Module - 1

Algorithm - definition.

Step by step procedure/instruction to solve problem domain

* Prblm domain

Algorithm	Program
• at the time of design	• Coorks at implementation stage

Algorithm is a step by step finite sequence of instructions to solve well defined computational problems.

① Difference b/w algorithm & program

Algorithm	program
• written at design time	• Program written at implementation
• designer should aware about prblm domain	• Programr should aware of prblm domain
• Language independent	• Language dependent such as C, C++, Java, Python etc
• Independent of h/w & s/w	• dependent on h/w & s/w such as windows ubuntu hard disk ram etc

Characteristics

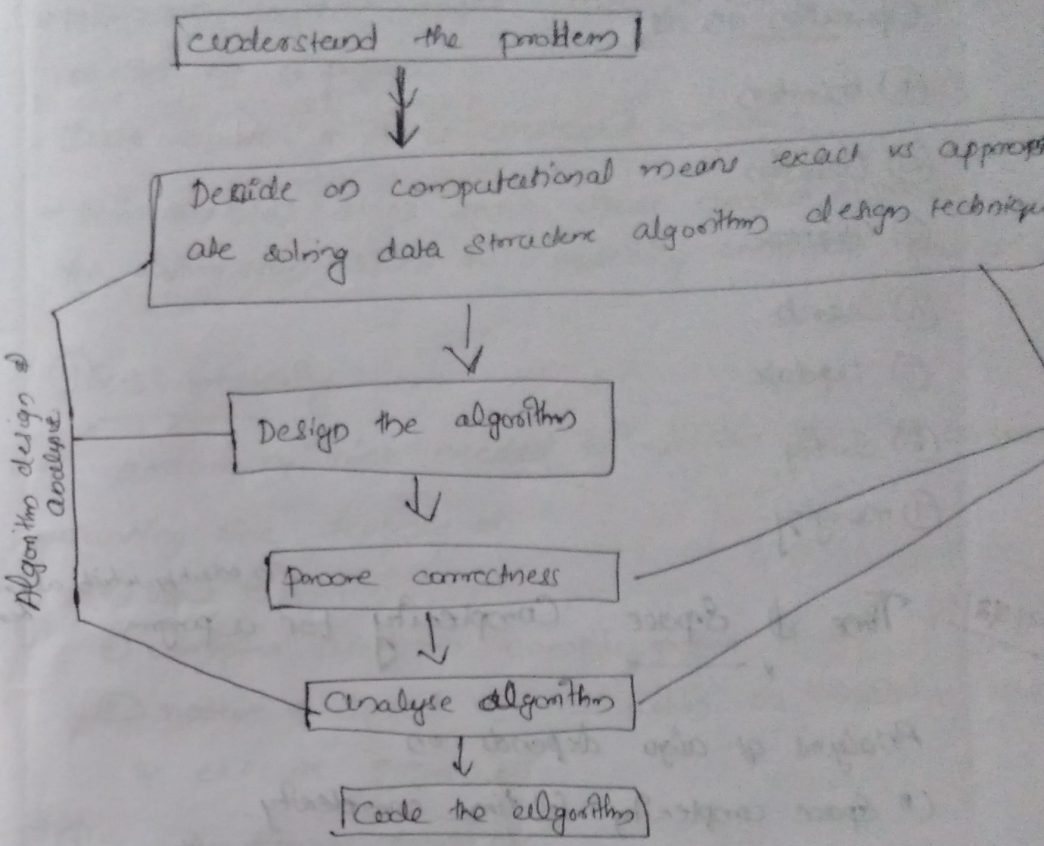
- Input – 0 or more value
- Output – atleast one value
- definiteness – each instructions are clear & unambiguous
- finiteness – steps are finite
- effectiveness – Do something, avoid unnecessary steps.

- Study of an algorithm involves 3 steps.

① algorithm design
② Proving the correctness of algorithm.
③ analysing the algorithm.

Data Structure – The way of organizing, managing, storing datas is called

- Primitive
  - int
  - char.
  - float
- non primitive
  - linear
    - linked list
    - Stack
      - queue
    - Array
  - non linear
    - graph
    - tree



Abstract data type

Some data structure resides inside other data structure and hides the structure just like

Stack & queue – use array / linked list

Operation on ds:

1. Insertion
2. Deletion
3. Traverse
4. Search
5. Update
6. Sorting
7. merging

19/1/21. Time & Space Complexity for a program

To calculate which algo is best

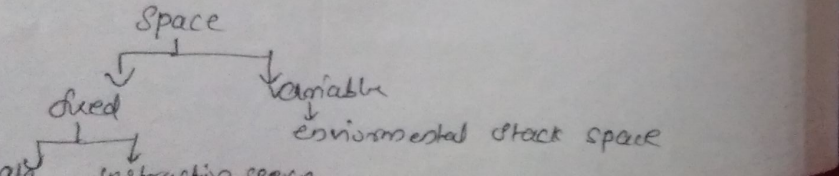
Analysis of algo depends on
(1) Space complexity (2) time complexity

Efficient algo can be computed in the min. requirement of time & space.

Space complexity

The amount of memory needs to run an algo for completion.

&



* Instruction space - Space needed to store executable version of a program.
* Data space - It stores constant & variables
* Environmental stack space - Space needed to store the information about the partially completed process

Time complexity

* The amount of time needed to run an algo for complete
* running time depends on
  (*) Input to prgm
  (*) compiler used to compile prgm
  (*) nature & speed of instruction in machine used to execute prgm.

Time & Space trade off

- It is not possible to achieve these 2 objectives such as time & Space simultaneously.
- Some algo may requeire more space, but less time to complete its execution. while other algo requeire less space, but takes more time to complete execution. So we need some sacrifice. If space is our constrain, then choose an algo/ prgm with les

Space at cost of time

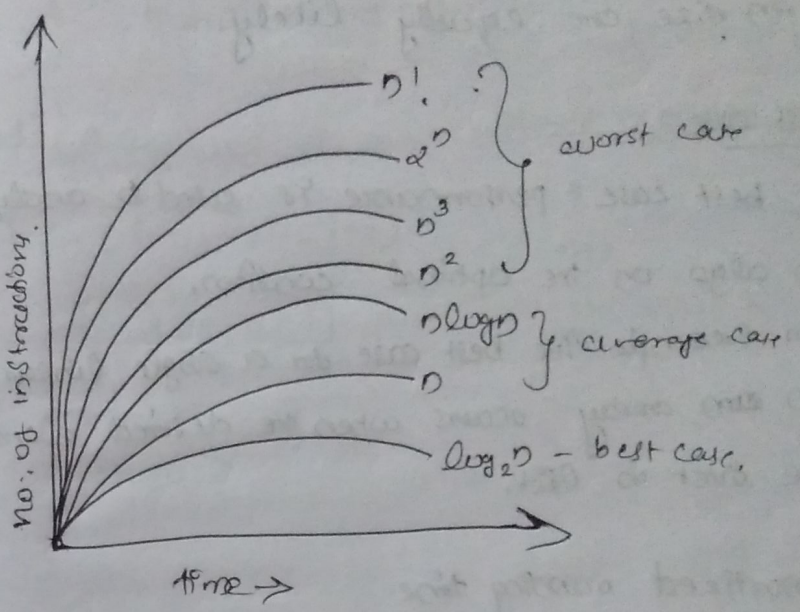
our constrain, choose a pgrm having less time to execute at the cost of more space.

Asymptotic notation

* Notation that enable us to make meaning full statements about time & space complexity of a pgrm. This notation is called asymptotic notation
* It describes the behaviour of time & space complexity of large instance characteristics. (large numbers)
* Common asymptotic fns.

function	name
* 1	constant
* n	linear
* log n	logarithmic fn
* n log n	"
* n^2	quadratic
* n^3	cubic
* 2^n	exponential.
* n!	factorial.

Growth rate



Worst case, avg case, best case & amortized time complexity

Worst case

- This denotes the behaviour of an algo w.r.t the worst possible case of input instance
- The worst case running time of an algo is an upper bound on the running time for any i/p

Average case

- The avg case running time of an algo is an

estimate of the running time of an avg i/p

- Avg case running time assumes that all i/ps of given size are equally likely.

Best case

- The best case performance is used to analyse an algo on the optimal condition.

for example: The best case for a single linear search on an array occurs when the desired element is the first in list.

Amortized running time

- It refers to the time required to perform a sequence of related operation averaged over all the operations performed. It
- It guarantees the avg performance of each operation in the worst case.

Analysing time complexity to basic statements:

```
Add()                          no: of execution
{   a=10;                      (1)
    b=20;                      (1)
    c=a+b;                     (1)
    Print c                    (1)
}   Total T = O(1).
```

```
Add(a,n)                       no: of execution.
{   S=0;                       1
    for(i=1; i<=n; i++)        n+1
    {  S=S+A[i]                n
    }
    return S                   1
}
```

Total = $2n+4 = n$

time = $O(n)$.

We have 2 types of algorithm.

① iterative ② Recursive

→ If our algo contains loops such as for, while etc then such algo are called iterative eg: Add(a,n)

→ If a fn calls by fn contains recursive calls. then it is called recursive algo.

- This method is called frequency count method.
- Another way of finding time complexity of n is polynomial form using 4 rules:

1) Drop low order terms because it is not significant

2) Drop constant multipliers.

eg: $2n^3 + 3n^2 + 4n + 6$.

$\Rightarrow 2n^3$

dropping coefficient

$\Rightarrow n^3$

$\therefore$ Time complexity = $O(n^3)$

eg: $f(n) = 4n^4 + 3n^2 + 4n + 8$

$\Rightarrow O(n^4)$

$f(n) = 16n + \log_2 n$

$\Rightarrow O(n)$.

Time complexity analysis for loop

	no: of execution
Add (a, b, r_1, r_2, c_1, c_2)	1
{ for (i=0, i<r_1; i++)	n+1
{ for (j=0; j<c_1; j++)	n^2+1
{ c[i][j] = a[i][j] + b[i][j]	n^2
}	
}	
return c	1
}	

Total = $2n^2 + n + 4$ ⇐ 2nd method

$= n^2$

$\Rightarrow O(n^2)$

	no: of execution
Multiply (a, b, r_1, r_2, c_1, c_2)	1
{ for (i=0, i<r_2; i++)	n+1
{ for (j=0, j<c_1; j++)	n^2+1
{ for (k=0; k<c_1; k++)	n^3+1
{ c[i][j] = a[i][k] + b[k][j];	n^3
}	
}	
return c	1
}	

$2n^3 + n^2 + n + 5$

$= n^3$

$\Rightarrow O(n^3)$.

Recursive algorithm:-

```
                                    no: of algo.
fact(n)                                 1
{ fact = 1                              1
  if (i==0)||(i==1)                     1
    fact = n x fact(n-1);               n
    return fact;                        1
                                       n+5
```

$\Rightarrow O(n)$

23/11/21

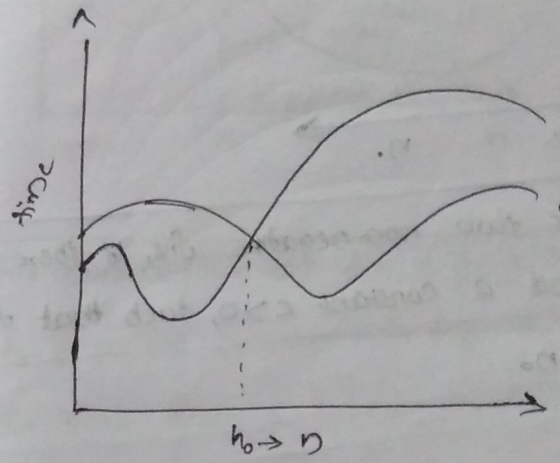
Asymptotic notation

Any $f(n)$ can be represented in 3 ways, such as

1) Bigoh (O),
2) Omega (Ω)
3) theta (Θ)

Bigoh is used for expressing the upper bound of an algorithm's running time.

- It is the measure of the longest amount of time possibly taken for algorithm to complete.
- It represents the worst case of an algorithm.



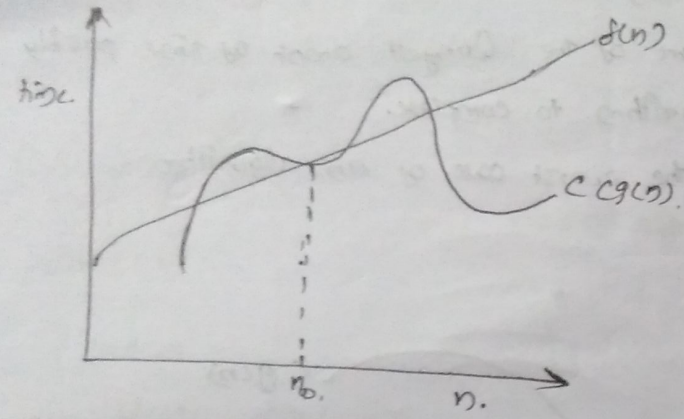
$f(n)$ and $g(n)$ are non-negative fns, if there exist an integer n_0 and a constant $c > 0$, such that for every integer $n \geq n_0$,

$f(n) \leq c.g(n)$.

This is denoted as $f(n) = O(g(n))$

* Omega notation (Ω)

- It is used for expressing the lower bound of an algorithm or it is used for representing the best case of an algorithm.



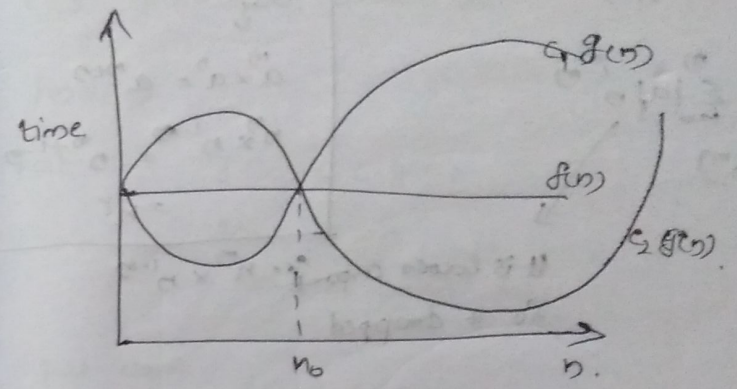
f(n) and g(n) are two non negative fns, if there exist an integer n_0 and a constant $c>0$, such that for every integer $n>n_0$

$f(n) \geq c.g(n)$.

This is denoted as $f(n) = \Omega(g(n))$

Theta notation (Θ)

- If the fn is in blw lower & upper bound then the fn is denoted by Θ notation
- It also represent avg case of an algorithm.



For non-negative fn f(n) & g(n), if there exist an integer n_0 & positive constant c_1 & c_2 $c_1>0$ & $c_2>0$ such that for every integers $n>n_0$

$c_1 g(n) \geq f(n) \geq c_2 g(n)$

This is denoted as $f(n) = \Theta(g(n))$

Theorem

$f(n) = \Theta(g(n))$

(Q) If $f(n) = a_m n^m + \dots + a_1 n + a_0$ and $a_m > 0$.

then $f(n) = \Theta(n^m)$

(A) Proof

f(n) is a fn or statement

$a_m > 0$ means value of $n > 0$

$f(n) \le \sum_{i=0}^{m} |a_i| n^i$

$\le n^m \sum_{i=0}^{m} |a_i| n^{i-m}$

$\le O(n^m)$

$n^i \downarrow$

$a^m \times a^n = a^{m+n}$

$n^m \times n^{i-m} = n^{m+i-m}$

$= n^i$

It is lower compared to $n^m \times n^{i-m}$

So it dropped

Asymptotic notation big O problems

(a) $1 < \log n < \sqrt{n} < n < n \log n < n^2 < n^3 < 2^n < 3^n < \dots < n^n$ ⇒ growth rate

- lower bound (Ω)
- average (Θ)
- upper bound (O)

(Q) $f(n) = 3n+2$, find $f(n) = O(n)$

(A) Assume $g(n) = n$

$f(n) \le c \cdot g(n)$

Put $c=1$, $f(n) \le c \cdot g(n)$

$3n+2 \le n$, condition fails.

$c=2$, $f(n) \le c \cdot g(n)$, $3n+2 \le 2n$, condition fails.

$c=3$, $3n+2 \le 3n$, condition fails.

Find $c(n)$

Put $n=1$ $\quad f(n) \le c \cdot g(n)$

$3n+2 \le 4n$

$5 \le 4$; cond^n fails.

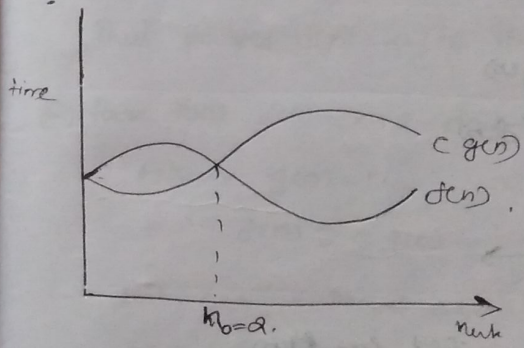
Put $n=2$ $\quad 3n+2 \le 4n$

$8 \le 8$, cond^n true $\therefore n=2$

Put $n=3$ $\quad 3n+2 \le 4n$

$11 \le 12$ cond^n true

$f(n) = O(n) \quad \forall\, c \ge 4$ & $n \ge 2$.



(Q) Given that $f(n) = 4n^3 + 2n + 3$., find $f(n) = O(n)$.

Assume $g(n) = n^3$

$f(n) \le c \cdot g(n)$

Put $c=1$

$4n^3+2n+3 \leq n^3$, condition fails.

Put $c=2$, $4n^3+2n+3 \leq 2n^3$, condition fails.

Put $c=3$, $4n^3+2n+3 \leq 3n^3$, condition fails

Put $c=4$, $4n^3+2n+3 \leq 4n^3$ condition fails

Put $c=5$, $4n^3+2n+3 \leq 5n^3$ condition satisfies

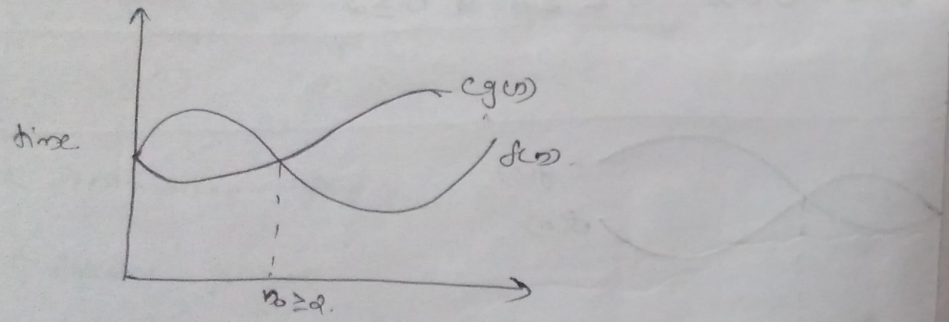
Put $n^3=1$

$4+2+3 \leq 5$

$n=2$ $\quad 4\times 2^3+2\times 2+3 \leq 5\times 2^3$

$39 \leq 40$

$\therefore n \geq 2$.



(Q) Prove that $f(n)=2n^2+n+3$, find $f(n)=\Omega(n)$

(A) Assume $g(n)=n^2$

$f(n) \geq c\,g(n)$

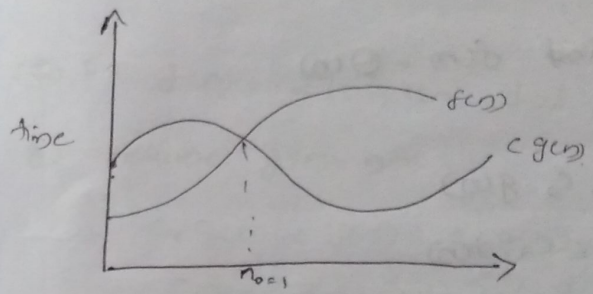
Put $c=1$

$2n^2+n+3 \geq n^2$, condition satisfies

Put $n=1$

$2+1+3 \geq 1$, condition true.

$\therefore n=1$



Thus proved $f(n)=\Omega(n)$ for $c=1$, $n_0 \geq 1$

(Q) Prove that $f(n)=3n+2$ and $\Omega(n)$.

Assume $g(n)=n$

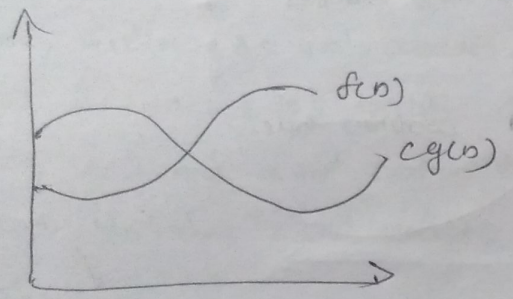
$f(n) \geq c\,g(n)$

Put $c=1$, $3n+2 \geq n$ condition true

put $n=1$, $3+2 \geq 1$ condition true.

$\therefore c=1$, $n_0=1$

Thus proved $f(n) = \Omega(n)$ for $n_0 \geq 1$ & $c \geq 1$



5) If $f(n) = 3n+2$ find $f(n) = \Theta(n)$

(A) Assume, $g(n) = n$

$$c_1 g(n) \leq f(n) \leq c_2 \cdot g(n)$$

$$c_1 g(n) \leq 3n+2 \leq c_2 \cdot g(n)$$

On putting values $1, 2, \ldots$, for both c_1 & c_2 we get

$$2n \leq 3n+2 \leq 4n$$

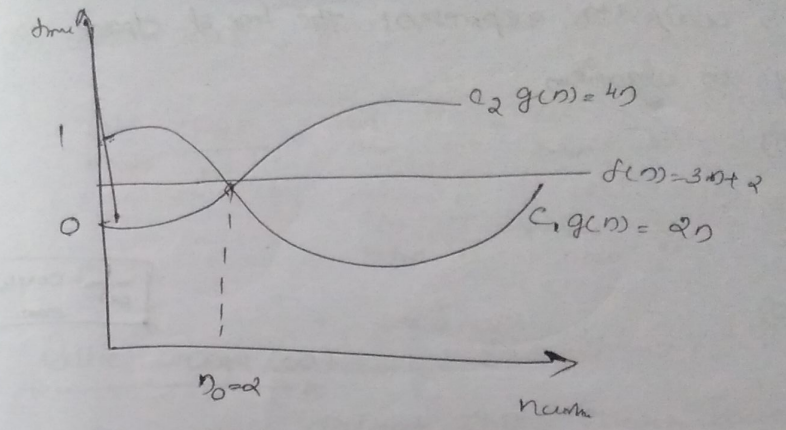
put $n=0$, $0 \leq 2 \leq 0$ false

put $n=1$ $2 \leq 5 \leq 4$, cond^n false

put $n=2$ $4 \leq 8 \leq 8$, cond^n true

Thus proved

$f(n) = \Theta(n)$ $\forall$ $c_1 = 2$, $c_2 = 4$ & $n_0 \geq 2$



(Q) PT $f(n) = 27n^2 + 16n$, find Θ

(A) Assume $g(n) = n^2$

$$c_1 g(n) \leq f(n) \leq c_2 \cdot g(n)$$

$$c_1 g(n) \leq 27n^2 + 16n \leq c_2 g(n)$$

$$26n^2 \leq 27n^2 + 16n \leq 28n^2$$

$\therefore c_1 = 26$, $c_2 = 28$

put $n=0$, $0 \leq 0 \leq 0$. cond^n true

Thus $f(n) = \Theta(n)$ $\forall$ $c_1 = 26$, $c_2 = 28$, $n_0 \geq 0$

There are 5 ways to represent the time & space complexity of an algorithm

(1) Big O (O)
(2) Big Ω
(3) theta
(4) Little o (o)
(5) Little ω.

$\frac{1}{\infty}$ = constant zero.

Little O (o)

- To find loose upper bound

Theorem to find out Little o.

If $f(n)$ & $g(n)$ be a fns such that

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1$ exist, then $f(n) = o(g(n))$ or $f(n) \in o(g(n))$

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1$ or any constant

ex: $f(n) = n^2$, $g(n) = n^3$. Check whether $f(n) \in o(g(n))$

(A) $\lim_{n\to\infty} \frac{n^2}{n^3} = \lim_{n\to\infty} \frac{1}{n} = \frac{1}{\infty} = 0$ / constant

Thus proved that $f(n) \in o(g(n))$

(Q) $f(n) = 7n+8$, $g(n) = n^2$, find & check whether $f(n) \in o(g(n))$

$$\lim_{n\to\infty} \frac{7n+8}{n^2} = \lim_{n\to\infty} \frac{7n}{n^2} + \lim_{n\to\infty} \frac{8}{n^2}$$

$$= \lim_{n\to\infty} \frac{7}{n} + \lim_{n\to\infty} \frac{8}{n^2} = \frac{7.1}{\infty} + \frac{8.1}{\infty} = k$$

Little omega (ω)

- ω is used to find out loose lower bound.
- Big omega (Ω) may represent exact order of growth but little omega (ω) denote a loose lower bound that is not asymptotically tight
- If $f(n) \in \omega(g(n))$ if and only if

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

Problem

(Q) $f(n) = 4n+6$ $g(n) = 1$ Check whether $f(n) \in \omega(g(n))$

$$\lim_{n\to\infty} \frac{4n+6}{1} = 4\infty + 6 = \infty.$$

Thus proved that $f(n) \in \omega(g(n))$

Asymptotic properties

4 main properties

(1) reflexive (2) transitive (3) Symmetric 4) transitive

(1) reflexive

If $f(n)$ is a fn then $f(n) = O(g(n))$

$f(n) = \Omega(g(n))$

$f(n) = \Theta(g(n))$

ex: $f(n) = n$.

$f(n) = O(n) = \Omega(n) = \Theta(n)$.

(2) Transitive.

If $f(n)$ is $O(g(n))$ and $g(n)$ is $O(h(n))$ then

$f(n) = O(h(n))$.

ex: $f(n) = n$, $g(n) = n^2$, $h(n) = n^3$. By transitive property

$f(n) \Rightarrow n = n^2$

$g(n) \Rightarrow n^2 = n^3$

$h(n) \Rightarrow n = n^3$

$a = b$
$b = c$
$a = c$

* It is true for O & Ω

(3) Symmetry

- Symmetry property true for Θ only.
- If $f(n) = \Theta(g(n))$ if and only if $g(n) = \Theta(f(n))$

ex: $f(n) = n^2$, $g(n) = n^2$ then $g(n) = n^2$

(4) Transpose

- If $f(n) = O(g(n))$ if and only if $g(n) = \Omega(f(n))$.
- Transpose property true for O & Ω

ex: $f(n) = n$ & $g(n) = n^2$ then $f(n)$ is $O(n^2)$ & $g(n)$ is $\Omega(n)$

General properties for asymptotic

(1) If $f(n) = O(g(n))$ then $a \times f(n)$ is $O(g(n))$.

It is true for O & Ω

(2) Prove that if $f(n) = 3n^2 + 2$ is $O(n^2)$ if $a = 5$

$f(n) = 3n^2 + 2$

$a \times f(n) = 5(3n^2 + 2) = 15n^2 + 10 \Rightarrow O(n^2)$

(3) Addition / manipulation (multiplication) property

If $f(n) = O(g(n))$ and $d(n) = O(e(n))$ by manipulation property $f(n) + d(n) = O(\max(g(n), e(n)))$

(4) $f(n) = O(n)$, $g(n) = O(n^2)$ find $f(n) + g(n)$.

$f(n) + d(n) = O(\max(g(n), e(n))) = O(\max(n, n^2))$

$= O(n^2)$

(3) multiplicative property

If $f(n) = O(g(n))$ & $d(n) = O(e(n))$ then $f(n) \times d(n) = O(g(n) \times e(n))$

(Q) $f(n) = 1/n$, $g(n) = n^2$

$f(n) \times g(n) = O(1/n \times n^2) = O(n)$

Log formulas

* $\log_a a = 1$
* $\log_a 1 = 0$
* $\log_m ab = \log_m a + \log_m b$
* $\log_m a/b = \log_m a - \log_m b$
* $\log_m a^b = b \log_m a$
* $a^b = n$ then, $b = \log_a n$
* $a^{\log_b c} = c^{\log_b a}$

$\log_2 2 = 1$

$\log_2 1 = 0$

ex $2^{\log_2 n} = n^{\log_2 2}$

$= n \times 1$

$= n$

30/11/21

Recurrence

- The way to find out the time complexity of recursive algorithm is known as recurrence relation.
- 4 methods to find out the recurrence relation.
  (1) Substitution method.
  (2) Iteration method
  (3) Recursion-tree method
  (4) Masters theorem

(Q) Solve $T(n) = \begin{cases} 1 & \text{if } n=0 \\ T(n-1)+1 & n>0 \end{cases}$

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

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

$= T(n-2) + 2$

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

$= T(n-3) + 3$

$= T[(n-4)+1] + 3$

$= T(n-4) + 4$

$\vdots \quad \vdots$

$= T(n-k) + k$

$= T(0) + n = 1 + n = O(n)$

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

$= T(n-2)+1$

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

$= T(n-3+1$

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

$= T(n-4)+1$

$\therefore n-k=0 \Rightarrow n=k$

(Q) $T(n) = \begin{cases} 1, & \text{if } n=0. \\ T(n-1)+n, & \text{if } n \geq 0. \end{cases}$

$= T(n-1)+n$

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

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

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

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

$T(n-4)+3n-3-3$

$T(n-4)+3n-6+n$

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

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

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

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

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

(Q) $T(n) = \begin{cases} 1 & \text{if } n=0 \\ T(n-1)+n & \text{if } n \geq 0. \end{cases}$

$= T(n-1)+n$

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

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

$= T[(n-4)+(n-3)+(n-2)+(n-1)+n]$

$= \vdots$

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

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

$=$ Where $n=k$

$T(n-1) =$

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

$T(n-2) =$

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

$T(n-3) =$

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

$T(n-4)+n-3.$

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

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

$= 1+T(n^2) = O(n^2)$

(Q) $T(n) = \begin{cases} a & \text{for } n=0. \\ a+T(n-1) & \text{for } n>0. \end{cases}$

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

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

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

$= a+a+a+a+T(n-4))$

$\vdots$

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

If $n=k$, $n-k=0$.

$= T(2n)+T(0) = 2n-1 = O(n)$

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

$= a+T(n-2)$

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

$= a+T(n-3)$

$T(n-3) = a+T(n-4)$

6/12/21

Substitution method for solving recurrence in 2 steps

① Guess the form of the solution ie

$T(n) = O(g(n))$

② Use mathematical induction to find constant c, n_0 in the form and show that solution works.

Step-1 : Base step

Prove that the guess is true for initial value.

Step 2 : Inductive step.

Prove that if the guess is true for $T(k) \leq c.g(k)$ for all $k < n$. Then this imply that $T(n) \leq c.g(n)$ for some $c > 0, n > n_0$

Problem.

① Solve the following recurrence relation using Substitution method.

$$T(n) = \begin{cases} 1, & n=1 \\ 2T(\lfloor n/2 \rfloor)+n, & n>1 \end{cases}$$

Guess upperbound

$T(n) = O(n(\log n))$ or

$T(n) \leq c.n\log n$ for some constant c & $n_0 \leq n$.

Hypothesis : For any value k

$T(k) \leq c\,k\log k$ for $k < n$. Here $k = n/2$, $n/2 < n$

Inductive Step : $T(n) = 2T(\lfloor n/2 \rfloor) + n$.

Substitute our hypothesis in the recurrence equation and prove that if it is true for value k then it true for value n.

Plug in $n/2$ into our hypothesis.

$\log a/b = \log$

$\log 2 = 1$

$$\begin{aligned} T(n) &= 2T(\lfloor n/2 \rfloor)+n \\ &\leq 2c(\lfloor n/2 \rfloor)\log(\lfloor n/2 \rfloor)+n \\ &\leq c[n\log(n/2)]+n \\ &\leq cn\log n - cn\log 2 + n \\ &\leq cn\log n - cn + n \longrightarrow \text{lower bound.} \\ &= cn\log n \end{aligned}$$

Thus proved that the result got from hypothesis should be less than or equal to $n\log n$.

Hence our guess solution $n\log n$ is correct. So $T(n) = O(n\log n)$ is true for recursively calling sorting such as merge sort & quick sort.

(Q) $T(n) = \begin{cases} 1, & n=1 \\ 2T(n/2)+n & \text{if } n>1 \end{cases}$

Guess $T(n) = O(n)$.

(A) Guess upper bound $T(n) = O(n)$ or

$T(n) \leq cn$ for some constant c

Hypothesis: For any value of k

$T(k) \leq cn$, for $k < n$. Here $k = n/2$, $n/2 \leq n$.

Inductive step: $T(n) = 2T(\lfloor n/2 \rfloor) + n$.

plug into $n/2$ into our hypothesis.

$$T(n) = 2T(\lfloor n/2 \rfloor) + n$$
$$\leq 2c(n/2) + n$$
$$\leq cn + n$$
$$= (c+1)n, \text{ which is not equal to } cn$$

The above inequality does not hold because $c+1$ can't be less than c. Hence $T(n) \neq O(n)$. So our guess $O(n)$ is not a solution for merge sort or quick sort.

10/12/21

Merge Sort.

A → array
p → position of 1st elem
r → position of last elem

MERGE-SORT (A, P, r)

1. if $p < r$
2. then $q \leftarrow \lfloor (p+r)/2 \rfloor$
3. MERGE-SORT (A, P, q)
4. MERGE-SORT (A, q+1, r)
5. MERGE (A, P, q, r)

ex:

1	2	3	4	5	6	7	8
1	5	7	8	2	4	6	9

1) $p < r$
2) ∴ Find the mid

Procedure for merge.

MERGE (A, P, q, r)

1. $n_1 \leftarrow q - p + 1$
2. $n_2 \leftarrow r - q$
3. Create arrays $L[1, \ldots, n_1+1]$ and $R[1 \ldots n_2+1]$
4. for $i \leftarrow 1$ to n_1
5. do $L[i] \leftarrow A[p+i-1]$
6. for $j \leftarrow 1$ to n_2

7) do $R[j] \leftarrow A[q+j]$

8) $L[n_1+1] \leftarrow \infty$

9) $R[n_2+1] \leftarrow \infty$

10) $i \leftarrow 1$

11) $j \leftarrow 1$

12) for $k \leftarrow p$ to n

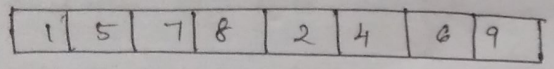
13) do if $L[i] \leq R[j]$

14) then $A[k] \leftarrow L[i]$

15) $i \leftarrow i+1$

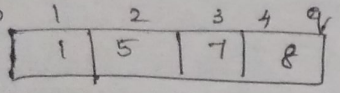
16) else $A[k] \leftarrow R[j]$

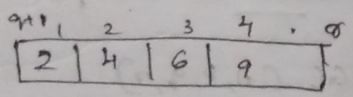
17) $j \leftarrow j+1$



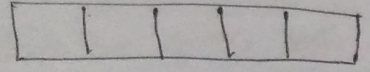
1.) $p < r$

2) :. Find the mid $q \leftarrow (p+r/2)$

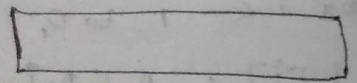
3) 

4) 

5) merging

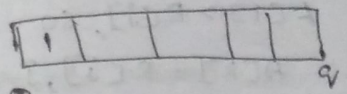
1. $n_1 = q-p+1 = 4-1+1 = 4$
2. $n_2 = r-q = 8-4 = 4$
3. 

$L[p \ldots n_1+1]$



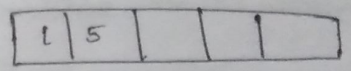
$R[q+1 \ldots n_2+1]$

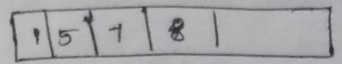
4. Repeat loop ie $i = 1$ to 4

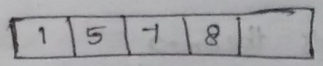
5. a) 

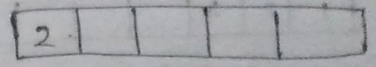
$L[i] = A[p+i-1]$

$= A[1+1-1] = A[1]$

b) 

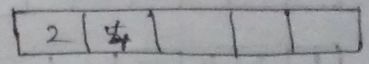
c) 

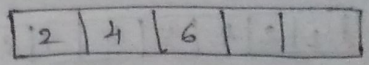
d) 

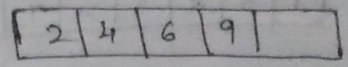
7a) 

$R[j] = A[q+j]$

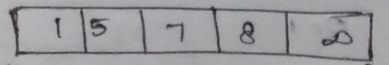
$= A[4+1] = A[5]$

b) 

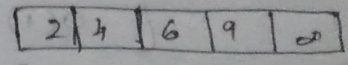
c) 

d) 

6. Repeat loop ie $j = 1$ to 4

8. 

put $(n_1+1) = \infty$

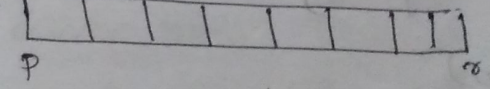
9. 

put $n_2+1 = \infty$

10. $i = 1$

11. $j = 1$

12. Create array k, contains $[p \ldots r]$ elements

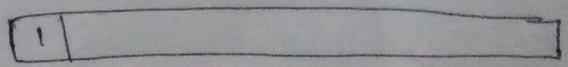


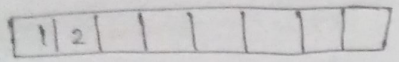
13. if $L[i] \leq R[i]$

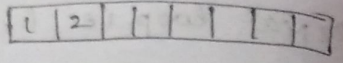
$1 \leq 2$

14. $A[k] = L[i]$ ie $A[i] = 1$

15) $i = i+1$

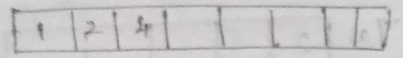
a) 

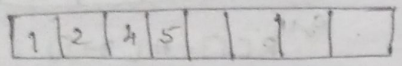
b) 

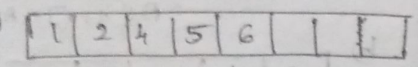
16) 

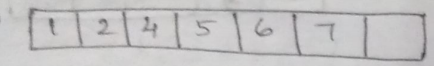
L[i] > R[j], so,

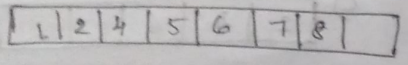
A[k] = R[j], j = j+1

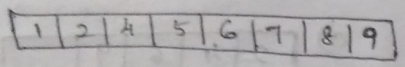
c) 

d) 

e) 

f) 

g) 

h) 

Time complexity By Master's theorem,

$$T(n) = \begin{cases} 2T(n/2) + O(n) & , n>1 \\ O(1) & , n=1 \end{cases}$$

$a=2$, $b=2$, $k=1$, $P=0$

$k = 2^1 = 2$

$a = b^k$

(2) i

$T(n) = \Theta(n^{\log_b a} \log^{P+1} n)$

$= \Theta(n \log n)$

Divide and Conquer Method

(i) Divide mean divide the unsorted list into two sublists equally.

(ii) Sort the each of the 2 subset of sublists recursively

(iii) Combine the 2 sorted sublists back into one sorted list.

If we have one element for sorting, then time complex

$T(n) = O(n)$

If we have $n>1$, then recursively solve, 2 subproblem each of size $n/2$, which contribute $2T(n/2)$ to the running time and combine 'n' sorted sublist.

General equation

$$T(n) = \begin{cases} O(1) & , \text{if } n=1 \\ 2T(n/2) + Cn & , \text{if } n>1 \end{cases}$$

Control abstraction for divide and conquer algo

function DC(x)

If x is sufficiently small then return x

else

Decompose x into simpler instance

$x_1, x_2, \ldots, x_n$

for i=1 to n

do

$y_i = DC(x_i)$

Recombine Y_i, Return Y

STRASSON'S MATRIX MULTIPLICATION

```
for (i=0; i<n; i++)
for (j=0; j<m; j++)
Printf ("%d", &a[i][j])
for (i=0; i<r; i++)
for (j=0; j<c; j++)
Printf ("%d", &b[i][j])
if (m==c) {
for (i=0; i<n; i++)
for (j=0; j<c; j++)
for (k=0; k<c; k++)
d[i][j] = (d[i][j] + a[i][k] * b[k][j]);
}.
Printf ("%d", d[i][j]);
```

Given 2 square matrices A & B of size n×n

$$A = \begin{bmatrix} 1 & 3 \\ 5 & 7 \end{bmatrix} \qquad B = \begin{bmatrix} 2 & 4 \\ 6 & 8 \end{bmatrix}$$

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

```
mulch()
{ for (i=0; i<m; i++)
  { for (j=0; j<n; j++)
    { for (k=0; k<n; k++)
      { c[i][j] = a[i][k] * b[j][k]
      }}}
```

We have 3 methods for finding matrix multiplication

(i) Naive method

(ii) Divide and conquer

(iii) Strassen's matrix multiplication

(i) Naive method

Time complexity is $O(n^3)$ because it contains 3 loops.

(ii) Divide and conquer method

Divide matrix A & B into 4 submatrix of size $n/2$.

A, B, C are square matrix of size $n \times n$, then

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$ are submatrix of A having size $n/2 \times n/2$.

$B_{11}, B_{12}, B_{21}, B_{22}$ be the submatrix of B having size $n/2 \times n/2$.

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

$C_{11} = A_{11}B_{11} + A_{12}B_{21}$

$C_{12} = A_{12}B_{12} + A_{12}B_{22}$

$C_{21} = A_{21}B_{11} + A_{22}B_{21}$

$C_{22} = A_{21}B_{12} + A_{22}B_{22}$

The recurrsive matrix multiplication gives 8 multiplication and 4 addition (means addition of matrix). So the time complexity for divide and conquer approach of matrix multiplication is

$$T(n) = 8T(n/2) + n^2$$

Using master's theorem

$a = 8,\ b = 2,\ k = 2$

$b^k = 2^2 = 4$

$a > b^k$

$$T(n) = \Theta(n \log_b a) \Rightarrow O(n \log_2 8)$$
$$O(n \log_2 2^3)$$

(iii) Strassen's method

Strassen reduce the no: of recurrsive call to 7 so he design a formula,

$P = (A_{11} + A_{22}) \cdot (B_{11} + B_{22})$

$Q = (A_{21} + A_{22})\ B_{11}$

$R = A_{11}\ (B_{12} - B_{22})$

$S = A_{22}\ (B_{21} - B_{11})$

$T = (A_{11} + A_{12})\ B_{22}$

$U = (A_{21} - A_{11})\ (B_{11} + B_{12})$

$V = (A_{12} - A_{22})\ (B_{21} + B_{22})$

$C_{11} = P + S - T + V$

$C_{12} = R + T$

$C_{21} = Q + S$

$C_{22} = P + R - Q + V$

Time complexity of SNM is $T(n) = 7T(n/2) + d$. Here, 7 multiplication and 18 addition and Substractions. So by using master's theorem

$a = 7,\ b = 2,\ k = 2$

$b^k = 2^2$

$a > b$

$$T(n) = \Theta(n \log_b a)$$
$$= O(n \log_2 7)$$
$$= O(n^{2.807})$$

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \quad \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

Quick Sort

Quick Sort is a recursive algorithm that follows divide and conquer approach in best case and average case. Time complexity of best case quick sort is $[n \log n]$ and time complexity of worst case is $O(n^2)$.

Given an array, take any element from the array and call it as pivot value and put that pivot value in a position in such a way that element lesser than pivot value is left hand side and element greater than pivot is RHS. Now we have 2 sublists and perform recursively quicksort for each sublist.

Partition procedure

(i) Increment i until found a value greater than or equal to pivot.

(ii) Decrement j until found a value lesser than pivot.

(iii) Exchange i & j

(iv) Whenever i & j crosses then we swap j^{th} position v^{th} pivot value.

Step(i) : Take 10 as the pivot element.

(ii)

10	16	8	5	9	∞
					j

i increments and j decrements.

(iii) $i > pivot \Rightarrow 16 > 10$ - T

$j < pivot \Rightarrow 9 < 10$ - T

∴ swap 16 & 9

10	9	8	5	16	∞
	i			j	

(iv) $i > p \Rightarrow 8 > 10$ - F

$i+1 \Rightarrow i = 5$

10	9	8	5	16
			i	j

$i > p \Rightarrow 5 > 10$ - F

$i+1 \Rightarrow i = 16$., $i > p \Rightarrow 16 > 10$

$j-1 \Rightarrow j = 5$, $j < p$, $j = 5$

10	9	8	5	16
			j	i

Swap the pivot with j

5	9	8	10	16

LHS of pivot is less

RHS of pivot is greater.

Pivot = 5 = i, j = 8

(5)	9	8	10	16

i j

i++ ⇒ i = 9

5	9	8	10	16

i j

i > p ⇒ 9 > 5 — T

j < 5 ⇒ 8 < 5 — F

j-- ⇒ j = 9 ⇒ j < 5

5	9	8	10	16

j i

j < 5 ⇒ false, i = 9 ∴ we can't interchange.

Then, Pivot = 9

5	(9)	8	10	16

i j

i > p ⇒ false

j < p = True

5	(9)	8	10	16

j i

∴ swap pivot and i, because j is overcrossed i.

5	8	9	10	16

2. Apply quicksort for the following elements.

10, 16, 8, 12, 15, 6, 3, 9, 5

10	16	8	12	15	6	3	9	5	∞

i j

i = 16, i > p ⇒ 16 > 10 - T

j--, j = 5, j < p ⇒ 5 < 10 - T

10	5	8	12	15	6	3	9	16

i j

i = 8, i > p ⇒ 8 ≯ 10

i = 12, i > p ⇒ 12 > 10 — T

j = 9, j < p ⇒ 9 < 10 — T

10	5	8	9	15	6	3	12	16

i j

i = 15, i > p ⇒ 15 > 10 - T

j = 3, j < p ⇒ 3 < 10 - T

10	5	8	9	3	6	15	12	16

j i

i = 6, 6 ≯ 10.

i = 15, 15 > 10

j = 6, 6 < 10, T

j overcrossed i, swap pivot & j

(6)	5	8	9	3	10	15	12	16

pivot = 6

i=6, j=10

i=5, F

i=8, T

j=3, T

6	5	3	9	8
		ij		

i=9, T

j=9, F

j=3, F

swap with pivot and j

3	5	6	9	8

pivot=9

3	5	6	8	9

15	12	16

i=12 - F

j=16 - F

i=16 - T

j=12 - T

Swap with pivot

12	15	16

The sorted list is:

3	5	6	8	9	12	15	16

Partition

```
Quick (l, b)
{ if l<b
  { j = partition (l, b)
    Quicksort (l, j)
    Quicksort (j+1, high)
  }.
}.

Partition(l, b)
{ pivot = A[low]
  i = low, j = high;
  while (i<j)
    { do
        { i++
        } while (A[i] ≤ pivot)
      do
        {j--
        } while (A[j] ≥ pivot)
    if (i<j)
      • swap A[i], A[j]
    }
    swap A[low], A[j]
    return j (partition)
}
```

Quick Sort

1) if p<r then
2) q ← partition (A,P,q)
3) Quicksort (A,P,q-1)
4) Quicksort (A, q+1, r)

Partition (A,P,r)

1) x ← A[r]
2) i ← p-1
3) for j ← P to r-1
4) do if A[j] ≤ x
5) then i ← i+1
6) Exchange A[i] ↔ A[j]
7) Exchange A[i+1] ↔ A[r]
8) return i+1

Quick Sort

Quick Sort is a recursive algorithm that follows divide & conquer approach in best case & average case. Time complexity of quicksort is bestcase is nlogn and in average case, nlogn. Worst case- O(n²)

Given an array, take any element from array and call it as pivot value. and put that value in that position in such a way that elements lesser than pivot value is LHS and element greater than pivot value is RHS. Now we have 2 sublists & perform recursively quick sort for each sublist.

Perform Quicksort for following numbers using partition functions.

2	8	7	1	3	5	6	4

(A) if p<r, p=1, r=8 ⇒ 1<8. go for partition

p r

2	8	7	1	3	5	6	4

∴ x ← A[r] ⇒ x=4

i = 1-1 = 0.

j = 1 to 7

2	8	7	1	3	5	6	4

i=0, j=1

i=0, j=1. 2 ≤ 4

2	8	7	1	3	5	6	4

i=1
j=1

i=1, j=2. ⇒

2	8	7	1	3	5	6	4

8 ≤ 4 no increment

i=1, j=3

2	8	7	1	3	5	6	4

i j

3 ≤ 4, no increment, no swap

i=1, j=4

2	8	7	1	3	5	6	4

i j

1 ≤ 4.

i++ A[i] ↔ A[j]

i=2, j=4

2	1	7	8	3	5	6	4

i j

i=2, j=5.

2	1	7	8	3	5	6	4

i j

3 ≤ 4.

i++, A[i] ↔ A[j]

i=3, j=6

2	1	3	8	7	5	6	4

i j

5 ≤ 4

no increment, no swap.

i=3, j=7

2	1	3	8	7	5	6	4

i j

6 ≤ 4

no increment, no swap

i=3, j=8.

2	1	3	8	7	5	6	4

i

4 ≤ 4.

i++, A[i] ↔ A[j]

2	1	3	4	7	5	6	8.

Quicksort (A, P, q-1)

Quicksort (A, q+1, 8)

2	1	3	4	7	5	6	8

I st II nd.

Worst case

2	1	3

p=1 3

i=0, j=1

1 ≤ 3.

2	1	3

i=1, j=2

2	1	3

i j

1 ≤ 3.

2	1	3

i,j.

i=3, j=3

we can't fix pivot

7	5	6	8

i=0, j=1 7 ≤ 8.

7	5	6	8

i++

i=1, j=1

i=1, j=2

7	5	6	8

i j

5 ≤ 8.

Time complexity of quicksort algorithm using substitution method

$$T(n) = \begin{cases} 2T(n/2) + O(n), & \text{if } n>1 \\ O(1), & \text{if } n=1 \end{cases}$$

Using Masters theorem.

$T(n) = 2T(n/2) + O(n)$

$a=2,\ b=2$

$k=1,\ p=0$ (Property 2, case 1)

$b^k = 2^1$

$T(n) = \Theta(n^{\log_b^a} \log^{p+1} n)$

$= \Theta(n^{\log_2^2} \log^1 n)$

$= \Theta(n \log n)$

Master's theorem

If $T(n) = aT(n/b) + \Theta(n^k \log^p n)$

$a>=1,\ b>=1,\ k>=0$ and p is real number

Properties

1) If $a>b^k$, then $T(n)$

a) If $p>-1$, then $T(n) = \Theta(n^{\log_b^a} \log^{p+1} n)$

b) If $p=-1$, then $T(n) = \Theta(n^{\log_b^a} \log\log n)$

c) If $p<-1$, then $T(n) = \Theta(n^{\log_b^a})$

3) If $a<b^k$ then

a) If $p \geq 0$, then $T(n) = \Theta(n^k \log^p n)$

b) If $p<0$, then $T(n) = \Theta(n^k)$

$(\log n)^2 = \log n \times \log n$

$(\log^2 n) = \log \log n$

$\log^0 n = 1$

$\log_2 2 = 1$

Q1) $T(n) = 3T(n/2) + n^2$

$a=3,\ b=2,\ k=2,\ p=0$

$b^k = 2^2$

$a<b^k$

$3<2^2$ (Property 3, case 1)

$T(n) = \Theta(n^k \log^p n)$

$= \Theta(n^2 \cdot 1)$

$= \Theta(n^2)$

$\log^0 n = 1$

Q2) $T(n) = 2T(n/2) + n \log n$

$a=2,\ b=2,\ k=1,\ p=1$

$a = b^k$ (Property 2, case 1)

$T(n) = \Theta(n^{\log_b a} \log_b^{p+1} n)$

$= \Theta(n^{\log_2 2} \log^{0+1} n)$

$= \Theta(n \log^2 n) = \Theta(n \log \log n)$

Q3) $T(n) = 5T(n/4) + n^2$

$a = 5, b = 4, k = 2, p = 0, b^k = 4^2$

$5 < 4^2$ (Property 3, case I)

$T(n) = \Theta(n^k \log^p n)$

$= \Theta(n^2 \times \log^0 n)$

$= \Theta(n^2)$

4) $T(n) = 7T(n/2) + n^2$

$a = 7, b = 2, k = 2, p = 0$

$b^k = 2^2$

$7 > 2^2$ (Property 1)

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

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

5) $T(n) = 2T(n/2) + \frac{n}{\log n}$

$T(n) = 2T(n/2) + n \log^{-1} n$

$a = 2, b = 2, k = 1, p = -1$

$b^k = 2$

$a = b^k$ (Property 2, case 2)

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

$= \Theta(n^{\log_2 2} \log \log n)$

$= \Theta(n \log \log n)$

$= \Theta(n \log^2 n)$