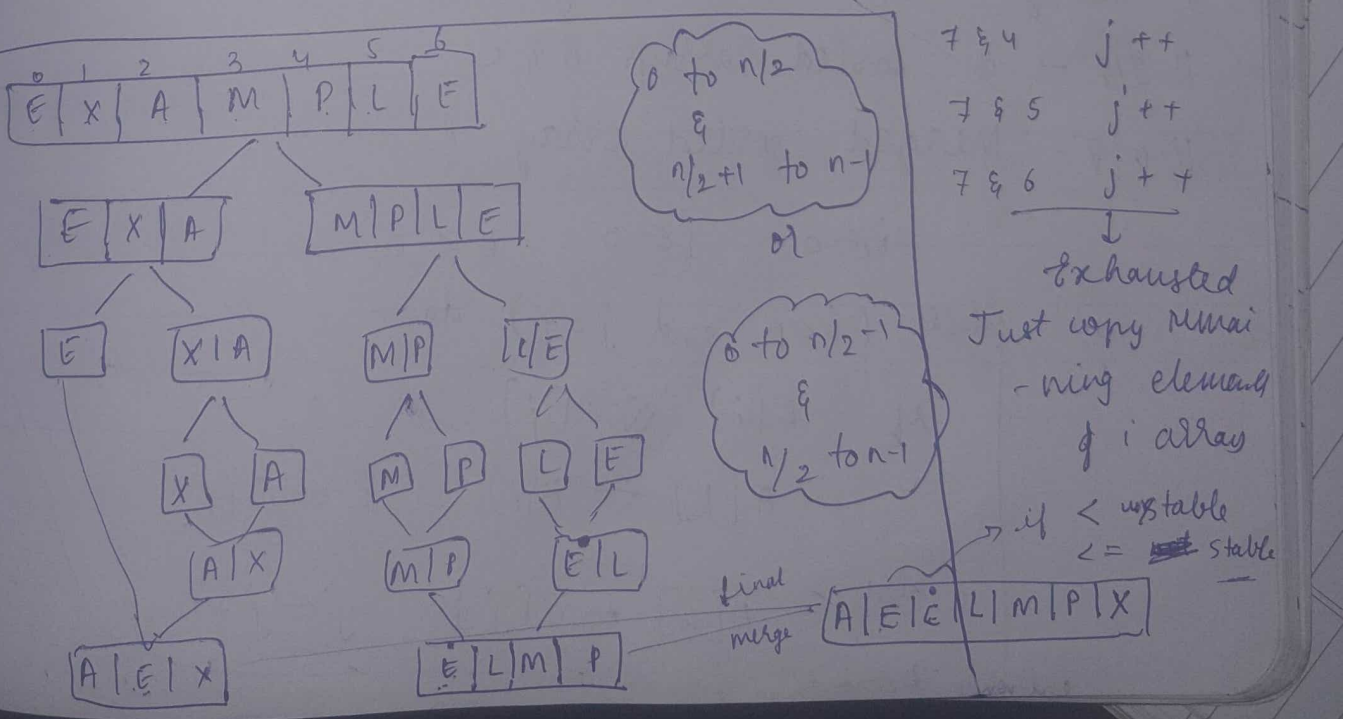
Using Biary search, search is efficient iff array is already sorted. If not sorted then linear search is eff (worst case $\in \Theta(n)$)

Merge Sort



1 & 2	i++
2 & 3	j++
3 & 4	i++
7 & 4	j++
7 & 5	j++
7 & 6	j++

↓
Exhausted
Just copy remaining elements of i array



if $E < \dot{E}$ means condition fails E is not copied → not stable

$E <= \dot{E}$ means condition ✓ E is copied → stable

Algorithm mergesort (A[0 .. n-1])

// Sorts array by applying mergesort

// I/P :- Array A

// O/P :- Sorted array in ascending order

if $n > 1$

copy (B[0 ... $n/2 - 1$] , A [0 ... $n/2 - 1$])

copy (C[0 .. $n - n/2 - 1$] , A [$n/2$ .. $n-1$])

~~Merge So~~

Mergesort (B) } For division of array

Mergesort (C)

Merge (B, C, A)

stop

Algorithm Merge (B[0 ... p-1] ; C[0 ... q-1] , A(0 ... p+q-1])

// Merges two sorted array B & C into A

// I/P → 2 sorted arrays B & C

// O/P :- Merged sorted array A

$i \leftarrow 0$ $j \leftarrow 0$ $k \leftarrow 0$

while ($i < p$ and $j < q$) do

if $B[i] \leq C[j]$

$A[k] \leftarrow B[i]$ $i \leftarrow i+1$

else $A[k] \leftarrow C[j]$ $j \leftarrow j+1$

end while $k \leftarrow k+1$

mergesort (A[7, 9, 3, 1, 5, 4, 2, 6])

B [7 9 3 1]
C [5 4 2 3]
Mergesort(B)

Mergesort (B[7 9 3 1])

B [7 9)
C [3 1]
Mergesort (B)

MS [B [7, 9])

B [7]
C [9]
Mergesort (B)

Mergesort(B[7])
n=1

mergesort(1)

mergesort(C[9])
n=1

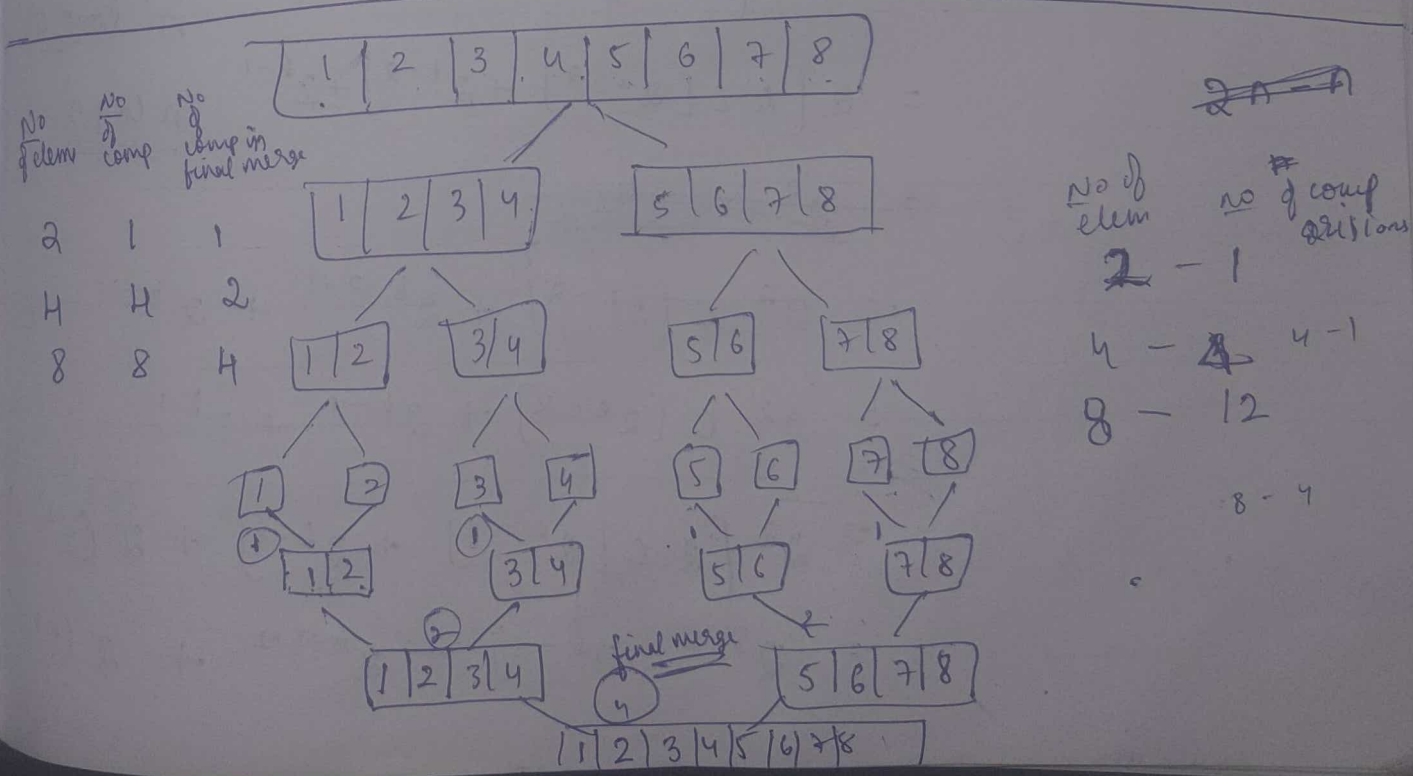
// After one is exhausted to copy other

if (i = p)

copy (A[k ... p+q-1], C [j ... q-1])

else

copy (A[k ... p+q-1] , B [i ... p-1])



No of elem	No of comp	No of comp in final merge
2	1	1
4	4	2
8	8	4

~~2 A = 1~~

No of elem — No of comp divisions

2 - 1

4 - 4 (4-1)

8 - 12 (8-4)

$c(n) = 2c(n/2) + f(n)$ — $n > 1$, $c(1) = 0$

best

$a = b = 2$

Rest merge are added here

$f(n)$ — split + merge

only for final merge

No time to split because we can directly copy it using index

if we try to make it inplace then split requires diff algo & becomes complex

∴ Inplace merge sort is only of theoritical interest

∴ For sorted case

$f(n) = \frac{n}{2}$

$$c(n) = 2c(n/2) + n/2$$

$2^k = n$

$$= 2c\left(\frac{2^k}{2}\right) + \frac{2^k}{2}$$

$$= 2c[2^{k-1}] + 2^{k-1}$$

$$= 2[2c[2^{k-2}] + 2^{k-2}] + 2^{k-1}$$

~~$= 2^2 c[2^{k-2}]$~~

$$= 2^2 c[2^{k-2}] + 2^{k-2+1} + 2^{k-1}$$

$$= 2^2 c[2^{k-2}] + 2^{k-1} + 2^{k-1}$$

$$= 2^2[2c[2^{k-3}] + 2^{k-3}] + 2(2^{k-1})$$

$$= 2^3 c[2^{k-3}] + 2^{k-3+2} + 2(2^{k-1})$$

Master theorem

$a = 2$

$b = 2$

$d = 1$

$a = b^d$

$n^1 \log_2 n$

$n \log n$

$= 2^3 c[2^{k-3}] + 2^{k-1} + 2^{k-1} + 2^{k-1}$

$= 2^3 c[2^{k-3}] \quad 3(2^{k-1})$

$\Rightarrow \boxed{2^i c[2^{k-i}] \quad i(2^{k-1})}$

Here $k - i = 0$

$k = i$

$= 2^k (0) + k 2^{k-1} \qquad c(1) = 0$

$\Rightarrow 2^k k / 2^{k-1}$

$= k \qquad 2^k = n$

$= n \log_2 n \qquad c(1) = 0$

$= n(0) + k 2^{k-1}$

$n(0) + \log_2 n \frac{2^k}{2}$

$= n(0) + \frac{n \log_2 n}{2}$

$= n\left[\frac{1}{2} + \frac{\log n}{2}\right]$

$= \frac{n \log n}{2} \rightarrow$ Exact count

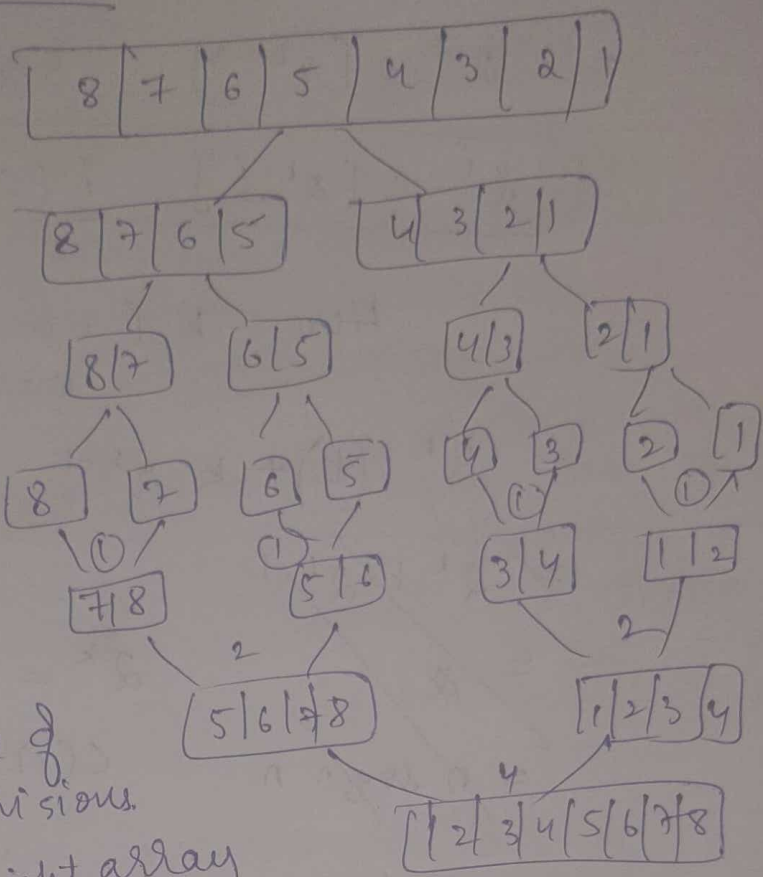
For Descending order also

$= \frac{n \log n}{2}$

$c(n) = 2(c(n/2)) + n/2$

best

Descending order



Same no of comparisions.

Here right array gets exhausted first

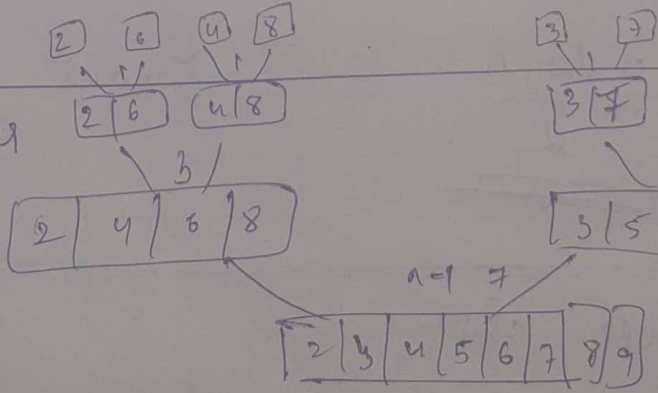
But no of comp is same

Observation :- Smaller element is coming only from one of the 2 subarrays in Both ascending & descending order

This is the Best case i/p.

$n \log_2 n$

Consider



4		
6		comparision
7		
17	2 3	
	3 4	
	4 5	
total comp	5 6	7
	6 7	
	7 8	
7 @ final merge	8 9	

Observation : Smaller elements are coming from alternate array. either of 2 arrays become empty until other array contains just one element.

$$C(n) = 2\,C(n/2) + n-1 \qquad f(n) = n-1$$

$n \log n - n + 1$

$$n = 2^k$$

$$= 2\left(C\left(\frac{2^k}{2}\right)\right) + 2^k - 1$$

$$= 2\left[C(2^{k-1})\right] + 2^k - 1$$

$$= 2\left[2\,C(2^{k-2}) + 2^{k-1} - 1\right] + 2^k - 1$$

$k - i = 0$

$k = i$

$$= 2^2 C(2^{k-2}) + 2\cdot 2^{k-1} - 2 + 2^k - 1$$

$$= 2^2 C(2^{k-2}) + 2^k + 2^k - 1 - 2 - 2^2$$

$$= 2^i C(2^{k-i}) + i\,2^k - \frac{(2^k - 1)}{2}(n-1)$$

$$= 2^k (0) + k\,2^k - \frac{(2^k+1)}{2}(n-1)$$

$$= n \log n - (n-1)$$

$$= n \log n - n + 1$$

→ To generate data 1st half odd, 2nd half even. Then recursively shuffle

---

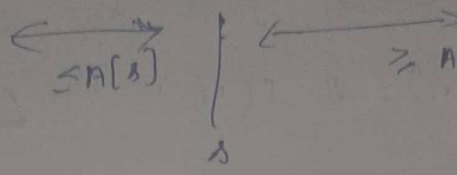
Quick Sort → partitions based on value of element

Merge sort partitions elements based on position

A[0] A[1] ... A[s] ... A[n-1]

Rearranges the elements to obtain the partition position say s.

Such that all elements towards left of split position are $\leq A[s]$ right are $\geq A[s]$ (not sorted)



After obtaining the partition position s
A[s] is in the correct position in sorted array
A[s] is in final position
and left and right are sorted independtly
so no merge is required here

But in merge sort → split and merge
split takes no time beoz by position (index)
sort here

But in Quick sort
split and sort (no merging)
more time is spent here

How to obtain split/partition position 's' ?

→ Pivot element Beoz Guiding role to split array into partitions
choose pivot element. → diff methods

Simple / Basic method → choose 1st element as pivot element.

→ Rearrangement → Diff methods
i.e we shld put pivot element in correct place.

$\leq A[s]$ pivot element $\geq A[s]$
s

This is the partition position / split position

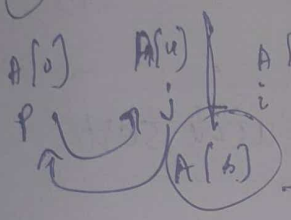
We are efficient method
involves 2 scans — left to right → by index (i)
right to left → (j)

Both scans are compared with pivot element
→ L-R scan should be continued as long as elements are lesser than pivot and it stops on encountering 1st element greater than or equal to pivot
→ R-L vice versa. stops when encounters 1st less/= to elem
→ when both scans stop 3 situations can arise.

A[l] .. A[i] .. A[j] .. A[r-1]

(1) scanning indices have not crossed
⇒ A[i] is greater than pivot } → then swap
⇒ A[j] is less than pivot }
i++ } then resume
j-- } scanning

(2) have crossed → we have got the split position
we get s by interchanging pivot element and A[j]
i.e. j is the split position



→ final position of that element

(3) scanning indices may coincide → then dont swap dont stop just resume scan
⇒ if they are equal to pivot element
can be merged with 2nd condition.

Eg:- 5 3 1 9 8 2 4 7 i = l+1
pivot i j = 4
i → j ← (not crossed)

5 3 1 4 8 2 9 7
i j (not crossed)

5 3 1 4 2 8 9 7
pivot j i (crossed)

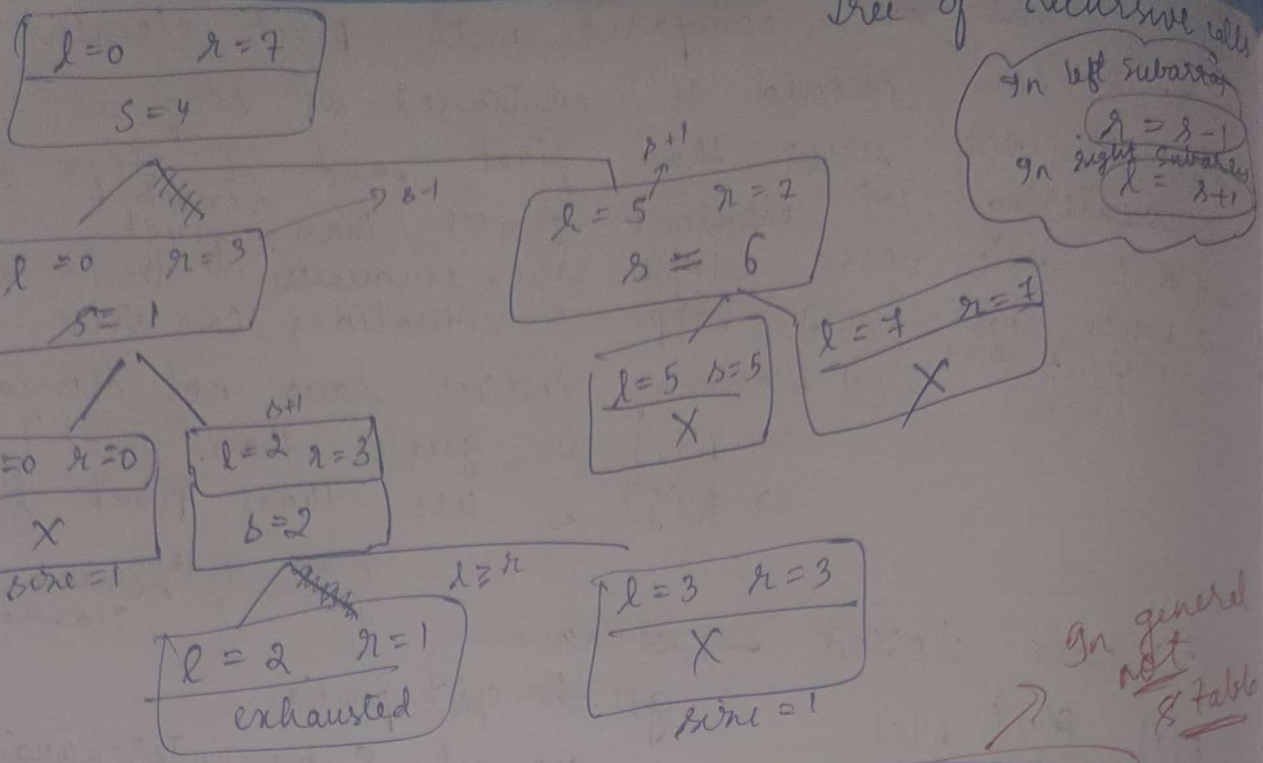
2 3 1 4 5 8 9 7 split position is j
i j pivot i j
not crossed

i ← j
2 3 1 4 8 7 9
j i crossed

1 2 3 4 7 8 9
only one element so no further

i j crossed swap

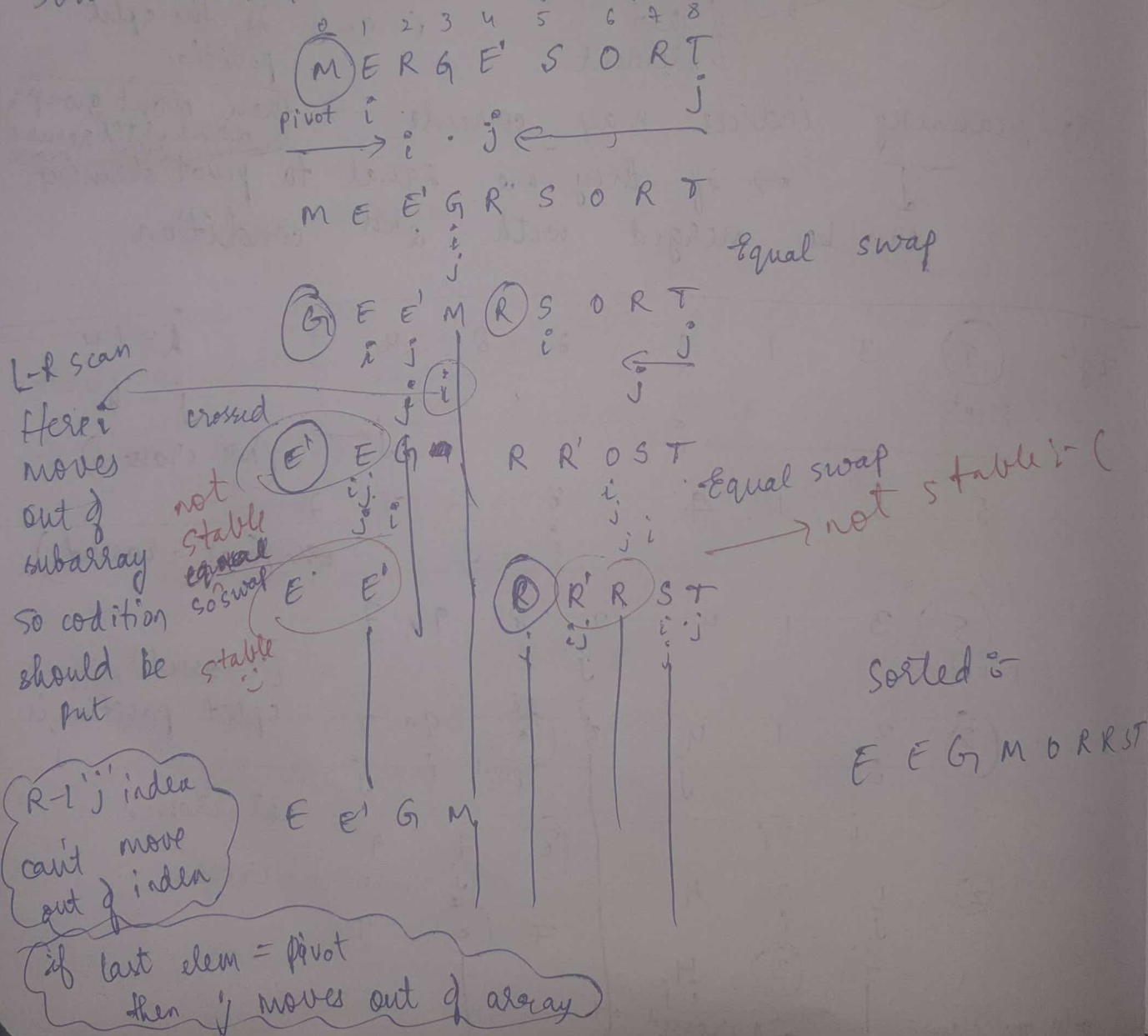
Tree of recursive calls

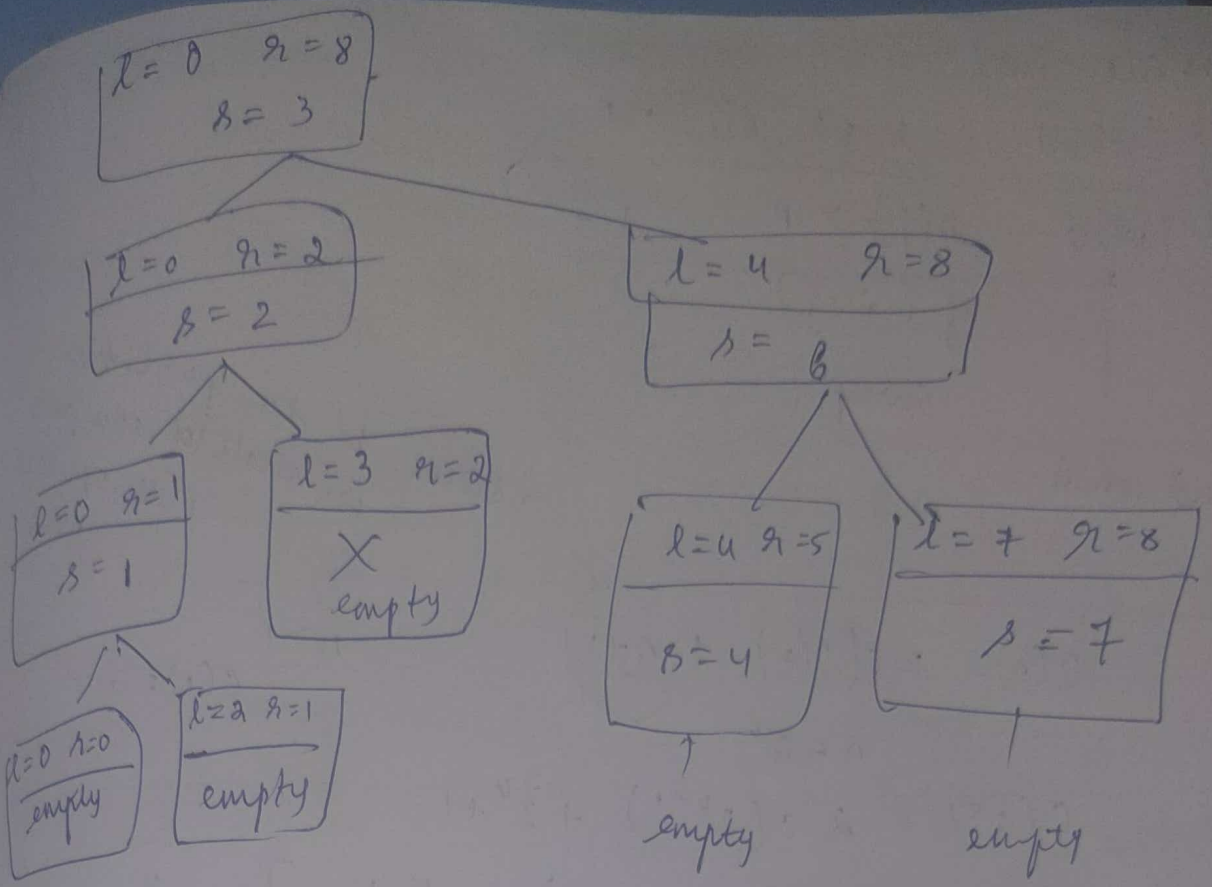


In general not stable

Quick sort

Sort MERGESORT by applying Quick sort

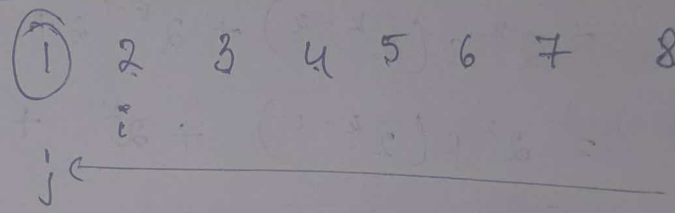




Ascending order

or

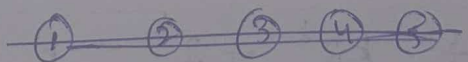
Descending order



size of subarray is just one less than previous array

So worst case.

It's not giving 2 subarrays



1st → 9

2nd → 8

⋮

1

n+1 for 8 elem

45 comparisons

$C(n) = 2C(n/2) + f(n)$

we divide to 2 instances and we are solving for both subarray.

split ✓ merge X

```
while i<j
    while i<=r && A[i]<P      i++
    while A[j]>P              j--
        if (i<j)
            swap(...)
end while
```

↗ below to do partition we should compare so its splitting time

$$C(n) = 2C(n/2) + n+1 \qquad n>1 \qquad C(1)=0$$

$$n = 2^k$$

$$C(2^k) = 2C(2^{k-1}) + 2^k + 1$$

$$= 2\left[2C(2^{k-2}) + 2^{k-1} + 1\right] + 2^k + 1$$

$$= 2^2 C(2^{k-2}) + 2^{k-1}\,2 + 2 + 2^k + 1$$

$$= 2^2 C(2^{k-2}) + 2^k + 2^k + 1 + 2$$

$$= 2^i C(2^{k-i}) + i\,2^k + 2^i - 1$$

$$k - i = 0$$

$$k = i$$

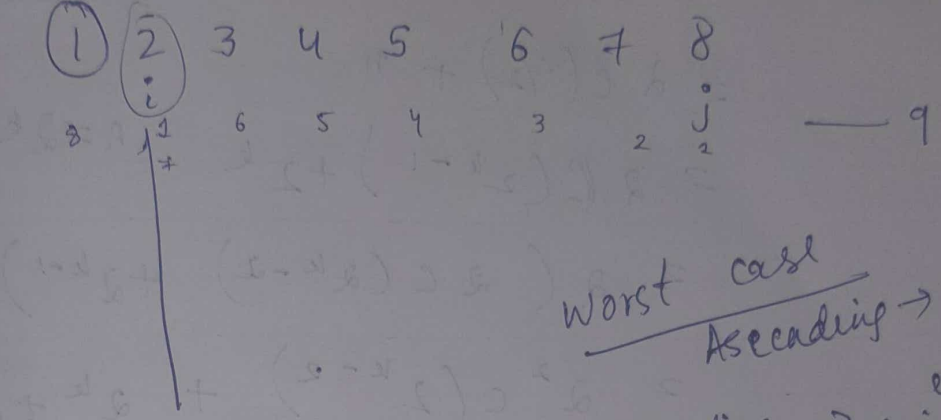
$$= 2^k(0) + k2^k + 2^k - 1$$

$$= n\log n + n - 1$$

$$C_{worst}(n) = (n+1) + n + (n-1) + (n-2) + \cdots + 3$$

$$= \frac{(n+1)(n+2)}{2} - 3$$

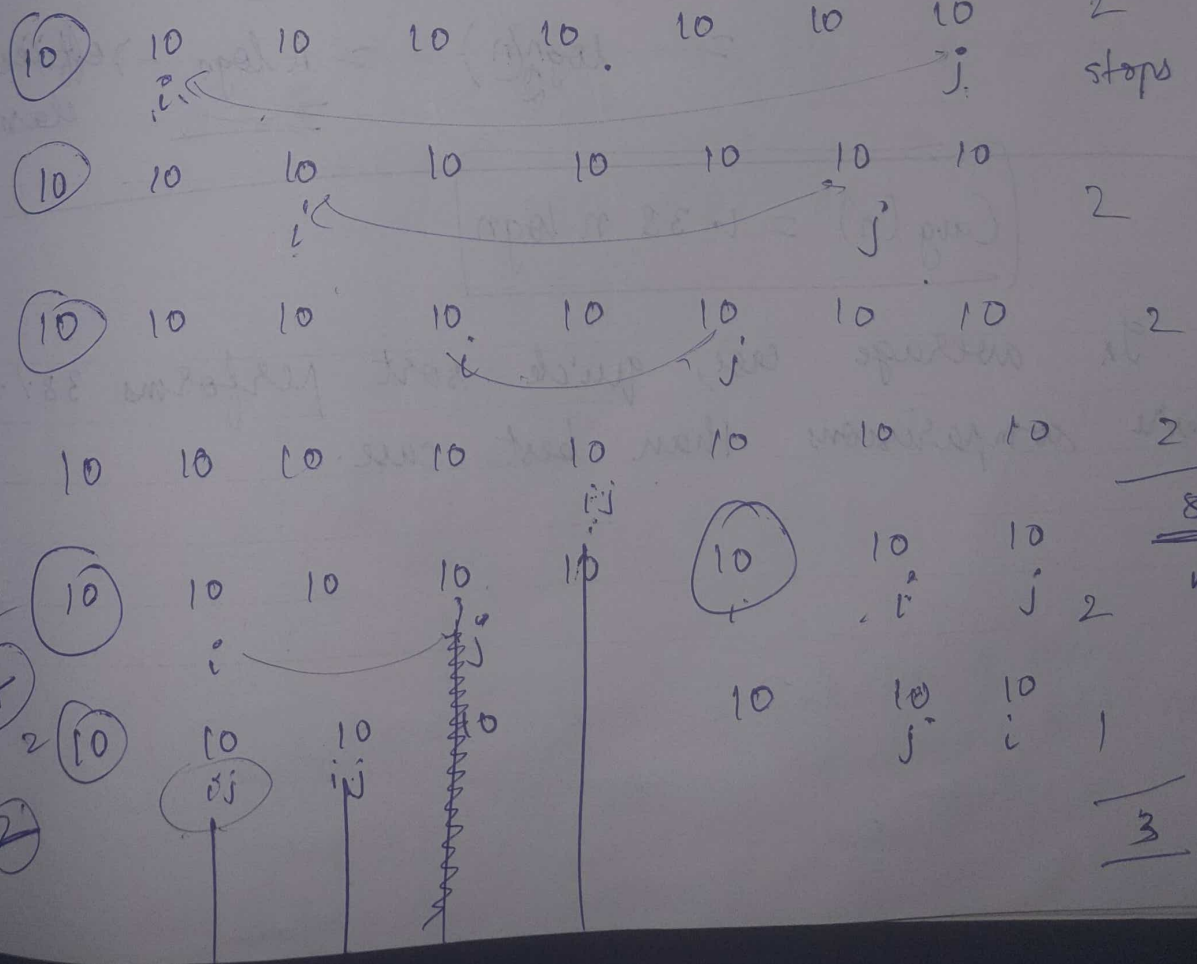
$$\in \Theta(n^2)$$



Worst case
Ascending → s is at start everytime
descending → s is at end everytime

Best → Everytime split is happening at middle of 2 subarrays.
Split always happens at middle

one eg is having all equal elements.



$C_{best}(n) = 2C(n/2) + f(n)$ → only split

$= 2C(n/2) + n$

condition $C(1) = 0$

By master theorem $d = 1$

$b^d = 2 = a$

$\therefore n^d \log n = n \log n$

$= 2C(n/2) + n$

$= 2[C(2^{k-1}) + 2^k$ $\quad n = 2^k$

$= 2(2C(2^{k-2}) + 2^{k-1}) + 2^k$

$= 2^2 C(2^{k-2}) + 2^k + 2^k$

⋮

$k - i = 0$

$k = i$

$= 2^i C(2^{k-i}) + i 2^k$

$= 2^k (0) + k 2^k$

$= \log_2(n) = n \log n$ → efficiency class

$$C_{avg}(n) = 1.38\, n \log n$$

In average case, quick sort performs 38% more comparisons than best case.

Algorithm QuickSort (A[l ... r])

// Sorts subarray A

// Input :- Subarray A with left index l and right index r

// output :- Sorted subarray

```
if (l < r)
    s ← partition (A[l ... r])
    QuickSort (A[l ... s-1])
    QuickSort (A[s+1 ... r])
```

Algorithm partition (A[l ... r])

// Finds the split position in subarray A using 1st element as pivot

// Input :- subarray A[l ... r]

// O/P :- split position

```
p ← A[l]        i ← l+1        j ← r
while i < j do
    while i ≤ r and A[i] < p do
        i ← i+1
    endwhile
    while A[j] > p do
        j ← j-1
    end while
    if i < j
        swap (A[i], A[j])
        i ← i+1
        j ← j+1
endwhile
swap(A[j], A[l])   → can't be p bcoz p changes A[l] not change :-(
return j
```

On Quick sort, the inner loop runs much faster than other sorting techniques for randomly sorted data. (So $n \log n$) → So name is quicksort

So quick sort is used

Another sort having $n \log n$ as eff is Heap sort

Diff versions of Quick sort

→ Pivot position element → median

→ Random pivot choose random index.

↓

we can implement quick sort as randomized sorting algo

Any algorithm involving randomness is called randomized algorithm. (Non deterministic algo)

↓

Ado is like tossing a coin.

The random pivot can generate best case.

↓

if pivot is always A[0] / find then Determin

Comments on Binary search

$$C_{best}(n) = 1$$

$$C_{worst}(n) = 1\,C(n/2) + 1 \qquad n \geq 1,\ C(1) = 1$$

$$= C(2^{k-1}) + 1$$

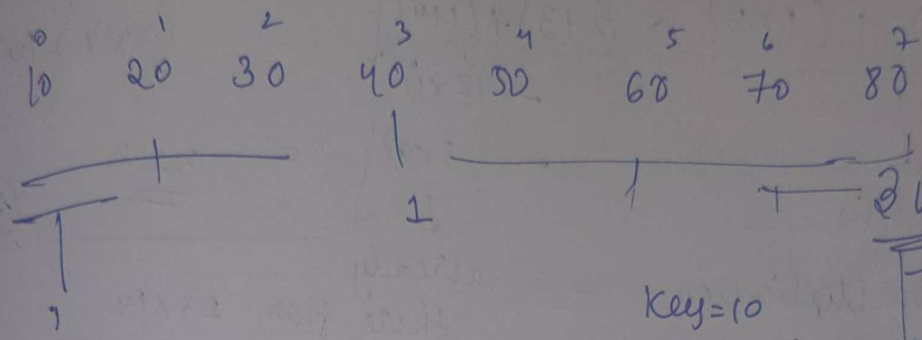
$$= C(2^{k-2}) + 2$$

$$= C(2^{k-i}) + i$$

$C_{worst}(n) = 1 + k$

$= 1 + \log_2^{\uparrow}$ } equal to this

$C(n)_{worst} = \log_2(n+1)$



Based ← { Comp = 3 (one single comparision we get to know if key is in middle or left half or right half)

key = 80

Comp = 4

It fits much better to Decrease by half than divide & conquer

(Atypical Case of Divide & Conquer)

Multiplication of Large integers

To multiply 2 n digit nos we need n^2 mult

$\frac{22 \times 16}{2}$

6×2
6×2
2×1
2×1
} $\rightarrow 4 = (2)^2$

22 & 16 are 2 digit nos

If one digit is smaller we can pad smaller no with zero to equalize no of digits

23 14

$23 = 2\times10^1 + 3\times10^0$

$14 = 1\times10^1 + 4\times10^0$

$23\times14 = (2\times1)10^2 + (2\times4 + 1\times3)10^1 + (3\times4)10^0$

$(2\times4) + (1\times3) = (2+3)*(1+4) - (2\times1) - (3\times4)$

$= 322$

4 multiplications

↓ already there from 23×14

∴ only 1×1

(3) multiplication

generalizing this for any 2 digit nos

$a\times b$

$c = a\times b$

$a = a_1 a_0$

$b = b_1 b_0$

$c = a\times b = c_2 10^2 + c_1 10^1 + c_0$

where

$c_2 = a_1 \times b_1$

$c_0 = a_0 \times b_0$

$c_1 = (a_1+a_0) * (b_1+b_0) - (c_2+c_0)$

$ab - (c_2+c_0)$ → the coeff of 10^1

whole no $-(c_2+c_0)$

No of $+^n$ is ↑ But $\times^n$ is costlier

$\times^n$ is ↓

if not pad with zero

2- n digit nos where n is power of 2

eg $a = 5343$ $b = 9786$

$a = 53\,|\,43$ (a_1 a_0) $= 53\times10^{n/2} + 43$

$b = 97\,|\,86$ (b_1 b_0) $= 97\times10^{n/2} + 86$

if $a = 53\,|\,43$ (a_1 a_0)

a_{11} a_{12}

b_{11} b_{12}

$$a = a_1 \times 10^{n/2} + a_0$$
$$b = b_1 \times 10^{n/2} + b_0$$

$$a \times b = (a_1 \times b_1)10^n + (a \times b_0 + a_0 \times b_1)10^{n/2} + (a_0 \times b_0)$$
$$= C_2 10^n + C_1 10^{n/2} + C_0$$

$a = 2101 \qquad b = 1130$

$21 \mid 01 \times 11 \mid 30 = C_2 10^4 + C_1 10^2 + C_0$

$a_1 \mid a_0 \quad b_1 \mid b_0$

$C_2 = a_1 b_1 = 21 \times 11$

$C_0 = a_0 b_0 = 01 \times 30$

$C_1 = (a_1 + a_0) * (b_1 + b_0) - (C_2 + C_0)$

Product $= (21 + 01) * (11 + 30) - [(21 \times 11) + (01 \times 30)]$

22×41

Now $C_2 = 21 \times 11$

$= (2 \times 1)10^2 + (1 \times 1)10^0 +$

$10^1 [(2+1) * (1+1) - (2 \times 1) + (1 \times 1)]$

$= 200 + 1 + 10(6 - 3)$

$= 231$

$C_0 = 01 \times 30$

$= (0 \times 3)10^2 + (0 \times 1)10^0$

$+ 10^1 [(0+3) * (1+0) - (0 \times 3) + (0 \times 1)]$

$= 0 + 0 + 10(3)$

$= 30$

5×10

$n << 1 + n << 3$

x^n with 10's multiple takes no time (const) Just by left shift

Also for 22 * 41

$= (2\times4)10^2 + (2\times1)10^0 + (2+2) * (4+1) - [(2\times4)+(2\times1)]$

$= (8)10^2 + 2 + [(4\times5) - [8+2]]_{10}$

$= 800 + 2 + 100$ (20 − 10)

$= 902$

Product $= C_2 10^4 + C_1 10^2 + C_0$

Product $= 231 \times 10^4 + (902 - (231+30))\ 10^2 + 30$

Product $= 2374130$

$M(n) = 3M(n/2) + f(n)$ $n \geq 1$

$M(1) = 1$

while splitting no ↑ so $f(n) = 0$ not taken

one digit ^n requires 1 ^n

$n = 2^k$

$= 3M\left(\frac{2^k}{2}\right)$

$= 3^2 M(2^{k-2})$

$= 3^i M(2^{k-i})$

$k - i = 0$

$k = i$

$M(n) = 3^k M(1)$

$= 3^k$

$= 3^{\log_2 n} = n^{\log_2 3} = n^{1.585}$

Strassen's matrix multiplication

$$C = A \times B$$

$$\begin{bmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{bmatrix} = \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} \times \begin{bmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{bmatrix}$$

$$= \begin{bmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 + M_3 - M_2 + M_6 \end{bmatrix}$$

$M_1 = (A_{00} + A_{11}) * (B_{00} + B_{11})$

$M_2 = (A_{10} + A_{11}) * B_{00}$

$M_3 = A_{00} * (B_{01} - B_{11})$

$M_4 = A_{11} * (B_{10} - B_{00})$

$M_5 = (A_{00} + A_{01}) * B_{11}$

$M_6 = (A_{10} - A_{00}) * (B_{00} + B_{01})$

$M_7 = (A_{01} - A_{11}) * (B_{10} + B_{11})$

No. of $x^n = 7$

No. of $+^n / -^n = 18$

$$\left[\begin{array}{cc|cc} 1 & 0 & 2 & 1 \\ 4 & 1 & 1 & 0 \\ \hline 0 & 1 & 3 & 0 \\ 5 & 0 & 2 & 1 \end{array}\right] \times \left[\begin{array}{cc|cc} 0 & 1 & 0 & 1 \\ 2 & 1 & 0 & 4 \\ \hline 2 & 0 & 1 & 1 \\ 1 & 3 & 5 & 0 \end{array}\right] = \begin{bmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{bmatrix}$$

A_{00} A_{01} A_{10} A_{11} B_{00} B_{01} B_{10} B_{11}

$$\begin{bmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 + M_3 - M_2 + M_6 \end{bmatrix}$$

$$M_1 = \begin{bmatrix} 10 \\ 4\ 1 \end{bmatrix} + \begin{bmatrix} 3 & 0 \\ 2 & 1 \end{bmatrix} * \begin{bmatrix} 0 & 1 \\ 2 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 1 \\ 5 & 0 \end{bmatrix}$$

$$(A_{00} + A_{11}) * (B_{00} + B_{11})$$

$$= \begin{bmatrix} A_{00} & A_{01} \\ 4 & 0 \\ 6 & 2 \\ A_{10} & A_{11} \end{bmatrix} * \begin{bmatrix} B_{00} & B_{01} \\ 1 & 2 \\ 7 & 1 \\ B_{10} & B_{11} \end{bmatrix}$$

$$= \begin{bmatrix} M_1 + M_4 - M_5 + M_7 & \\ & \end{bmatrix}$$

$$M_1 = \begin{bmatrix} 4 & 8 \\ 20 & 14 \end{bmatrix}$$

$$M_2 = \begin{bmatrix} 2 & 4 \\ 2 & 8 \end{bmatrix}$$

$$M_3 = \begin{bmatrix} -1 & 0 \\ -9 & 4 \end{bmatrix}$$

we need to do

$M_1 = (4+2) * (1+1)$

$M_1 = 12$

$M_2 = (6+2) * 1$

$M_2 = 8$

$M_3 = 4 * (2-1)$

$= 4$

$M_4 = 2 * 7-1 = 12$

$M_5 = (4+0) * 1 = 4$

$M_6 = (6-4)(1+7) = 6$

$M_7 = -16$

$(-2) * (7+1)$

$= -16$

$$M_4 = \begin{bmatrix} 2 & 3 \\ -2 & -3 \end{bmatrix}$$

$$M_5 = \begin{bmatrix} 8 & 3 \\ 10 & 5 \end{bmatrix}$$

$$M_6 = \begin{bmatrix} 2 & 3 \\ -2 & -3 \end{bmatrix}$$

$$M_7 = \begin{bmatrix} 3 & 2 \\ -9 & -4 \end{bmatrix}$$

$C_{00} = M_1 + M_4 - M_5 + M_7$

$= \begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \end{bmatrix}$

Why

C_{01}

C_{10}

C_{11}

Divide and conquer is ideally suitable for ~~divide and conq~~ parallel processing

4 parallel process each to calculate C_{00} C_{01} C_{10} C_{11} respectively

---

4 × 4 is divided into 4 ~~each~~ 2 × 2
8 × 8 to 4 × 4 then each 4 × 2 to 2 × 2

$M(n) = 7M(n/2)$

For each matrix we need 7×7 → i.e in $M_1 M_2 \cdots M_7$

We divide array to half each time

$18 + 0$ → no time to split or merge just substitute

$M(n) = 7M(n/2) + 0 \quad n > 1$

$M(2^k) = 7M(2^{k-1}) \quad M(1) = 1$

$n > 1$

where n is a multiple of 2 if not pad with zeroes

Eg: $\begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}$

So pad with k^2 of 0 $\begin{bmatrix} 0 & \cdot & \cdot & \cdot \\ 0 & \cdot & \cdot & \cdot \\ 0 & 0 & 0 & 0 \end{bmatrix}$ to make

$= 7 M(2^{k-2})$

$= 7^i M(2^{k-i})$

$k - i = 0$

$k = i$

$= 7^k M(1)$

$= 7^k$

$= 7^{\log_2 7}$

$= n^{\log_2 7}$

$M(n) = n^{2.807}$

a^n by Divide & conquer

$a^{n/2} + a^{n/2}$

Based on same strategy the max decrease in expon is $n^{2.376}$

Reduced only by small fraction from n^3 But still its efficient than Brute force as n tends to ∞

But Brute force is suitable for working smaller matrices.

The decrease in exponent is obtained at cost of increasing complexity

So for small values, Brute force is ✓

Lower bound :- Minimum amount of work required by any algo to solve the problem

Note: We dont have any algorithm in sorting for T^{er} efficiency than nlogn

i.e Basic op cant go less than nlogn

Lower bound for sort → $n \log n$

matrix x^n → ~~n^3~~ n^2

tight

we can't directly ↑ eff to n

We can just improve bh n and $n \log n$

large gap b/n lower bound and optimality ∈ { too tight i.e. diff to go near n^2

i.e. n^2 $n^{2.376}$

As it is too tight there is scope only for fractional improvement

i.e. there is scope for improvement.

we can't directly make it n^2 only fracti

---

from $M_1 \ldots M_7$

$7 * ^n$

$18 + ^n$

$n = 4$

$\frac{4}{2} = 2^2 = 4$

$A(n) = 7\,M(n/2) + 18\,(n/2)^2$

$n > 1$

$A(1) = 0$

$A(n) = 7\,M(n/2) + \frac{18}{4} n^2$

$f(n) = \frac{18}{4} n^2$

$\in \Theta\, n^2$

$d = 2$

$a = 7$

$b = 2$

$b^d = 4$

$a > b^d$

$A(n) = n^{\log_2 7}$

$A(n) \in n^{2.807}$

$$C_{00} = \begin{bmatrix} 5 & 4 \\ 4 & 5 \end{bmatrix} \quad 18 + ^n$$

$$C_{01} = \begin{bmatrix} 7 & 3 \\ 4 & 9 \end{bmatrix} \quad 18 + ^n$$

$$C_{10} = \begin{bmatrix} 8 & 1 \\ 5 & 8 \end{bmatrix} \quad 18 + ^n$$

$$C_{11} = \begin{bmatrix} 3 & 7 \\ 7 & 1 \end{bmatrix} \quad 18 + ^n$$

Growth of $+^n$ is same as that of x^n

if we solve the recurrance then

$$A(n) = 6\left(7^{\log_2 n} - \underbrace{4^{\log_2 n}}_{\text{neglected}}\right)$$

$$= 6 \cdot 7^{\log_2 n}$$

$$\in O(n^{\log_2 7})$$

Decrease and Conquer

Exploiting solution to given instance.
and solution to smaller instance of same problem.

And establishing relationship b/n them

Eg: $a^n = a \cdot a^{n-1}$

Then problem is solved either by iteration or recursion

same no Basic op count as Brute force

Same as Brute force But approach is diff

3 variations in Decrease & Conquer

→ A) Decrease by a constant

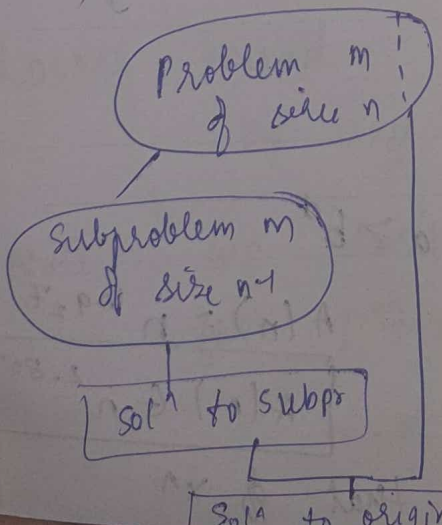
→ B) Decrease by a constant factor

→ C) variable size decrease.

Typical constant value is 1
i.e. Decrease by 1.

Eg: $a^n = \begin{cases} a^{n-1} * a & n > 1 \\ a & n = 1 \end{cases}$

In every step instance size is reduced by 1



Size of instance is decreased by a constant factor

usually constant factor = 2

Problem of size n

→ Decrease by some factor (2)

Eg:- ① Binary search
Divide to 2 solve only one

Subproblem of size n/2

Sol^n to subproblem

Sol^n to original problem

② $a^n = \begin{cases} (a^{n/2})^2 & \text{if } n \text{ is even +ve int} \\ (a^{(n-1)/2})^2 a & \text{if } n \text{ is odd +ve} \\ a & n=1 \end{cases}$

Eg:- 2^4 $= (2^2)^2$

$= ((2^1)^2)^2$

$= (2 \times 2)^2$ → 1st x^n

$= (4)^2$

$= 4 \times 4$ → 2nd x^n

$= 16$

2 x^n

Here decrease & conquer is efficient

* Same using divide and conquer ←

$a^n = a^{n/2} \cdot a^{n/2}$ until $n=1$

$= 2^2 \quad 2^2$

$= 2(\times 2) \; (\times) \; 2(\times 2)$ → 3 x^n

* 2^6

Divide

$2^3 \times 2^3$

$2^2 \times 2^1 \times 2^2 \times 2^1$

$2 \times 2 \times 2 \times 2 \times 2 \times 2$

$= 5 \; x^n$

Decrease

$(2^3)^2$

$((2^1)^2 \cdot 2)^2$

$((2 \times 2) \times 2)^2$ - 2 x^n

(8×8) - 3 x^n

2^5

$(2^2)^2 \cdot 2$

$((2^1)^2)^2 \cdot 2$

$(2 \times 2)^2 \cdot 2$

$(4 \times 4) \; 2$

16×2

3 x^n

So by using ~~divide~~ decrease and conquer no of x^n growth is logarithmic in constant factor 2 bcoz it is getting reduced by half

c) Variable size decrease

Eg:- Computing GCD using Euclid's algorithm

$GCD(M,N) = GCD(N, M \% N)$

size of second parameter is reduced everytime atleast after 1st iteration (i.e. if $M < N$ ↳ 1st it is swapped then decreased)

It is neither reduced by a constant nor by const factor

---

3 problems
- Sorting
- Graph problems (DFS, BFS)
- Topological sorting problems

---

1) Insertion sort :- Decrease & conquer by 1

Size is n

Decrease the problem to subproblem of size n-1 and assume the left one element to be sorted.

Eg:- 15 | 10 45 19 3 7 25

15 assumed to be sorted

$i=1$ to $n-1$

We can scan in any dir

But we prefer Right to left scan

15 (10) 45 19 3 7 25

$j = i-1$, i, $n-1$

$V = A[i]$

compare $A[i]$ with all elements from R to L

$15 > 10$

15 15 45 19

$j=-1$

come out of loop swap $A[j+1]$ with V

10 15 45 19 3 7 25
j i
stop

10 15 45 19 3 7 25
j i

10 15 45 45
j

10 15 45 45
j

j+1 = v

10 15 19 45 3 7 25
j i=v

10 15 19 45 45
j

expanding →

10 15 19 19 45
10 15 15 19 45
10 10 15 19 45
j = -1
So out of loop
& A[j+1] = v
∴ A[0] = 3

3 10 15 19 45 7 25

3 7 10 15 19 45 25

3 7 10 15 19 25 45

Algorithm Insertion (A[0 ... n-1])
// sorts
// I/P :- Array A
// O/P : A in ascending order

for i ← 1 to n-1 do
    v ← A[i]
    j ← i-1
    while (j ≥ 0) and A[j] > v
        A[j+1] = A[j]
        j ← j-1
    end while
    A[j+1] ← v
end for

To avoid this we ↑ size by 1
Add sentinal element at left extreme

| -999 | 3 | 7 | 10 .. |

for Right left to ~~left right~~ scan

Scan from left to right

10 9 8 7 6 5 4 3 7

j i

V = 1

j=0 always | if j >= 1 stop | A[j] > V stop

~~j<i~~

9 10

A[i-j] ← A[i-j-1]

j → 0 to i-1

i → 1 to n-1

7 5 3 ~~6~~ 6 2

j=0 i

5 7 3 ~~6~~ 6 2

j i

3 5 7 6 2

j i

for j ← 0 to i-1 do

if A[j] > V

shift

Try it

Analysis (R to L)

(Ascending order) → Best case

10 20 30 40 50

j i

stop

j i

stop

j i

stop

j i

stop

inner loop one time for each iteration

outer loop executes for (n-1) from i=1 to n-1

Cworst(n) = n-1

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

Linear

(descending order) ⟶ Worst case

10	9	8	7	6	5	4		
9	10	8	7	6	5	4	1 comp	i.e. 0 to i-1
8	9	10	7	6	5	4	2 comp	

$$C(n) = \sum_{i=1}^{n-1} \sum_{j=0}^{i-1} 1$$

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

Quadratic

Avg case is proved to be $\frac{n^2}{4}$ $C_{avg}(n) = \frac{n^2}{4}$

Insertion sort performs better in avg case than Bubble & selection sort

Any sorting algo except Insertion sort can't be used if sorting data is available at same time. Eg:- Data stream is coming online

i.e. works even if all data is not available ⟶ online algorithm.

1) Straight insertion sort ⟶ The above

2) Binary insertion sort ⟶ Given element to be inserted

Find the position to be added using Binary search ⟶ worst case ⟶ $\log n$

⟶ In ordinary inner loop is working linear times so this uses efficien

3) Shell sort

DFS and BFS

Decrease by 1

DFS:- ★ Start at any vertex mark as visited
★ Choose the unvisited adjacent vertex
★ If 2 / more unvisited adj vertex tie, is broken arbitrarily and based on data structure used for storing

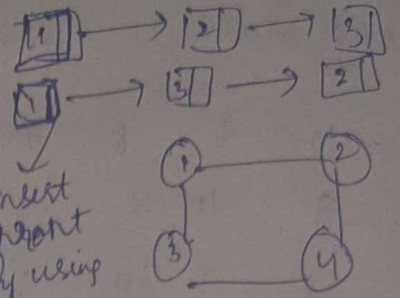
Stack is required

The choice of next unvisited vertex is dictated by choice of ds and operation it performs

★ Stops when dead vertex is reached → vertex with no adjacent unvisited vertex

Once reached the algo backs up one edge from where dead end was reached

★ Stops when starting vertex becomes dead end



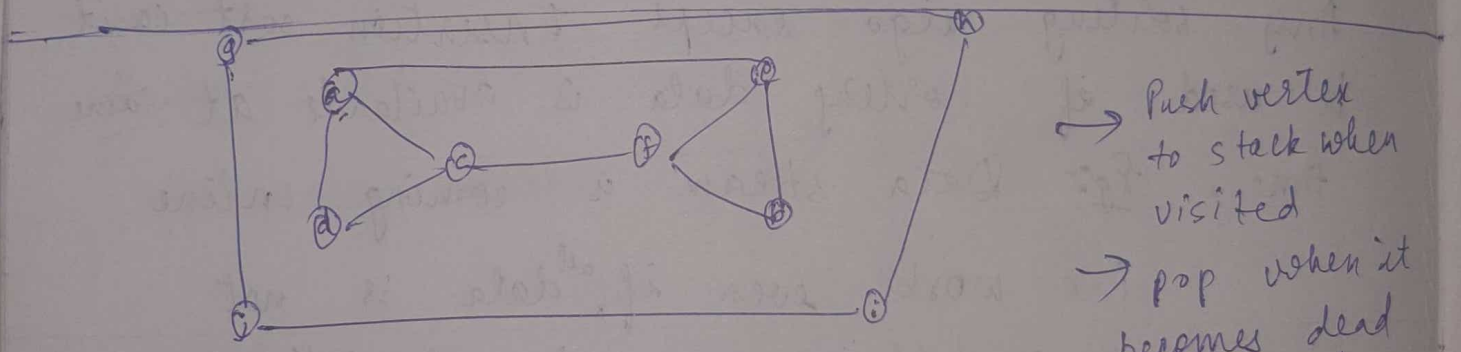
→ if input read by maintaining lp [no time inserted]

insert front by using lp (no time)

	1	2	3	4
1	0	1	1	0
2	1	0	0	1
3	1	0	0	1
4	0	1	1	0

→ Here in chronological order

① ② ④ ③



→ Push vertex to stack when visited

→ pop when it becomes dead end

If array then chronological order

b 6,2 → 2nd dead end

e 5,3

f 4,4

d 3,1 ← dead end

c 2,5

a 1,6

Restarting DFS by choosing other unvisited vertex

j 10,7

i 9,8

h 8,9

g 7,10

Minimum no of edges in a connected graph of n vertices is $n-1$. → linked list

max no of edge $= \frac{n(n-1)}{2}$ → complete graph → Adjacency matrix.

~~Appl^n of DFS~~

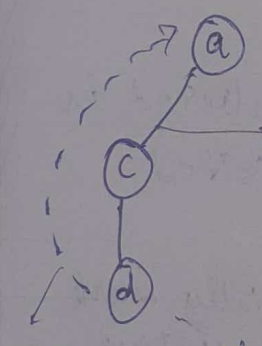
→ If DFS is restarted after choosing 1st vertex then graph is not connected

→ Also we can collect connected components of graph Here its a c d b e f and g h i j

→ It is useful in constructing DFS forest.

forest = collection of trees

Starting vertex chosen to start DFS → it is the → root of the 1st tree of forest

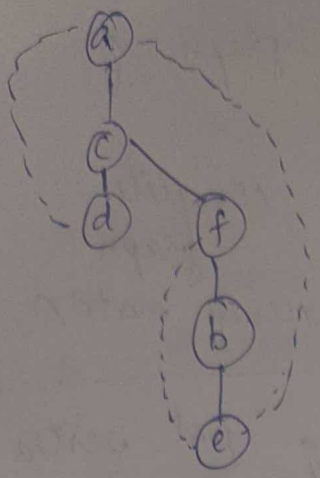


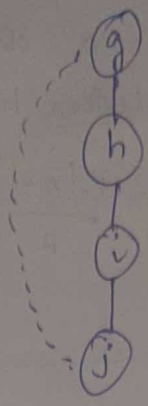
Whenever unvisited adjacent vertex is obtained from root add it as a child of root

An edge connecting (v, w) where v is already visited w is not yet visited

An edge connected such vertices is called Back edge ← (v, w) → both are visited and w is not an immediate predecessor of v

i.e (d, c) → c is immediate predecessor of d





DFS forest → ① If DFS forest has only 1 tree then connected graph.

If has 2 trees → ② Each tree gives set of connected components

③ If a back edge is encountered then graph is not acyclic ⇒ If atleast one backedge then cyclic

---

Algorithm DFS(G)

// visits the vertices in DFS

// I/P :- G = <V, E>

// O/P :- Graph G with its vertices numbered in the order in which they are visited

Mark each vertex with 0 indicating unvisited

count ← 0 // → global variable

for each vertex v in V do
(i.e. 0 to n-1)

| if v is marked 0
| dfs(v)

end for

stop

Initially mark all vertices as unvisited

v[0] = 0
v[1] = 0
.
.
v[n-1] = 0

↑ vertex

Sorting

Alg	efficiency class	C(n)	Stable	Inplace	Swap
Selection sort	Quadratic	$\frac{n(n-1)}{2}$	no	Yes	linear
Bubble sort (Base version)	Quadratic	$\frac{n(n-1)}{2}$	Yes	yes	Quadratic
Better version					

Algorithm dfs(v)

// Recursively visits the unvisited adjacent vertices of v and marks v with the global counter count.

v[0] = 1
: =0
; =0
v[n] = 0

count ← count + 1
mark v with count
// print v
for each vertex w in V adjacent to v
    if w is marked 0
        dfs(w)
end for

Basic operation : check for adjacency (comparision)

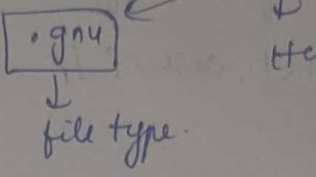
For each vertex, it is compared v times

∴ v.v

$|V^2|$

Linear search Analysis

Design menu for diff cases
Record Basic operation count in diff files < Best.txt, Worst.txt
Plot graph — By loading file



Steps :- for Best case

① → Create 2 files < LSdata.txt, Best.txt

② → ~~Create~~ using for loop, initially ~~set~~ make array Arr
of 10 elements
20 elements
:
100 elements

③ Set elements of array using random no generator

④ Set key = 1st elem of arr
(∵ Best case is 1st elem = key)

⑤ Now start searching Arr and keep count of no of iter i.e. count of Basic op

⑥ Now update LS_data.txt with
i) iteration no of for loop
ii) no of element in arr in that iteration
iii) All elements of arr in that iteration
iv) key in that iteration

⑦ Now update Best.txt with No of elements | No of iteration i.e. count of Basic op

⑧ Then create .gnu file to Plot data in Best.txt

Now Worst case → element is not there only i.e. key = 12345
Avg case → element is somewhere in middle key = arr[n/2]

GCD algorithm analysis

Design menu driven program to find GCD of 2 nos

Euclid's consecutive int check

Record basic op count for different values of m & n

Identify best case & worst case i/p for problem

Modified Euclid's algo.

max op^n count i.e max ÷ (worst

1 to 10 < max / min ÷ (best)

20 to 30 < max / min

.

.

.

190 to 200

Euclata

cbest

eworst

mc