Design & Analysis of Algos

- Features of good Algos:

- Efficiency
  → Time-complexity
  → Space Complexity

- Simplicity
  → Program Complexity
  → Easy to maintain

- Correctness
  → Formal method: related to program verification
  → Informal method

- RAM model as a generic implementation of an algorithm

- Algo is a specific sequence of steps to achieve a specific task

- Problem as a mapping to input to output

- Data structure as a systematic way of organizing and manipulating data
  → Data organization
  → Funcs for manipulating data

Insertion Sort

```
InsertionSort(A, n)
    for j ← 2 to n
        do key ← A[j]
           i ← j-1
           while i>0 and A[i] > key
                 do A[i+1] ← A[i]
                    i ← i-1
           A[i+1] ← key
```

Worst-Case : $O(n^2)$
Average-Case: $\theta(n^2)$
Best-case : $\Omega(n)$

- Algo finds the correct position to place the element

- Situations to use in:
- Already Sorted / Nearly sorted data
- Small Datasets
  → Simple
  → Low constant factors

- Systems with minimal memory usage
  → Space complexity of O(1)

- Online sorting
  → Sequential data

Q- Apply RAM Model and calculate Running Time

	Individual Cost	Repetition	Total
x ← 0	c_1	1	c_1
for i ← 1 to n	c_2	n+1	$c_2(n+1)$
temp ← i+1	c_3	n	$c_3(n)$
for j ← 1 to n	c_4	$n(n+1)$	$c_4(n^2+n)$
If (j mod 2 = 0) Then	c_5	n^2	$c_5(n^2)$
x ← x + temp	c_6	$n^2(k)$ $0 \le k \le 1$	$c_6(n^2k)$
Else	c_7	$n^2(1-k)$	$c_7[n^2(1-k)]$
x ← x - 1	c_8	$n^2(1-k)$ $0 \le k \le 1$	$c_8[n^2(1-k)]$

* Checking of else statement

$T(n) = c_1 + c_2(n+1) + c_3(n) + c_4(n^2+n) + c_5(n^2) + c_6(n^2 k) + c_7[n^2(1-k)] + c_8[n^2(1-k)]$

$= n^2(c_4 + c_5 + kc_6 + c_7 - kc_7 + c_8 - kc_8) + n(c_2 + c_3 + c_4) + (c_1 + c_2)$

$= n^2(c_4 + c_5 + c_7 + c_8 + k(c_6 - c_7 - c_8)) + n(c_2 + c_3 + c_4) + (c_1 + c_2)$

$T(n) = An^2 + Bn + C$, a quadratic-time algorithm

where $A = c_4 + c_5 + c_7 + c_8 + k(c_6 - c_7 - c_8)$
$B = c_2 + c_3 + c_4$
$C = c_1 + c_2$

Asymptotic Analysis

- Mathematical method for evaluating an algorithm's efficiency, how the performance (time or space complexity) scales with large input sizes

- Types of Analysis
- Worst-case
→ Upper Bound
→ gurrantee that algo would not run longer regardless of input size

- Best case
→ Lower Bound
→ Input is the one combo for which algo runs the fastest
$LB \leq RT \leq UB$

- Average case
→ Prediction about running time
→ Assumes input is random

- To compare algos we express them in form of functions of input size n.
→ Independant of machine time, programming style etc.

- Big OH (O) Notation:

- $O(g(n)) = f(n)$: there exists +ve constants c, and n_0 such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0$

Q- $2n^2 = O(n^3)$ — $g(n) = n^3$, $f(n) = 2n^2$

$2n^2 \leq cn^3$
$2 \leq cn$ — Simplifying
$2 \leq n$ — putting $c=1$
$c=1$, $n_0 = 2$

Hence expression is valid for $c=1$ & $\forall\ n \geq n_0$

Q- $n^2 = O(n^2)$

$n^2 \leq cn^2$
$1 \leq c$ — Simplifying
$1 \leq c$ — $n=1$
$1 \leq c$ — $n=2$

Hence exp is valid for $c=1$, $n \geq 1$

$f(n) \leq cg(n)$
$n^2 \leq cn^2$
$1 \leq c$

Q- $1000n^2 + 1000n = O(n^2)$

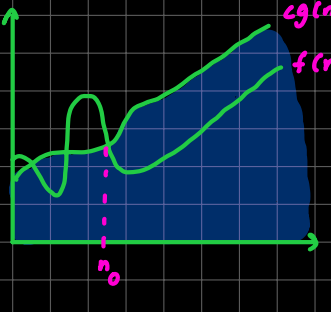
$1000n^2 + n^2 = O(n^2)$; $1000n = n^2$
$1001n^2 = O(n^2)$ $1000 = n$

$1001n^2 \leq cn^2$
$1001 \leq c$

// convert into one term superior until left with a single term

exp is valid for $c = 1001$ and $n \geq 1000$

$c = 1001$
$n_0 = 1000$



Q- $n^4+100n^2+10n+50$
→ $O(n^4)$

- $10n^3+2n^2$
→ $O(n^3)$

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

- 10
- constants = $O(1)$

Q- $30n+8 = O(n)$
$31n = O(n)$ $\quad n=8$

$31n \leq cn$
$31 \leq c$

$c=31,\ n_0 \geq 8$

Q- $n = O(n^2)$

$n \leq cn^2$
$1 \leq cn$
$cn \geq 1$

Valid for $c=1,\ n_0=1$
for $n \geq n_0$

- Big Omega (Ω):

- $\Omega(g(n)) = f(n)$: there exists +ve constants c, n_0 such that $0 \leq cg(n) \leq f(n)$ for all $n \geq n_0$

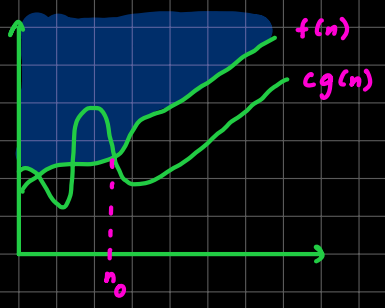
- Q- $5n^2 = \Omega(n)$ $\quad 0 \leq cn \leq 5n^2$

$f(n) = 5n^2$
$g(n) = n$

$5n^2 \geq cn$
$5n \geq c$

$n_0 = 1,\ c = 1$

so expression is valid



Q- $100n+5 = \Omega(n^2)$ $\quad 0 \leq cn \leq 100n+5$

$100n+n \geq cn^2$ $\quad 5=n$ $\quad -5$
$101n \geq cn^2$
$101 \geq cn$

* Subha Average and best-case.

Merge Sort

- MergeSort (A, s, e)
  If (s<e) — c_1 — 1
  $m \leftarrow \lfloor \frac{s+e}{2} \rfloor$ — c_2 — 1
  MergeSort (A, s, m) — $T(\frac{n}{2})$ — 1
  MergeSort (A, m+1, e) — $T(\frac{n}{2})$ — 1
  Merge (A, s, m, e) — $\theta(n)$ — 1

$T(n) = c_1 + c_2 + T(\frac{n}{2}) + T(\frac{n}{2}) + \theta(n)$

$= 2T(\frac{n}{2}) + \theta(n)$ $\Rightarrow$ Recurrence Eq

- Worst-case = Best-case = Average-case = $O(n \log n)$
- space complexity = $O(n)$

- cases to be used in:
- Large Datasets
→ complexity remains same at any dataset, other algos degrade in performance

- External sorting
→ when data is large to fit into main memory

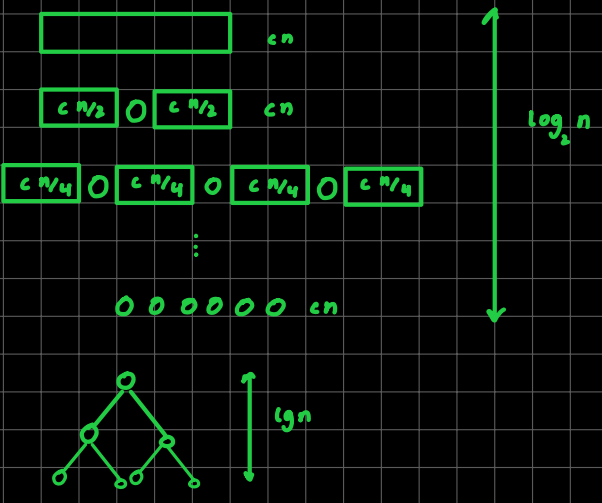
- stable-sorting requirement
→ Preserves original order of identical values; databases

- Sorting linked lists
- Parallel processing
- Inversion count problems

Quick Sort

- Quicksort (A, s, e)
  if (s<e)
    m ← Partition (A, s, e) — $\theta(n)$
    Quicksort (A, s, m-1)
    Quicksort (A, m+1, e)

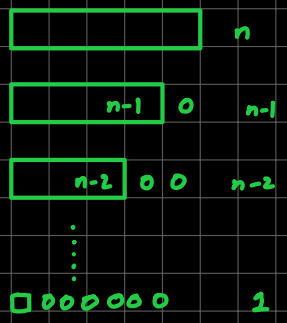
- Best-Case Analysis



$T(n) = 2T(\frac{n}{2}) + \theta(n)$

$= cn.\log n$
$= \theta(n \log n)$
$= \Omega(n \log n)$

- Worst-Case Analysis



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

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

$= \theta(n^2)$
$= O(n^2)$

- Average-Case Analysis
  10 : 90

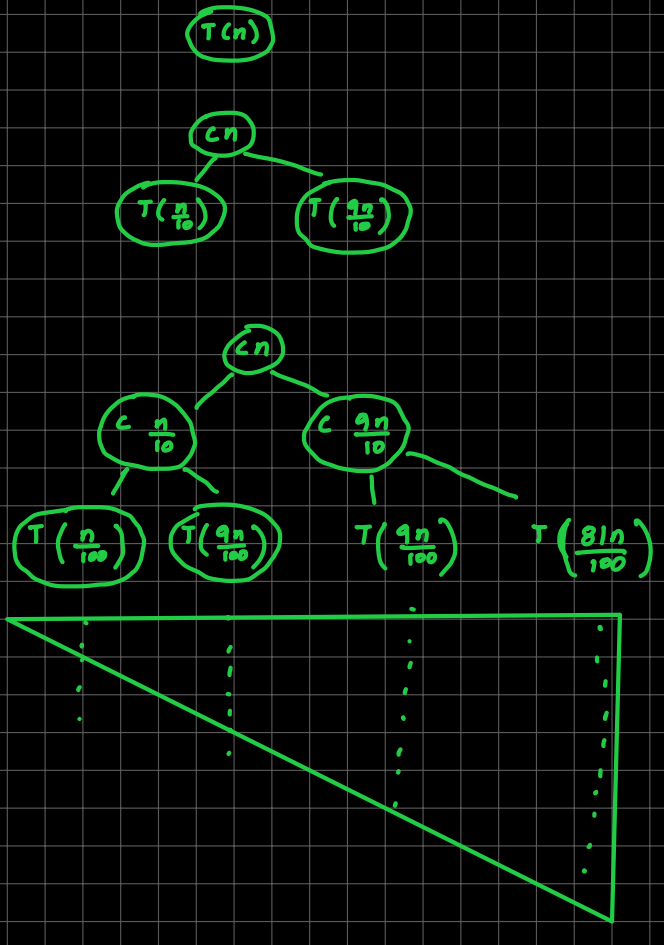
$T(n) = T(\frac{n}{10}) + T(\frac{9n}{10}) + \theta(n)$

$T(\frac{n}{10}) = T(\frac{n}{100}) + T(\frac{9n}{100}) + \theta(\frac{n}{10})$

$T(\frac{9n}{10}) = T(\frac{9n}{100}) + T(\frac{81n}{100}) + \theta(\frac{9n}{10})$

- Tree
  # of nodes : n
  height : log n
  Leaves : $\frac{n}{2}$ approx

- Array
- elements : n
- Permutation : n!
- Branches : n!
- Leaves : n!
- Total nodes : 2n!
- height = $\log(2n!) \approx O(n \log n)$



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

- Suitable
- In-place → used in limited memory situations
- Finding k^{th} smallest / largest element in an unsorted array

Radix Sort

- MSASP (A,s,e)
  if (s<e)
    $m \leftarrow \frac{s+e}{2}$
    L ← MSASP (A,s,m)
    R ← MSASP (A,m+1,e)
    C ← combineAr(A,s,m,e)
    Return Max(L,R,C)

Radix sort = d (sorting Algo)

= $d(n^2)$ ins/Quick/Bub/Sel

= d (n log n) Merge

= d (n+k) k≤10 k= base

=> d(n) k<<n d= N.o of digits in a number

=> $\theta(n)$ if d<<n

- Situations to use:
- Sorting fixed-width data
- Large datasets with small number of digits

- Average-case:
- O(nk) or O(d(n+b))

k,d= # of digits

- Worst = $\theta(nd)$
- Best= $\theta(nk)$

Count-Sort

- A, B, C array

A= Original Array
B= Final-Sorted Array
C= Cumulative frequency array

- Stable
- In-place → transforms input data structure itself without requiring additional storage.
- Worst= Average = Best = O(n+k)

k= Range of input values (Max-Min+1)

- Space= O(1)

$B[C[A[j]]] \leftarrow A[j]$

$C[A[j]] \leftarrow C[A[j]] - 1$

start from end, place, and subtract from count array

- Situations:
- Sorting integers with small range
- Sorting categorical data
- Stability required → relative order of identical elements matter

- Inversion count
→ worst, average, best= O(n log n)

- Maximum Sub-array
→ $T(n) = 2T(\frac{n}{2}) + \theta(n)$

- $\theta(n \log n)$