```
Algorithm Sum (A,n)
{ integer n, A[n], i, sum;        ⌈2B
    Sum=0;                        (2B
  for i ← 1 to n
    Sum = Sum + A[n];
  }
```

Time = O(n)
Space = O(1)

Input n, A[n]
Auxiliary space = 1 Bytes
O(1)

Space Complexity = We define the space used by an algorithm to be the number of memory cells (or words) needed to carry out the computational steps required to solve an instance of the problem excluding the space allocated to hold the input. In other words, it is only work space required by the algorithm

All definations of order of growth and asymptotic bounds pertaining to time complexity carry-over to space complexity. It is clear that the work space cannot exceed the running time of the algorithm, as writing into each memory cell requires atleast a constant amount of time.

Thus if we let T(n) and S(n) denote respectively, the time and space complexities of the algorithm, then

$$S(n) = O(T(n))$$

Example: In algorithm linear search, only one memory cell is used to hold the result of the search. If we add local variables eg for looping, we conclude that amount of space needed is O(1). This is also the case in algorithms Binary Search, Selection Sort and Insertion Sort.

→ Algo LS (A, n, x)

```
{ int n, A[n], x, i;
  for i ← 1 to n
  { if (x = A[i])
    { print (i);
      return i; }
    }
  return (0); }
```

C2B

Time = O(n)

Space = O(1)

→ Integer n, A[n];

```
Algorithm RSum (A, n)
{ if (n=1) return (A[n]);
  else
  { return (A[n] + RSum (A, n-1));
    }
  }
```

n-records

Auxiliary/working space

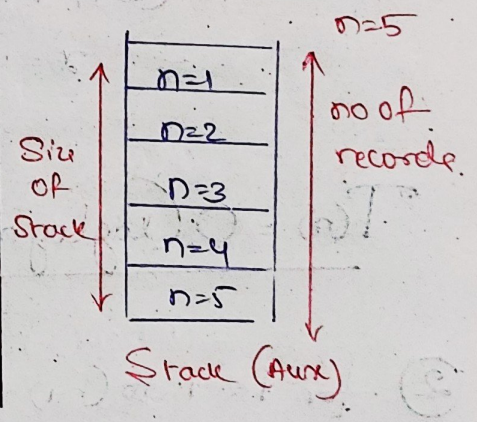
Space = O(n)

Sum of elements of array:

Time = O(n)

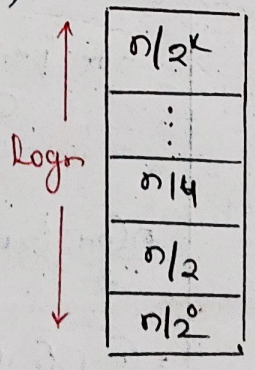
T(n) = T(n-1) + c

Space Complexity



→ Algorithm Do_it (n)

```
{ if (n=1) return,
  else
  { Do_it (n/2); }
  }
```



Time = O(logn)

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

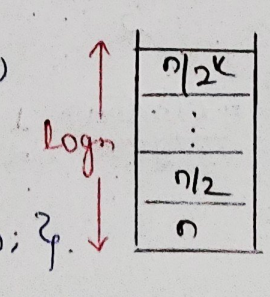
Space = O(logn)

1, 2, 4, 8, ... 2^k

$2^k = n \Rightarrow k = \log n$.

→ Algorithm Test (n)

```
{ if (n=1) return;
  Else { Test (n/2);
         Test (n/2); }
  }
```



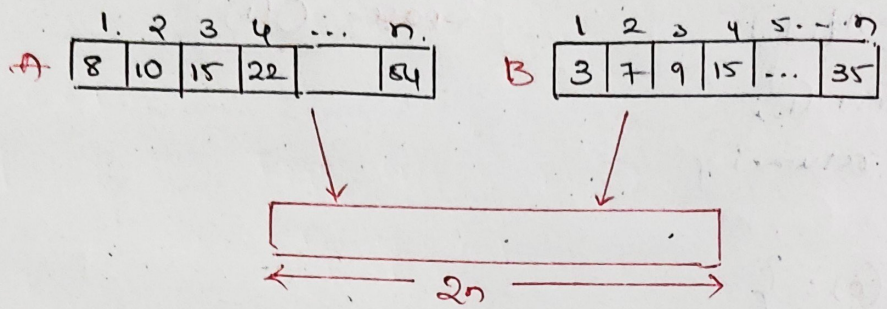
Time = O(n)

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

Space = O(logn)

Homework

Q1. Given two sorted array of size 'n' each, the space complexity to merge the two arrays is O(?)



31/7/2023

Q Obtain Time Complexity:

(1) for (i=2; i≤n; i=i^2) } log log n
  for (j=1; j≤n; j++)
    for (k=1; k≤n; k=k+j)
      x = y+2;

j = 1, j = 2
k = n/1, n/2 ... n/n
n · Σ_{x=1} 1/x = n log n

$T(n) = O(\log\log n \cdot (n \log n))$;

(2) for i ← 1 to (n-1)
  for j ← i+1 to n
    for k ← 1 to j
      print ("*");

What is the no of times '*' gets printed?

$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \sum_{k=1}^{j} O(1)$

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

$i=1 \qquad i=2$

$\left(\frac{n(n+1)}{2} - 1\right) + \frac{n(n+1)}{2} - (1+2) + \ldots$

for i ← 1 to n, n
  C = C+1 ; ⟹ $\sum_{i=1}^{n} O(1)$
  or print ("*");

$\frac{n(n+1)}{2}(n-1) - \left[1 + (1+2) \ldots (1+2+3 \ldots n-1)\right]$

$= \frac{(n-1)(n)(n+1)}{2} - \sum_{j=1}^{n-1} \left(\frac{j(j+1)}{2}\right)$

$= \frac{(n-1)(n)(n+1)}{2} - \frac{1}{2}\left[\sum_{j=1}^{n-1} j^2 + \sum_{j=1}^{n-1} j\right] = \frac{n(n+1)(n-1)}{3}$

Design Strategies.

31/7/23

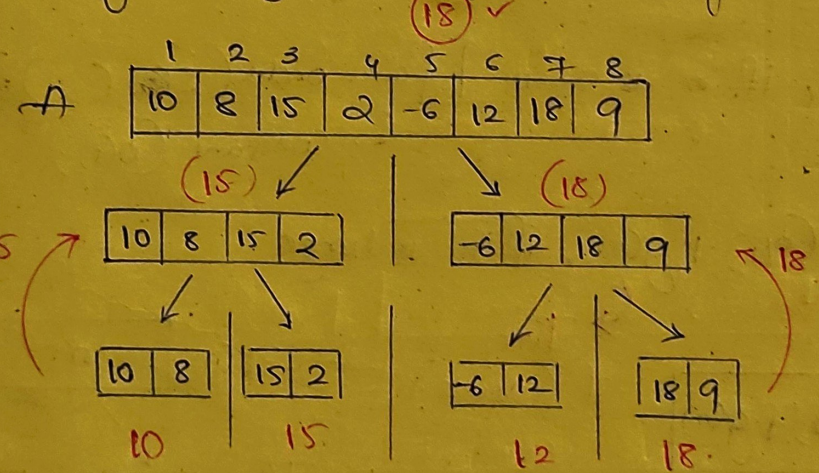
Divide and Conquer : (DandC)

* When the problem becomes large / complex, then divide the problem into sub problems into further sub problems until the sub problem becomes small.

* Solve the smaller problems, combine the results if required to get solution to the original problem

> In general a problem is said to be small if can be solved in one or two basic operations.

eg: Finding the largest element in an array




```
Algorithm DAndC(P)
{ if Small(P) then return S(P);        { Control Abstraction }
  else
  { divide P into smaller instances P1, P2 ... Pk. (k≥1);
    apply DAndC to each of these sub problems;
    return Combine(DAndC(P1), DAndC(P2) .. DAndC(Pk));
  }                                    (recursion)
}
```

→ Algorithm DandC (A, 1, n)

```
{ if (SMALL(1,n))
    return (S(A,1,n));
  else
  { m ← DIVIDE (1,n);
    S1 ← DandC (A, 1, m);
    S2 ← DandC (A, m+1, n);
    Combine (S1, S2); }
}
```

Time Complexity (framework) for DandC Problems

→ let T(n) represent Time Complexity of DandC(n);

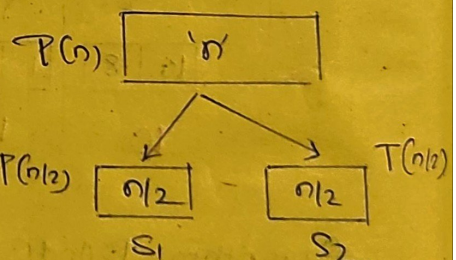
$T(n) = f(n)$, if 'n' is small [$f(n)$ = order(S(n)); → constant (O(1)).

$= 2.T(n/2) + \{g(n)\}$ if 'n' is large

→ Time of (divide + Combine + Small)

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

- no. of Sub problems
- Size of each Sub problem
- Time for divide & Combine.



Generalised Form : $T(n) = a.T(n/b) + g(n)$

- a → No. of subproblems
- n/b → Size of each subproblem
- $a \geq 1$, $b > 1$, $g(n)$ is true

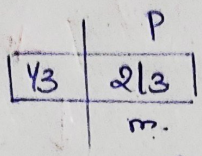
I> Symmetric.

eg: $T(n) = 2T(n/2) + g(n)$

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

$T(n) = 4T(n/2) + g(n)$

II> Asymmetric - I



$T(n) = T(n/3) + T(2n/3) + g(n)$

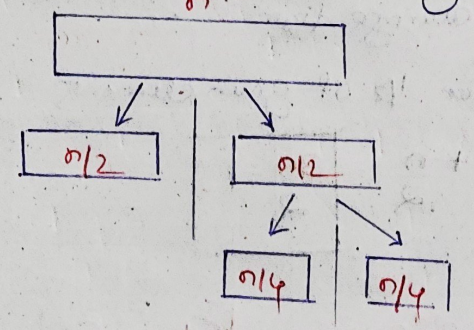
General Form

$T(n) = T(\alpha n) + T((1-\alpha)n) + g(n)$

$0 < \alpha < 1$.

III> Asymmetric - II.

Ex: $T(n) = T(n/2) + T(n/4) + g(n)$



Note: In D and C Strategy, divide and conquer is mandatory but combine is optional. (depends on application)

① MaxMin : Procedure to find max and min (Simultaneously) in an array of size 'n'.

1	2	3	4	5	6	7	8	9	10
8	16	0	-5	18	2	9	60	20	25

Write an Algorithm!

Algorithm Non_DC (A, n, max, min)

{

1. max ← min ← A[1];
2. for i ← 2 to n {(n-1) times}
   { if (A[i] > max) — 1
     max ← A[i];
     else if (A[i] < min) — 1. } instead of "if" if we use "else if" then (n-1) comparisions in best case
     min ← A[i]; }
   }

Q1> Total no of element Comparision made in Non_dc = $2^*(n-1)$ comparision for all cases

↓

(i) Best case: If increasing order: (n-1)

(ii) Worst Case: - 2 (n-1) if elements are in decreasing order

(iii) Average Case: on an average first comparision fails for 1/2 of given elements.

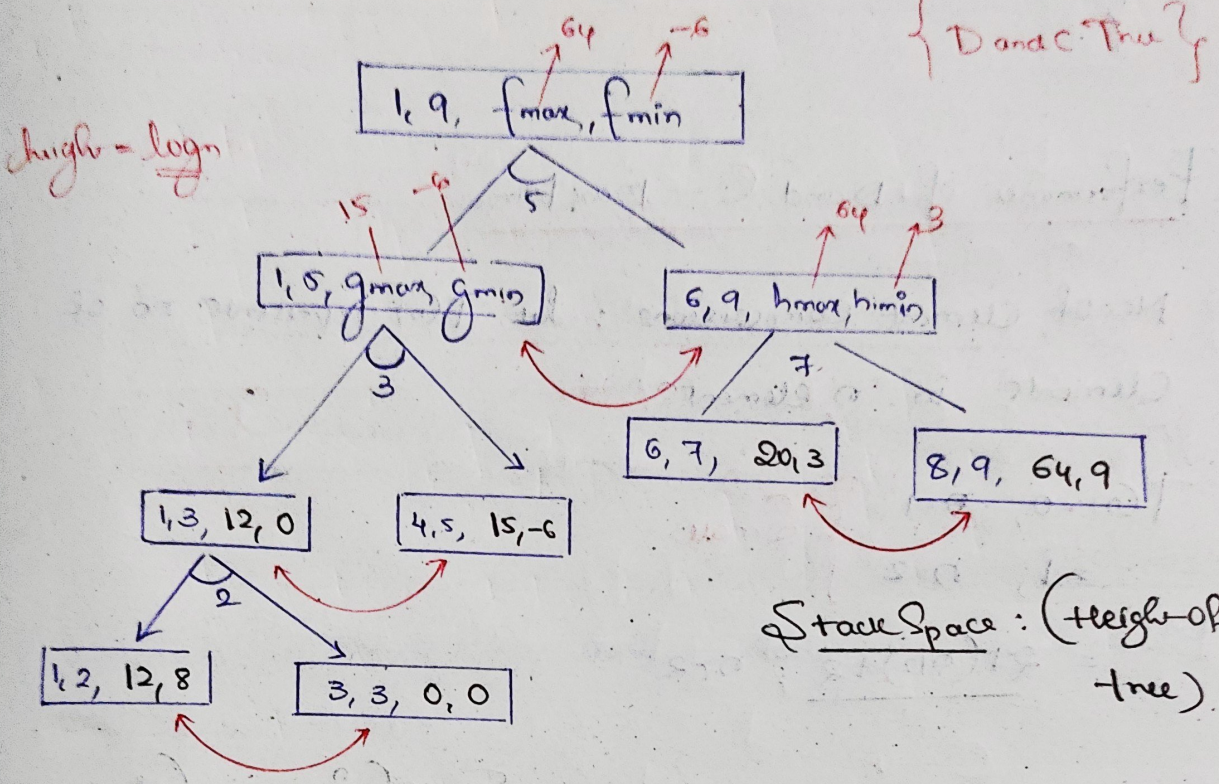
$$(n-1)+\frac{n}{2} = \boxed{\frac{3n}{2}-1}$$

	1st Comp.	2nd Comparision.	[Total Comparision]
1. Best Case – Increasing order	(n-1)	0	(n-1)
2. Worst Case – decreasing order	(n-1)	(n-1)	2(n-1)
3. Avg case	(n-1)	n/2	$(\frac{3n}{2}-1)$

DandC – MaxMin

-A

1	2	3	4	5	6	7	8	9
8	12	0	-6	15	20	3	64	9



Stack Space : (height of tree)

DandC Code for MaxMin

```
Algorithm MaxMin(i, j, max, min)
{
  if (i=j) then max: min = a[i]; // Small(P)
  else if (i=j-1) then // another call of Small(P)
    { if (a[i]<a[j]) then
      { max := a[j]; min := a[i]; }
    else
      { max := a[i]; min = a[j]; }
    }
  else { mid := ⌊(i+j)/2⌋;
         MaxMin(i, mid, max, min);
         MaxMin(mid+1, j, max1, min1);
```

```
    if (max < max1) then max = max1;
    if (min > min1) then min = min1; }
}
```

} Combine.

Performance of D and C – MaxMin

(i) No. of element comparisons : Let T(n) represents no. of elements in 'n' elements.

$$T(n) = \begin{cases} 0, & n=1 \\ 1, & n=2 \\ 2T(n/2)+2, & n>2 \end{cases}$$

} Small.

$$T(n) = 2T(n/2)+2 \quad -(1)$$

$$T(n/2) = 2T(n/4)+2 \quad -(2)$$

$$T(n) = 2[2T(n/4)+2]+2$$

$$= 4T(n/4)+4+2 \quad -(3)$$

$$= 2^2T(n/2^2) + \sum_{i=1}^{2} 2^i$$

$$\vdots$$

$$2^kT(n/2^k) + \sum_{i=1}^{k} 2^i$$

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

$$= \frac{n}{2} + 2\cdot 2^k - 2 = \frac{n}{2} + n - 3$$

$$S_n = \frac{a(r^n-1)}{r-1} = \frac{2(2^k-1)}{2-1}$$

$$= 2^{k+1} - 2$$

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

(→ O(n) → no. of Element Comparisons (all cases))

$$\frac{n}{2^k} = 2$$

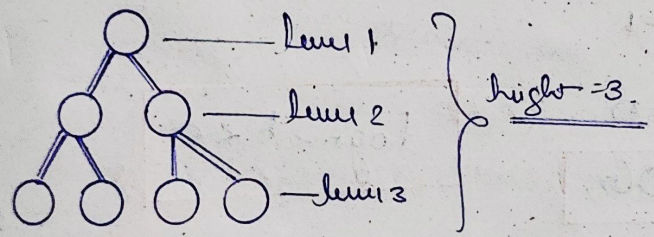
$$\frac{n}{2} = 2^k$$

	Non-DC	DandC
1) Increasing order	$(n-1)$	$\left(\frac{3n}{2}-2\right)$
2) Decreasing order	$2(n-1)$	$\left(\frac{3n}{2}-2\right)$
3) Random order	$\left(\frac{3n}{2}-1\right)$	$\left(\frac{3n}{2}-2\right)$

Space Complexity: Non-DC : $O(1)$; DC : $O(\log n)$

01/8/2023

Q) Given a full binary tree with n-nodes, then height of the binary tree = _____



→ Maximum number of nodes at any level 'i' of a binary tree = 2^{i-1}

→ Total no of nodes in a binary tree of height $h = \sum_{i=1}^{h} 2^{i-1}$

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

$$n = \frac{1}{2}\left[2^{h+1}-2\right] = 2^{h}-1$$

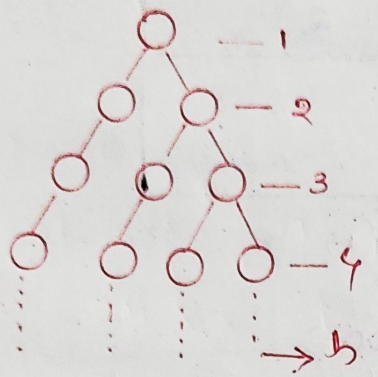
$$\therefore n = 2^{h}-1 \Rightarrow n+1 = 2^{h}$$

$$h = \log_2 n+1 = O(\log_2 n)$$

$S_n = \frac{a(r^n-1)}{r-1}$

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

Q1. Consider a binary tree where root is at level 1 and every other level 'i' of the binary tree has exactly 'i' nodes. The height of such a binary tree having 'n' nodes is order of _____



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

$$h^2 + h = 2n$$

$$h^2 \leq n$$

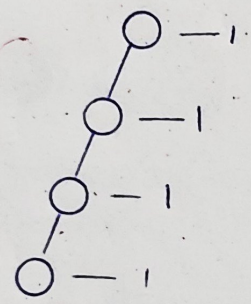
$$h = \sqrt{n}$$

$$\therefore h = O(\sqrt{n})$$

Q2>

Max-height of a binary tree with n-elements is n.

Skewed Tree



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

$$h = n$$

$O(n)$

$$\log n < h < n$$

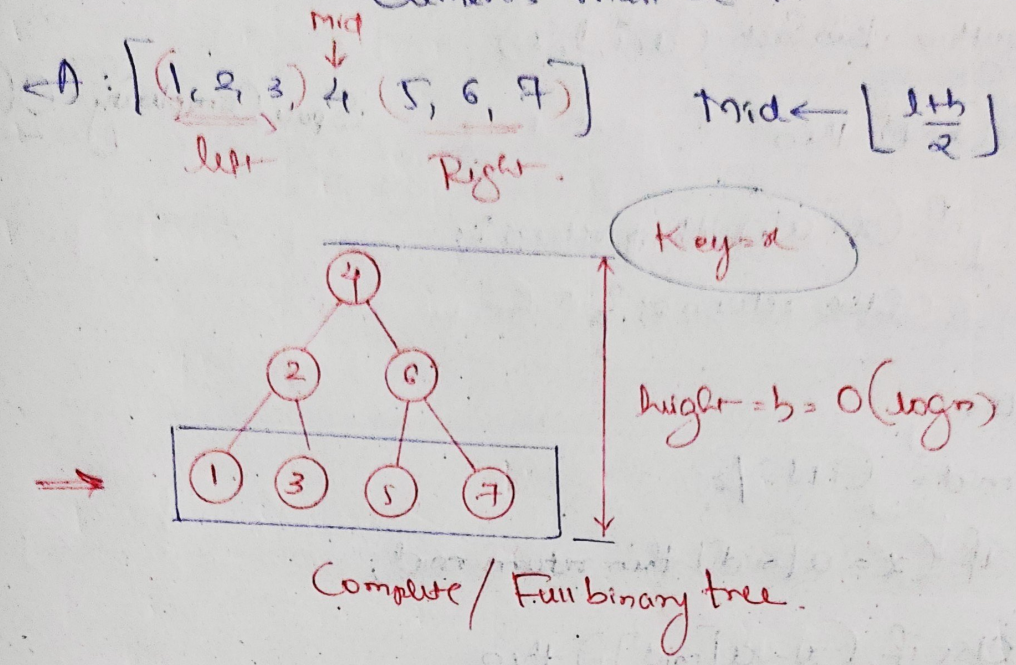
Q3> N items are stored in a sorted doubly linked list. For a delete operation, a pointer is provided to the record to be deleted. For a decrease key operation, a pointer is provided to the record on which the operation is to be performed. An algorithm performs the following operations on the list in this order: $\Theta(N)$ delete, $O(\log N)$ insert, $O(\log N)$ find and $\Theta(N)$ decrease-key. What is the T.C of all these operations combined?

	Total time
1. Delete - N,	$N \times 1$
2. Insert - $\log n$,	$N * \log n$
3. Find - $\log n$,	$N * \log n$
4. Decrease - N,	N^2

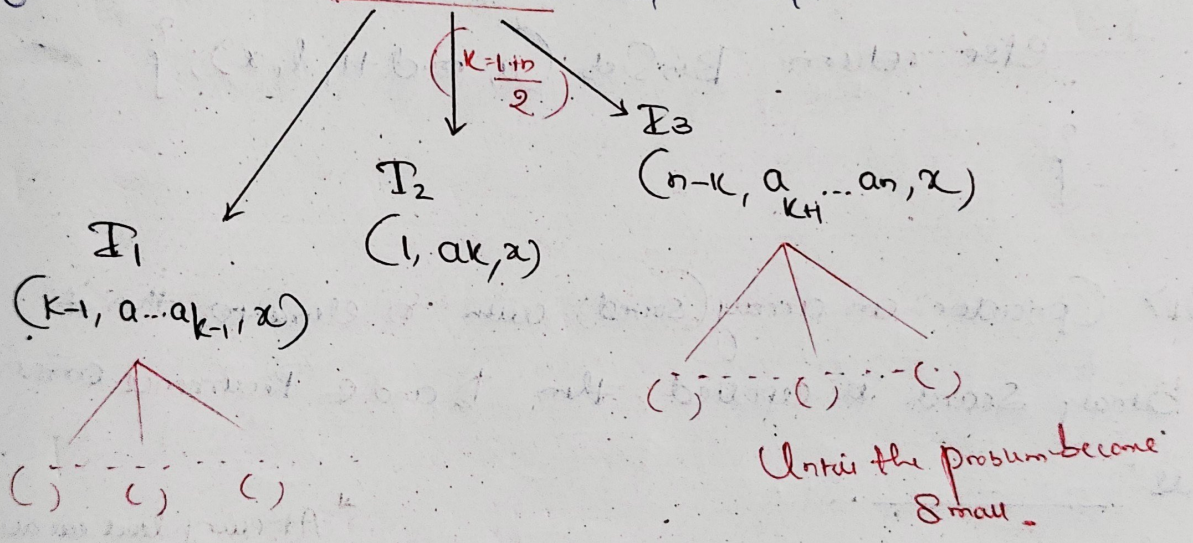
$$(N + N\log n + N\log n + N^2)$$

$$\Rightarrow \underline{\underline{O(n^2)}} \text{ (Overall)}$$

(2) Binary Search : The primary requirement is that the list of elements must be in sorted order.



eg: $I : \langle n, a_1 a_2 \ldots a_n, x \rangle \quad n > 1$



Iterative Binary Search

```
Algorithm BinSearch(a, n, x)
{
    low = 1, high = n;
    while (low ≤ high) do
    {
        mid = (low + high)/2;
        if (x < a[mid]) then high = mid - 1;
        else if (x > a[mid]) then low = mid + 1;
            else return mid; }
    return 0; }
```

Space Complexity = O(1)

Recursive Binary Search

```
Algorithm BinSrch (a, i, l, x)
{  if (l = i) then
   { if (x = a[i]) then return i;
     else return 0; }
   else
   { mid = (i+l)/2;
     if (x = a[mid] then return mid;
     else if (x < a[mid]) then
        return BinSrch (a, i, mid-1, x);
     else return BinSrch (a, mid+1, l, x); }
   }
```

Space Complexity = O(log n)

Q1> Consider an array (sorted) with 'n' elements then if binary search is applied, then D and C Recurrence arising is _____.

* At every level we are solving 2 out of 3 subproblems.

$$T(n) = c, \quad n = 1$$
$$= a + T(n/2), \quad n > 1.$$

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

$$T_C = O(\log n)$$

* In binary search there is no combine operation.

③ Merge Sort : Principle of merging. (conquer)

→ Given two sorted lists $L_1(n_1)$, $L_2(n_2)$ where $n_1 \le n_2$. it is required to merge them into a single sorted having elements, using 2-way merging:

$n_1 = 4$ $n_2 = 6$

$L_1 : <4, 5, 10, 12>$ $L_2 <3, 7, 11, 15, 18, 25>$

↑ i ↑ j

Compare i and j elements

if i is small add to list & increment i.

else if j is small add to list & increment j.

at last after all comparisions, add the remaining ones to list.

$L = <3, 4, 5, 7 \ldots\ldots 15, 18, 25>$

← $n_1 + n_2$ →

Q1) Given two sorted lists $L_1(n_1)$ and $L_2(n_2)$ the min and max no. of comparisions needed to get a single sorted list $(n_1 + n_2)$ is ________

Minimum : $|L_1| <$ First $|L_2|$. Case: $L_1 : <2, 3>$ $L_2 <4, 5>$

Min. comparisions $= n_1$ $[n_1 \le n_2]$

Maximum : $(n_1 - 1) L_1 <$ First $[L_2]$ and all elements of L_2

$|L_2| <$ Last $[L_1]$

$L_1 : <2, 3, 5, 50>$ $L_2 : <8, 10, 12, 15, 18, 25, 40>$

$< 2, 3, 5, 8, 10, 12, 15, 18, 25, 40, 50 >$

Max Comparisions $= (n_1 - 1) + n_2$

→ The number of comparisions required to merge two sorted lists $L_1(n_1)$ and $L_2(n_2)$, $n_1 \leq n_2$ lies between

(n_1) and (n_1+n_2-1)

↓ Min — $O(n_1)$

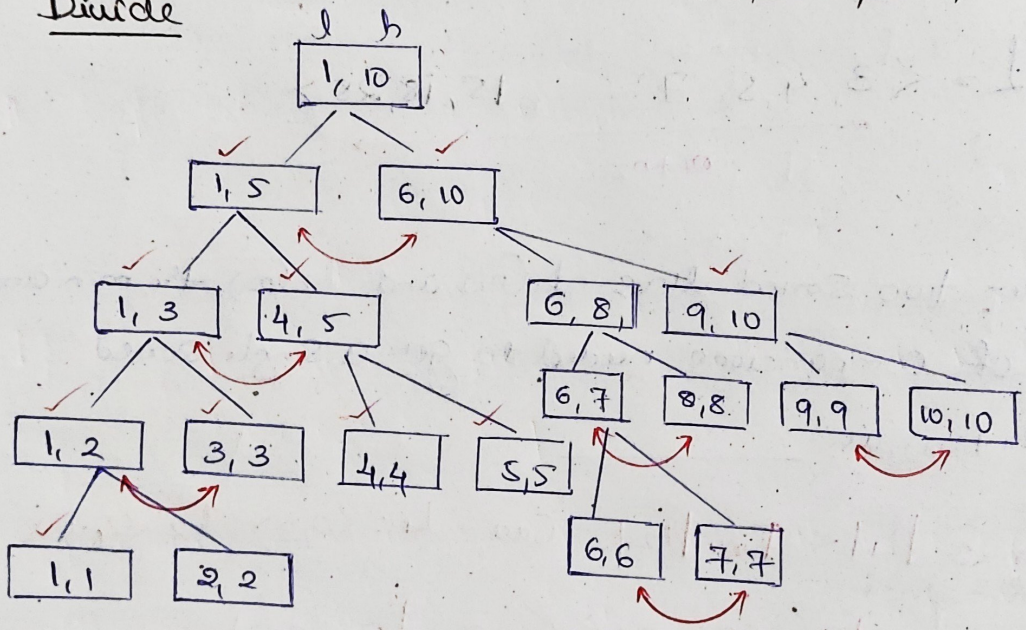
↓ Maximum — $O(n_1+n_2)$

Merge Sort Example

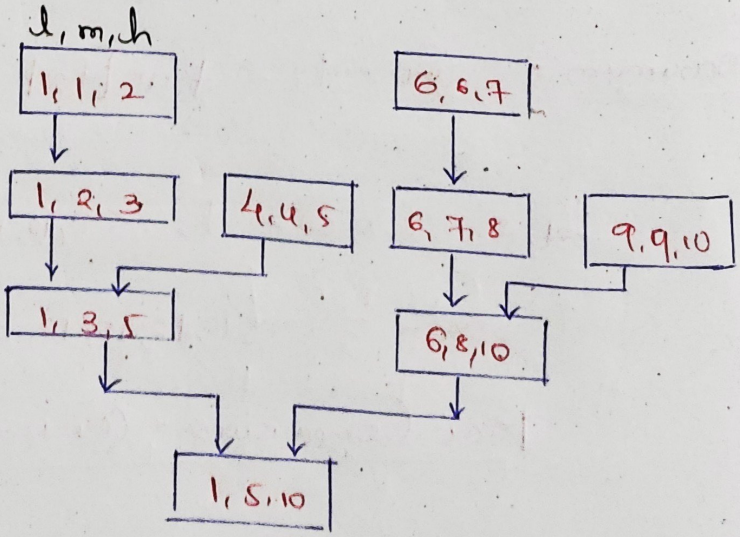
Input A : [(1) 310, (2) 285, (3) 179, (4) 652, (5) 351, (6) 423, (7) 861, (8) 254, (9) 450, (10) 520]

Additional Array : B[n] . B : [179, 254, 285, 310, 351, 423, 450, 520, 652, 861]

Divide



Merge Tree



Algorithm – Merge Sort

```
Algorithm MergeSort (low, high)
{ if (low < high) then
  { mid = ⌊(low+high)/2⌋;
    MergeSort (low, mid);
    MergeSort (mid+1, high);
    Merge (low, mid, high); }
}
```

```
Algorithm Merge (low, mid, high)
{ h = low, i = low, j = mid+1;
  while ((h ≤ mid) and (j < high)) do
  { if (a[h] ≤ a[j]) then
    { b[i] = a[h], h = h+1; }
    else
    { b[i] = a[j], j = j+1; }
    i = i+1; }
  if (h > mid) then
  { for (k = j; k to high) do
    { b[i] = a[k] i = i+1; } }
  else
  { for (k = h to k to mid) do
    { b[i] = a[k], i = i+1; } }
  for (k = low, k to high) do a[k] = b[k];
}
```

↙ Copying elements from array b to A

02/8/23

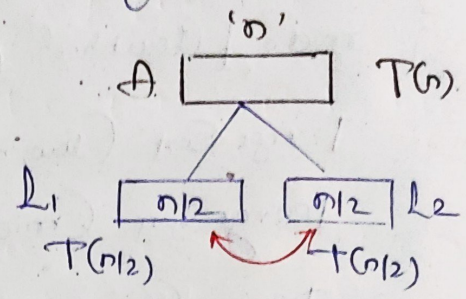
Performance of Merge Sort.

1. Time Complexity:

Let T(n) represents time complexity of D and C merge sort.

$T(n) = c_1, \quad n=1.$

$= \boxed{2T(n/2) + b.n, \quad n \geq 1 \; b > 0}$



Time Complexity:

$= O(n \log n)$

$= \Omega(n \log n)$

$= \Theta(n \log n)$

Merge: Best case → n/2

Worst case → (n-1)

→ O(n)

→ in both case

Space Complexity $= (n + \log n)$

$= O(n)$

Note: * Time wise an algorithm is efficient if the time is bounded by a polynomial.

* Space wise an algorithm is efficient if space requirement is atmost $O(\log n)$ → $O(1)$ is recommended other than recursion algorithms.

↳ for recursion algorithms.

Bottom-up Merge Sort / 2 Way Merge Sort

Array A [310, 285, 179, 652, 351, 423, 861, 254]

$n = 8$

$n = 2^k$

$k = \log n.$

Pass 3: [179, 254, 285, 310, 351, 423, 652, 861] ✓

Pass 2: [179, 285, 310, 652] [254, 351, 423, 861]

Pass 1: [285, 310] [179, 652] [351, 423] [254, 861]

A: [310] [285] [179] [652] [351] [423] [861] [254]

Time Complexity = $n \log n$

Pass 3: [ ]

Pass 2: [x_1, x_2, y_1, y_2] [z_1, z_2]

Pass 1: [y_2, y_1] [x_1, x_2] [z_2, z_1]

[y_1] [y_2] [x_1] [x_2] [z_1] [z_2]

Q.1) Calculate the minimum and maximum no. of element comparisons involved in 2-way merge sort (Assuming $n = 2^k$, $(k > 0)$).

— Also solve for n where n is not power of 2.

Q1) For merging two sorted lists of size m and n into a sorted list of size m+n, we require comparisons of

→ O(m+n)

Q2) If one uses straight two-way merge sort algorithm to sort the following elements in ascending order

[20, 47, 15, 8, 9, 4, 40, 30, 12, 17.]

The order after 3rd pass of the algorithm will be?

→ [20][47][15][8][9][4][40][30][12][17]

Pass 1: [20,47][15][4,9][30,40][12,17]

Pass 2: [8,15,20,47][4,9,30,40][12,17].

∴ The order = 8, 15, 20, 47, 4, 9, 30, 40, 12, 17.

Q3> Assume that merge sort takes 30 secs to sort 64 elements in worst case. What is the approximate no. of elements that can be stored in the worst case using merge sort using 6 minutes?

(worst case ↳ (nlogn) units..

→ n=64 → T( ) = 30 sec (platform dependent)

Apriori: $64 \times \log_2 64$

$= 6 \times 64$ units → 30 sec.

1 unit → ?.

$\frac{30}{6 \times 64}$ s → 1 unit.

$= \frac{360 \times 6 \times 64}{30}$ units ⇒ 'n' elements

$= 4608$ ~~elements~~

units → n elements

$n \log n = 4608$. [Let $n = 2^k$]

$k \cdot 2^k = 4608$.

$9 \times 512 = 4608$. | $k=9$, $n=512$.

④ Quick Sort [Partition-Exchange Sort] : Tony Hoare.

→ Quick Sort is based on partitioning (splitting) process.

→ Partitioning Process [Divide]

1	2	3	4	5	6	7	8	9
A :	65	70	75	80	85	60	55	50	45

Pivot = 65 (select pivot as first elm of array)

(60 55 50 45) [65] (70 75 80 85)

↑ fixed at correct place.

(i) (Find the correct place of pivot in the final sorted list.

(ii) It ensures that all the elements that are less than pivot are placed at left and elements greater than pivot are placed at right of pivot.

Note: Quicksort is the default sorting algorithm in C++ Std library sort function.

```
Algorithm QuickSort(p,q)
{
  if(p<q) then
  { j = partition(a, p, q+1);
  left - QuickSort(p, j-1);
  Right - QuickSort(j+1, q); }
  }

Algorithm Partition(a,m,p)
{ v = a[m]; i=m; j=p;
  repeat
  { repeat i=i+1
    until (a[i]>=v);
```

loop left → Right

```
repeat
    j = j - 1;
until (a[j] ≤ v);
```
loop Right → left

```
if (i < j) then Interchange (a, i, j);
}
until (i ≥ j);
a[m] = a[j]; a[j] = v; return j;
}
```

Time Complexity of Partition = O(n)

```
Algorithm Interchange (a, i, j)
{ p = a[i];
  a[i] = a[j]; a[j] = p; }
```

~~Partition~~

Performance of QuickSort

Recursive Stack = (log n)

I : <n, a1, a2, ... an>

Partition : O(n)

Best Case

I1 : <n, (n/2)L [a1] (n/2)R>

2nd is (n/2)L [a1] (n/2)R

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

= O(n log n)

I2 : <(n-1)L [a1]>

(n-2) is decreasing order

I3 : <[a1] (n-1)R>

Increasing order

Worst Case

If elements are in sorted order:

→ $T(n) = O(n) + T(n-1) \Rightarrow$ **$O(n^2)$ Time Complexity.**

Note: ↓ QuickSort behaves in worst case when elements are in sorted order.

↓ It performs the best when the elements are unsorted.

Space Complexity: Best Case: $O(\log n)$

Worst Case: $O(n)$

eg: Array A:

1	2	3	4	5	6	7	8	9	10
65	70	75	80	85	60	55	50	45	∞

(added at last for a purpose)

Pivot. V ← 65

i = initially

j initially

i	j
1	10
2	9
3	8
4	7
5	6
6	5

2 loops. Elementary Comparisons.

After completion =

After partition:

1	2	3	4	5	6	7	8	9	10
60	45	50	45	65	85	80	75	70	∞

left Right

j pivot found

— the left & right again go further for sorting process until all sorted.

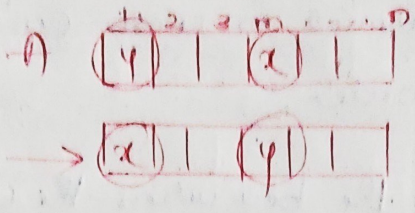
" 10 "

↓ ∞

— Implementation of QuickSort requires an additional element (n+1) initialized to '∞' (infinity) → to avoid an infinite looping of first loop (1).

↳ Happens when first element of left side is higher than elements at right side causing infinite looping.

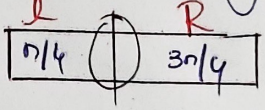
Q1> In using Quick Sort suppose the central element of the Array is always choosen as the pivot then worst case complexity of Quicksort may be $O(n^2)$



Q2> The median on Array of size n can be fand in O(n) time. If median is selected as pivot, then the worst case complexity of quick sort ____

$$T(n) = \underbrace{O(n)}_{\text{Median}} + \underbrace{O(n)}_{\text{Paitationing}} + 2T(n/2) \Rightarrow \boxed{O(n\log n)}$$

Q3> In applying Quicksort to an unsorted list if (n/4) the element is selected as pivot then the time complexity of Quicksort will be ____



$$T(n) = O(n) + O(n) + T(n/4) + T(3n/4)$$

$$\Rightarrow \boxed{O(n\log n)}$$

Q4> Consider the quicksort algorithm. Suppose there is a procedure for finding a pivot element which splits the list into two ~~Subjects~~ Sub lists each of which contains at least one-fifth of the elements. Let T(n) be the number of comparisions required to sort n elements. Then

$$\rightarrow T(n) = O(n) + T(n/5) + T(4n/5)$$

03/8/2022

Q5) Let P be a quicksort program to sort numbers in ascending order. Let t_1 and t_2 be the time taken by the program for the inputs [1 2 3 4] and [5 4 3 2 1] respectively. Which of the following holds?

→ $t_1 < t_2$

Q6) Let P be a quicksort program to sort numbers in ascending order using the first element as pivot. Let t_1 and t_2 be the no. of comparisons made by P for the inputs {1,2,3,4,5} and {4,1,5,3,2} respectively. Which of the following holds? (Best case, worst case)

→ $t_1 > t_2$

Q7) Quick sort is run on two inputs shown below to sort in ascending order taking first element as pivot.

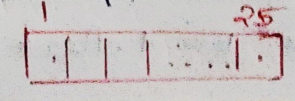
(i) 1, 2, 3, ... n

(ii) n, n-1, n-2, ... 2, 1.

Let C_1 and C_2 be the number of comparisions made for the inputs (i) and (ii) respectively then,

→ $C_1 = C_2$.

Q8) An array of 25 distinct elements is to be sorted using quicksort. Assume that the pivot element is chosen uniformly at random. The probability that element gets placed in the worst possible locations in the first round of partitioning (rounded off to 2 decimal position) is 0.08.

→ 

$= \frac{1}{25} + \frac{1}{25} = \frac{2}{50} \Rightarrow 0.04$

⑤ Matrix Multiplication:

$A_{m \times n}$, $B_{n \times n}$, $C_{m \times n}$.

$A \pm B = C$.

$$A\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix} + \begin{bmatrix}5 & 6\\ 7 & 8\end{bmatrix} \quad (B)$$

$$C = \begin{bmatrix}6 & 8\\ 10 & 12\end{bmatrix}$$

```
for i ← 1 to n                    O(n²)
    for j ← 1 to n
        C[i][j] = A[i][j] ± B[i][j]
```

$A_{(2,2)} \times B_{(2\times2)} = C_{2\times2}$.

$$A\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix} \quad B\begin{bmatrix}5 & 6\\ 7 & 8\end{bmatrix} \quad C\begin{bmatrix}C_{11} & C_{12}\\ C_{21} & C_{22}\end{bmatrix}$$

$C_{11} = a_{11} * b_{11} + a_{12} \times b_{21}$

$C_{12} = a_{11} * b_{12} + a_{12} \times b_{21}$

$C_{21} = a_{21} \times b_{11} + a_{22} * b_{21}$

$C_{22} = a_{21} * b_{12} + a_{22} * b_{22}$

Algorithm multiply

```
{
  for i ← 1 to n:
    for j ← 1 to n:                 O(n³)
      C[i,j] = 0;
      for K ← 1 to n
        C[i,K] = A[i,k] * B[k,i] }
```

$$A = \left[\begin{array}{cc|cc} 1 & 2 & 3 & 4\\ 5 & 6 & 8 & 2\\ \hline 1 & 3 & 5 & 7\\ 9 & 1 & 2 & 5 \end{array}\right]_{4\times4} \quad (A_{11}, A_{12}, A_{21}, A_{22})$$

$$B = \left[\begin{array}{cc|cc} 5 & 6 & 7 & 2\\ 9 & 8 & 1 & 9\\ \hline 6 & 5 & 4 & 2\\ 3 & 1 & 5 & 6 \end{array}\right]_{4\times4} \quad (B_{11}, B_{12}, B_{21}, B_{22})$$

$$\Rightarrow C \left[\begin{array}{c|c} C_{11} & C_{12}\\ \hline C_{21} & C_{22} \end{array}\right]_{4\times4}$$

Sub matrix multiplication & Sub matrix addition

$C_{11} = (A_{11} * B_{11}) + (A_{12} * B_{21})$ — $T(n/2)$, $T(n/2)$

$C_{12} = A_{11} * B_{12} + A_{12} * B_{22}$

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

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

8×8 → 4×4 → 2×2 → 1×1.

→ Let $T(n)$ represents time complexity to multiply two Square matrices A and B of order $n \times n$;

$$T(n) = c, \quad n \le 2$$
$$= 8T(n/2) + bn^2, \quad n > 2, \ b > 0.$$

$$T(n) = 8T(n/2) + bn^2 \quad (1)$$
$$T(n/2) = 8T(n/4) + bn^2/4 \quad (2)$$
$$T(n) = 8[8T(n/4) + bn^2/4] + bn^2$$
$$= 64 \cdot T(n/4) + 3bn^2 \quad (3)$$
$$= 8^2 T(n/2^2) + (2^2 - 1)bn^2 \quad (4)$$
$$= 8^k \cdot T(n/2^k) + (2^k - 1)bn^2 \quad (5)$$

$$\frac{n}{2^k} = 1 \Rightarrow n = 2^k$$
$$k = \log_2 n.$$

$$T(n) = 8^{\log_2 n} \cdot c + (n-1)bn^2$$
$$= n^3 \cdot c + bn^3 - bn^2 \quad (6)$$
$$= cn^3 + bn^3 - bn^2 \Rightarrow O(n^3)$$

T and C using Divide and Conquer = $O(n^3)$

→ In D and C, there are presently 8 – Sub matrix multiplications involved in eqns $C_{11}, \ldots C_{22}$.

→ Time Complexity will be reduced only if no. of submatrix multiplications are reduced from 8 to a lower value.

→ A, B, C : $n \times n$.

A_{ij}, B_{ij}, C_{ij} : $\frac{n}{2} * \frac{n}{2}$.

P, Q, R, S, T, U, V : $\frac{n}{2} * \frac{n}{2}$ (additional matrices).

$T(n)$ represents time complexity of STRAS_DANDC$(n \times n)$

$$T(n) = c, \quad n \le 2$$
$$= 7 \cdot T(n/2) + bn^2, \quad n > 2.$$

$$P = (A_{11} + A_{22})(B_{11} + B_{22})$$
$$Q = (A_{21} + A_{22})B_{11}$$
$$R = A_{11}(B_{12} - B_{22})$$
$$S = A_{22}(B_{21} - B_{11})$$
$$T = (A_{11} + A_{12})B_{22}$$
$$U = (A_{21} - A_{11})(B_{11} + B_{12})$$
$$V = (A_{12} - A_{22})(B_{21} + B_{22})$$

$$C_{11} = P + S - T + V$$
$$C_{12} = R + T$$
$$C_{21} = Q + S$$
$$C_{22} = P + R - Q + U.$$

$T(n) = 7T(n/2) + bn^2$ — (1)

$T(n/2) = 7T(n/4) + bn^2/4$ — (2)

$T(n) = 7(7 \cdot T(n/4) + bn^2/4) + bn^2$ — (3)

$= 49\,T(n/4) + (7/4)bn^2 + \left(\frac{7}{4}\right)^0 bn^2$ — (4)

$= 7^2 \cdot T(n/2^2) + bn^2 \sum_{i=0}^{1} (7/4)^i$

$\vdots$

$= 7^k \cdot T(n/2^k) + bn^2 \sum_{i=0}^{k-1} (7/4)^i$

$\sum_{i=1}^{n} x^i < x^{n+1} \quad x > 1$

$T(n) < 7^k \cdot c + bn^2 \left(\frac{7}{4}\right)^k$

$< c \cdot 7^k + bn^2 \cdot \frac{7^k}{4^k}$

$\frac{n}{2^k} = 1$

$n = 2^k$

$k = \log n$

$T(n) < d \cdot 7^k < d \cdot 7^{\log n}$

$\Rightarrow n^{\log_2 7}$

Time Complexity → $T(n) < 2n^{2.81}$

Space Complexity:

1. School method — $O(1)$
2. D and C method — $O(\log n)$
3. Strassen's method — $\log n + n^2 \Rightarrow O(n^2)$

↳ either Optimizing time we need to Sacrifice Space

Master Theorem (method) for solving D and C Recurrences

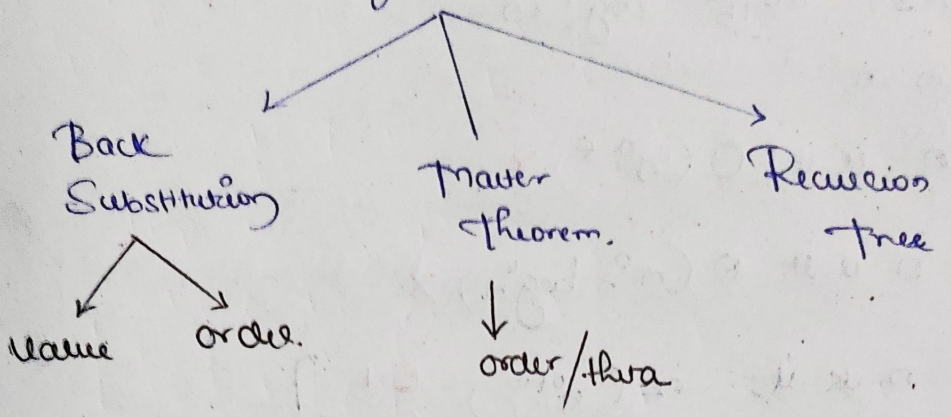
$T(n) = aT(n/b) + f(n), \quad n > d$, $\quad a \geq 1,\ b > 1,\ f(n)$ is true.

$= c, \quad n \leq d.$

Case I: If $f(n)$ is $O(n^{\log_b a - \epsilon})$ for some $\epsilon > 0$, then

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

Solving D and C Recurrences



Case II: If $f(n)$ is $\Theta(n^{\log_b a} * \log^k n)$ for some k, such that

(a) $k > 0$, then $T(n)$ is $\Theta(n^{\log_b a} * \log^{k+1} n)$

(b) $k = -1$, then $T(n)$ is $\Theta(n^{\log_b a} * \log\log n)$

Case III: If $f(n)$ is $\Omega(n^{\log_b a + \epsilon})$ for some $\epsilon > 0$, and

$a \cdot f(n/b) \leq \delta \cdot f(n)$ for $\delta < 1$, then

$T(n)$ is $\Theta(f(n))$

1) $T(n) = 4 \cdot T(n/2) + n$

$a = 4$
$b = 2$
$f(n) = n$

$\log_b a = \log_2 4 = 2$

Case I: n is it $O(n^{2-\epsilon})$ $\epsilon = 1$, $\epsilon = 0.5$

$n = O(n)$.

$\therefore$ $T(n)$ is $\Theta(n^2)$ ✓

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

$a = 2$
$b = 2$
$f(n) = n \cdot \log n$

$\log_2^2 = 1$.

Case I: $n \cdot \log n$ is it $\Theta(n^{1-\epsilon})$ ×

Case II: $n \cdot \log n$ is it $\Theta(n^1 \log^k n)$

$k = 1, 0$

$\therefore$ $T(n)$ is $\Theta(n \cdot \log^2 n)$

3) $T(n) = T(n/3) + n$.

$a=1$, $b=3$, $\log_b a = \log_3 1 = 0$.

$f(n) = n$.

Case I: n is it $O(n^{0-\epsilon})$ ×

Case II: n is it $\Theta(n^0 \cdot \log^k n)$ ×

Case III: n is it $\Omega(n^{0+\epsilon})$ $\epsilon = 1$, $\epsilon = 0.5$

$a \cdot f(n/b) \le \delta \cdot f(n)$ for $\delta < 1$

$1 \cdot \frac{n}{3} \le \delta \cdot n$. $\delta = \frac{1}{3} < 1$

$\therefore T(n) = \Theta(n)$

4) $T(n) = 9T(n/3) + n^{2.5}$

$a=9$, $b=3$, $f(n) = n^{2.5}$ $= f(n/3) = (n/3)^{2.5} = \left(\frac{n^{2.5}}{9\sqrt{3}}\right)$

$\log_3 9 = 2$.

Case I: $n^{2.5}$ is it $O(n^{2-\epsilon})$ ×

Case II: $n^{2.5}$ is it $\Theta(n^2 \log^k n)$ ×
$(n^2\sqrt{n})$

Case III: $n^{2.5}$ is it $\Omega(n^{2+\epsilon})$ $\epsilon = 0.5$

$a \cdot f(n/b) \le \delta \cdot f(n)$

$9 \cdot \frac{n^{2.5}}{9\sqrt{3}} \le \delta \cdot n^{2.5}$ for $\delta = \frac{1}{\sqrt{3}} < 1$ $= \Theta(n^{2.5})$

$\therefore T(n) = \Theta(n^{2.5})$

1. Max-Min:

$T(n) = 2T(n/2) + 2 \rightarrow \left(\frac{3n}{2} - 2\right) = O(n)$

$a=2$, $b=2$, $f(n)=c$

Case I: c is it $O(n^{1-\epsilon})$ $\epsilon = 1$

$\therefore T(n)$ is $\Theta(n^1)$

② Merge Sort:

$T(n) = 2 \cdot T(n/2) + n$ | $a=2,\ b=2,\ f(n)=n$, $\log_2 2 = 1$.

Case I: n is it $O(n^{1-\epsilon})$ $\epsilon > 0$ ✗

Case II: n is it $\Theta(n \cdot \log^k n)$ $k=0$ ✓

$\therefore T(n)$ is $\Theta(n \cdot \log n)$

③ Matrix Multiplication

(a) D and C: $T(n) = 8T(n/2) + n^2$

Case I: n^2 is it $O(n^{3-\epsilon})$ $\epsilon = 1$ ✓

$\therefore T(n) = \Theta(n^3)$

(b) Strassen: $T(n) = 7T(n/2) + n^2$. $\{ \log_2 7 = 2.81$.

Case I: n^2 is it $O(n^{2.81-\epsilon})$ $\epsilon = 0.81$

$\therefore T(n) = \Theta(n^{2.81})$.

④ Binary Search: $T(n) = T(n/2) + c$

$a=1,\ b=2,\ f(n)=c,\ \log_2^1 = 0$

Case I: C is it $O(n^{0-\epsilon})$ ✗

Case II: C is it $\Theta(n^0 \cdot \log^k n)$, $k=0$

$\therefore T(n)$ is $\Theta(\log n)$

Homework

1. $T(n) = 3T(n/2) + n$
2. $T(n) = 16 \cdot T(n/4) + n$
3. $T(n) = 4T(n/2) + \log n$
4. $T(n) = \sqrt{2} \cdot T(n/2) + \log n$
5. $T(n) = 6 \cdot T(n/3) + n^2 \log n$
6. $T(n) = 6 \cdot T(n/3) + n^2 \log n$
7. $T(n) = 2T(n/2) + \frac{n}{\log n}$.
8. $T(n) = 4T(n/2) + n^2$
9. $T(n) = 2T(n/2) + \sqrt{n}$
10. $T(n) = 3T(n/3) + n$
11. $T(n) = 2^2 \cdot T(n/4) + n$
12. $T(n) = 2 \cdot T(\sqrt{n}) + \log n$.

5/8/2023

Q1> The running time of an algorithm is represented by the following recurrence relation:

$$T(n) = \begin{cases} T\left(\frac{n}{3}\right) + cn & n \le 3 \\ \text{Otherwise.} \end{cases}$$

Which of the following represents the time complexity of the algorithm:

→ $\Theta(n)$

Q2> Which of the following correctly determined the solution of the recurrence relation with $T(1)=1$?

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

→ $a=2, b=2$ $\log n$ is $O(n^{1-\epsilon})$ $\epsilon = 0.5$

$O(\sqrt{n})$

$\Rightarrow \Theta(n)$.

Q3> For constants $a \ge 1$ and $b > 1$ consider the following recurrence defined on the non-negative integers:

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

Which of the following is correct about recurrence $T(n)$?

→ If $f(n)$ is $O(n^{\log_b(a)-\epsilon})$ for some $\epsilon > 0$, then $T(n)$ is $\Theta(n^{\log_b(a)})$

Q4> For parameters a and b, both of which are $\omega(1)$, $T(n) = T(n^{1/3}) + 1$, and $T(b) = 1$. Then $T(n)$ is

→ $T(n) = T(n^{1/a}) + 1$ — ①

$T(n^{1/a}) = T(n^{1/a^2}) + 1$ — ②

$T(n) = T(n^{1/a^2}) + 2$ — ③

$= T(n^{1/a^k}) + k$ — ④

$n^{1/a^k} = b$

$\frac{1}{a^k} \cdot \log_2 n = \log_2 b$

$a^k = \frac{\log_2 n}{\log_2 b} = \log_b n$

$a^k = \log_b n$

$k \log_2 a = \log_2(\log_b n)$

$$k = \frac{\log_2(\log_b n)}{\log_2 a} = \log_a \log_b n$$

$\Rightarrow T(1) + (\log_a \log_b n)$

$\therefore$ Time Complexity $= \Theta(\log_a \log_b n)$

Homework

Q5) Consider the following recurrence relation

$T(1) = 1$

$T(n+1) = T(n) + \lfloor \sqrt{n+1} \rfloor$ for all $n \geq 1$.

The value of $T(m^2)$ for $m \geq 1$ is _____

Q6) When $n = 2^{2^k}$ for some $k \geq 0$, the recurrence relation

$T(n) = \sqrt{(2)}\, T(n/2) + \sqrt{n}$, $T(1) = 1$.

Evaluate to: $\sqrt{(n)}(\log n + 1)$

By Masters theorem: $a = \sqrt{2}$, $b = 2$ $\log_2 \sqrt{2} = 1/2$

Case I: $\sqrt{n}$ is it $O(n^{1/2 - \epsilon})$ ✗

Case II: $\sqrt{n}$ is it $\Theta(n^{1/2} \cdot \log^k n)$

$\therefore T(n)$ is $\Theta(\sqrt{n} \cdot \log n)$ ✓ $k = 0$

① $T(n) = (2^n)\, T(n/2) + n$ → a not constant

② $T(n) = 0.5\, T(n/2) + n^2$ → $a \geq 1$

} In admissible Case cannot be solved by masters theorem

① $T(n) = 2T(\sqrt{n}) + \log n$ | $b=1, b>1$

Changing the variables:

Let $n = 2^k$, $k = \log n$

$T(2^k) = 2 \cdot T(2^{k/2}) + k$ — ①

Let $T(2^k) = S(k)$

$T(2^{k/2}) = S(k/2)$

$S(k) = 2 \cdot S(k/2) + k$ — ③

$\Theta(k \cdot \log k) \Rightarrow \Theta(\log n * \log\log n)$

$T(n) = 2T(n/2) + n$ → $\Theta(n \cdot \log n)$

② $T(n) = T(\sqrt{n}) + 1$

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

Let $T(2^k) = S(k)$

$S(k) = S(k/2) + 1$ — ②

$\Theta(\log k) \Rightarrow \Theta(\log\log n)$

$T(n) = T(n/2) + 1$ → $\Theta(\log n)$

+ve

③ $T(n) = 2T(\sqrt{n}) + 1$

Homework.

Q7) An algorithm performs $(\log N)^{1/2}$ find operations, N-insert operations, $(\log N)^{1/2}$ delete operations and $(\log N)^{1/2}$ decrease key operations on a set of data items with keys drawn from a linearly ordered set. For a delete operation, a pointer is provided to the record that must be deleted. For the decrease key operation a pointer is provided to the record that has its key decreased. Which one of the following data structure is the most suited if goal is to achieve best total asymptotic complexity considering all operations?

⑥ Long-Integer Multiplication. (LIM)

→ Given two integers, say 'u' and 'v'

int (u, v) → regular integers
a) u+v; } O(1)
b) u*v; }

↓
Max: 32767

→ long int u, v
→ 4B/8B → can store 8/10 digit number
not more than that.

→ we cannot store more longer data / no hardware solution.

Software Solution: We can store digits of long integer in an array

	0	1	2	3	4	5	6	7	8	9	10
→ u:	1	2	3	4	5	6	7	8	9	7	1
v:	8	7	6	4	3	2	1	5	2	9	1

⇒ int v[n], u[n], C[n];

a) u ± v ⇒ O(1) ?
⇒ NO.

```
{ for i ← 1 to n
      C[i] = v[i] ± u[i] }
```
→ O(n), → with possible carry.

b) u*v = C;
⇒ $O(n^2)$

```
{ for i ← 0 to n
     for j ← 0 to n
        .....
        ..... }
```

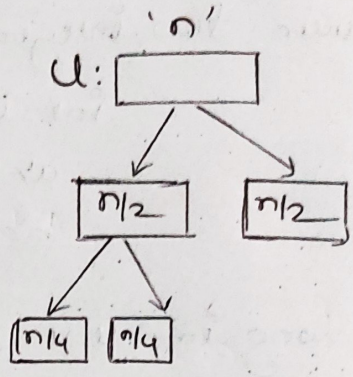
D and C method for Long integer multiplication

→ u, v : Long integers of 'n' digits each.

u = 1234 ; v = 5678

= $12 \times 10^2 + 34$

$1 \times 10^1 + 2$ $\quad$ $3 \times 10^1 + 4$



w, x, y, z = n/2 digit numbers

Let $m = \lfloor n/2 \rfloor$

$u = (w \times 10^m + x)$; $v = (y \times 10^m + z)$

$(u.v) = (w \times 10^m + x) \cdot (y \times 10^m + z)$

$= \underbrace{w.y \times 10^{2m}}_{(n/2)} + \underbrace{[(w.z) + (x.y)]}_{(n/2)} \times 10^m + \underbrace{x.z}_{(n/2)}$ ——— (1)

$w = u / 10^m$; $x = u \% 10^m$

$y = v / 10^m$; $z = v \% 10^m$

Let T(n) represents time complexity of multiplying two long ints of n-digits each.

$(u.v)_n = (w.y)_{n/2} \times 10^{2m} + (w.z + xy)_{n/2} \times 10^m + (x.z)_{n/2}$ ——— 1.

$T(n) = 4.T(n/2) + b.n, \quad n > 1$ ——— (1)

$= c, \quad n = 1$ ——— (2)

$\Rightarrow T(n) = O(n^2)$

(no improvement in D and C approach instead space complexity = O(logn))

↓ reducing multiplications.

Anatolye Karatsuba Optimizations

Let t = n/2 digit no.

$t = (w + x) \cdot (y + z)$

$= wy + [wz + xy] + xz$

$(w.z + xy) = t - (w.y + xz)$.

→ let w.y = Product 1.

x.2 = Product 2

t = product 3.

- A.K Optimization:

$$(u.v)_n = \text{Product 1} \times 10^{2m} + \text{Product 3} - (\text{Product 1} + \text{Product 2}) \times 10^{m} + \text{Product 2} \quad \text{--- (2)}$$

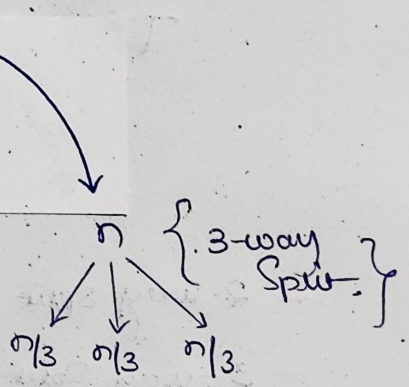
For A.K Optimization:

$$T(n) = 3.T(n/2) + b.n, \quad n > 1$$
$$= c \quad : n = 1$$

Case I: $\Theta(n^{\log_2 3}) = \Theta(n^{1.58})$ ✓

Toom and Cook Optimization:

→ 3-way Split.



$(u.v)_n = 9(x.y)_{n/3}$ — ①

1) $T(n) = 9T(n/3) + n$ — ② [D and C]

↳ $\Theta(n^2)$

2) A.K optimization : $8T(n/3) + n$ — ③

↳ $\Theta(n^{\log_3 8}) = \Theta(n^{1.89})$

$\therefore x.T(n/3) + n < \Theta(n^{1.58})$

↳ $\Theta(\log_3 x)$

Q) What should be value of x?

→ $\log_3 x < 1.58$

$x < 3^{1.58} < 3\sqrt{3} = 5$

if $(x == 5)$ → then Time Complexity will be better.

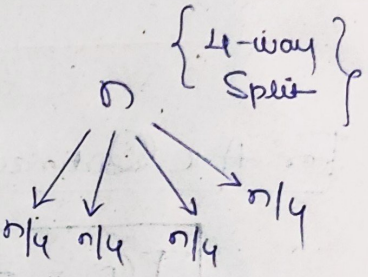
: They solved system of linear equations to get

$(a,v)_n = (w,x)_{n/3}$ cuts 5.

$T(n) = 5.T(n/3) + n$ — ①

↳ $\Theta(n^{\log_3 5}) \Rightarrow \Theta(n^{1.46})$

4-way Split



1) D and C : $T(n) = 16.T(n/4) + n \Rightarrow \Theta(n^2)$

2) A.K : $T(n) = 15.T(n/4) + n \Rightarrow \Theta(n^{1.95})$

3) T.C : $T(n) = x.T(n/4) + n \Rightarrow \Theta(n^{\log_4 x}) < \Theta(n^{1.46})$

$T(n) = 7T(n/4) + n$ $\quad x = 4^{1.4c} \sim 7$

↳ $\Theta(n^{\log_4 7}) = \Theta(n^{1.40})$

- 2-way split, D and C : 4, A.K = 3, TC = 3.
- 3-way split, D and C : 9, AK : 8, T.C : 5
- 4-way split, : D and C : 16. AK : 15. T.C : 7

→ K-way split, D and C : K^2, AK : (K^2-1), TC. $(2K-1)$

Generalized equations of time complexities for K-way split.

① D and C : $T(n) = K^2 T(n/K) + bn$.

② AK : $T(n) = (K^2-1).T(n/K) + bn$.

③ TC : $T(n) = (2K-1).T(n/K) + bn$

07/8/2023

Greedy Method (Gm.)

- Greedy method is a design strategy used for solving problems, where solutions are viewed (seen) as a result of making a set of decisions.
- These set of decisions based on greedy method are made in a step-wise manner (step-by-step) manner.
- At each step, out of available options, greedily select that option that satisfies the given criteria/conditions objectively.

Terminology

→ Problem definition

→ Constraints (conditions/requirements) → Implicit (local to problem)
→ Explicit (Boundary Conditions)

eg:

① N-Queens Problem } example for constraints

n-queens $(q_1, q_2 \ldots q_n)$

$n = 4$

	1	2	3	4
q_1	×	q	×	×
q_2	×	×	×	q
q_3	q	×	×	×
q_4	×	×	q	×

int x[1..n];

x[i] = position (column) in which Queen will be placed.

x Array: 2,4,1,3 <2,4,1,3>

positions

$1 \le x[i] \le n$. (4) in this case.

→ Solution Space: All possible ways of organising inputs, satisfying only explicit constraints

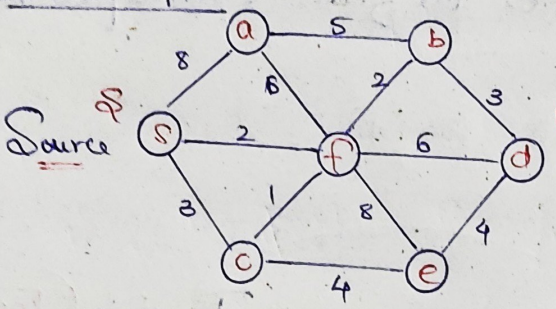
- for n-queen problem, the size of solution space = n!
- All problems will have its solution space as a function of input size.

→ Feasible Solution: feasible solutions are those solutions in the solution space that satisfy implicit constraints of the problem.

* Problem may have multiple feasible solutions.

→ Objective Function: Few problems may have objective function which tries to minimize/maximize a given iteration of the problem.

eg: Shortest Path.



Source

Destination.

Objective function: To minimize the cost of path.

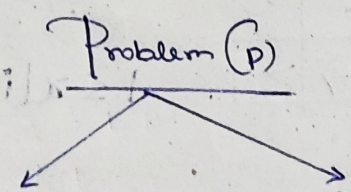
Solution = ?

→ Optimal Solution: It is that feasible solution which satisfies the objective function.

* Optimal solution always refers to the value and hence it is always unique.

Q.) What is the objective function of N-queens problem?

→ No objective function.



→ Its result is always either yes/no. (T/F) or (0/1)

eg: Sorting an array, n queens

→ requires you to determine a max/min value of a given criterion. (Optimal Solution/Objective function).

eg: Shortest paths

Tell whether they are decision / optimisation problem.

1. CPU Scheduling → Optimization
2. Page replacement → Optimization
3. Disk scheduling → Optimization
4. Banker's Safety Algorithm → decision
5. GATE 2024 Problem → ?
6. Congestion Control. → Optimization

Control Abstraction / General Method

```
Algorithm Greedy (a, n)
{   Solution = 0;
    for i = 1 to n do
    {   x = select(a);
        if feasible (solution, x) then
            solution = union (solution, x); }
    } return solution;
}
```

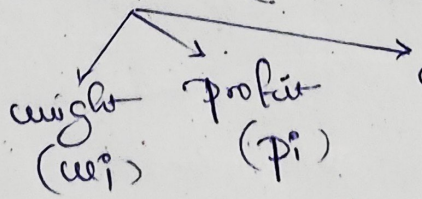
Time Complexity can't be always greater or equal to O(n)

if select operation is O(1)

→ No greedy algorithm can take less than O(n) time.

Problem ① : Knapsack Problem.

- Given a knapsack of capacity 'm' (bag).
- Given n objects (o1, o2, o3 .... on)



- If object of weight (wi) is put into the knap, then knap gets filled up by weight of wi & we get profit "pi".
- Objective is to maximize the profit, each knap fill with weight.
- Maximise the profit subject to the condition that the total weight being put into the knapsack does not exceed its capacity.

Knapsack problem

- **Fractional / Real / Continuous**
  - $0 \le x_i \le 1$
  - $\langle \infty$ values $\rangle$
- **Binary $\langle 0/1 \rangle$**
  - either include it totally or exclude it.
  - $\langle x_i = 0/1 \rangle$

1) $\sum_{i=1}^{n} w_i \le m \Longrightarrow$ Total profit $= \sum_{i=1}^{n} p_i$

- $\sum_{i=1}^{n} w_i$ → Summation of all weights.
- $x_i = 1$ → $\langle$no-decision$\rangle$ (Trivial)

---

2) For solving the problem, this

$$\sum_{i=1}^{n} w_i > m$$

- Explicit Constraint.
- if sum of all weights > Cont. Cond.
- we need to decide the things to add.

$$\text{Max} \sum_{i=1}^{n} p_i * x_i \rightarrow \text{Objective function}$$

$$\text{Subject to } \sum_{i=0}^{n} w_i * x_i \le m, \text{ where } 0 \le x_i \le 1$$

- $\sum_{i=0}^{n} w_i * x_i \le m$ → Implicit Constraint.
- $0 \le x_i \le 1$ → Explicit Constraint.

Solution Space

- (Greedy m) Fractional " ∞ "
- (D.P) 0/1 KNAP " 2^n "

eg: $n=3$, $m=20$; $\langle w_1, w_2, w_3 \rangle = \langle 18, 15, 10 \rangle$

$\langle p_1, p_2, p_3 \rangle = \langle 25, 24, 15 \rangle$

} Fractional knap.

8/8/2023

a) Greedy about profit:

$x_1 = 1$
$x_2 = 2/15$
$x_3 = 0$

$\sum w_i x_i = 20$

$\sum p_i x_i = \left(25 + \frac{2}{15} * 24 + 0\right)$

$= 28.3$

b) Greedy about weight:

$x_3 = 1$
$x_2 = 10/15 = 2/3$
$x_4 = 0$

$\sum w_i x_i = 20$

$\sum p_i x_i = 31$

Optimal Selection $\Rightarrow$ { $w_i \rightarrow p_i$; $1 \rightarrow ?$ } (P_i/w_i) (Select higher ratio)

c) Greedy about Profit per unit:

$\frac{P_1}{w_1} = \frac{25}{18} = 1.38 \quad \therefore x_1 = 0$

$\frac{P_2}{w_2} = \frac{24}{15} = 1.6 \quad \therefore x_2 = 1$

$\frac{P_3}{w_3} = \frac{15}{10} = 1.5 \quad \therefore x_3 = 5/10 = 1/2$

$\sum w_i x_i = 20;$

$\sum p_i x_i = 0 + 24 + \frac{1}{2} * 15$

$= 31.5$

Q1) Find an optimal solution to the knapsack instance $n=7$, $m=15$, $(p_1, p_2 \ldots p_7) = (10, 5, 15, 7, 6, 18, 3)$, and weights $= (2, 3, 5, 7, 1, 4, 1)$.

$\frac{P_1}{w_1} = \frac{10}{2} = 5$

$\frac{P_2}{w_2} = \frac{5}{3} = 1.66$

$\frac{P_3}{w_3} = \frac{15}{3} = 3$

$\frac{P_4}{w_4} = \frac{7}{7} = 1$

$\frac{P_5}{w_5} = \frac{6}{1} = 6$

$\frac{P_6}{w_6} = \frac{18}{4} = 4.25$

$\frac{P_7}{w_7} = \frac{3}{1} = 3$

$m = 15$

$\langle 1 + 2 + 4 + 5 + 1 + \frac{2}{3} * 6 \rangle$

$x_5 = 1$
$x_1 = 1$
$x_6 = 1$
$x_3 = 1$
$x_7 = 1$
$x_2 = 2/3$

= profit $\Rightarrow 55.33$

```
Procedure Greedy_Knapsack(p, w, m, x, n)
{ // Objects are ordered.
  real p(1:n), w(1:n), x(1:n), m, cu;
  integer i, n;
  x ← 0   — initialize Solution to zero
  cu ← m.   cu = remaining knapsack Capacity
  for i ← 1 to n do:
      if w(i) > cu then exit endif
      x(i) ← 1   ← adding the object to Solution vector.
      cu ← cu − w(i)
  repeat.
      if i ≤ n then x(i) ← cu/w(i) endif
  end Greedy_Knapsack.
```

p → profits, w → weights, m → Knapsack Size, x → Solution vector

Time Complexity = O(n) — if Sorting is Considered then ⇒ O(n log n)

Q2) Consider the weights and values of items listed below. Note that there is only one unit of each of them.

Item No.	Weight (Kgs)	Value (Rupees)
1	10	60
2	7	28
3	4	20
4	2	24

Your task is to pick a subset of these items such that their total weight is no more than 11 Kgs and their total value is maximised. Moreover, no item may be split. The total items, total value of items picked by an optimal algorithm is denoted by Vopt. A greedy algorithm sorts the item by their value to weight ratios in descending order and picks them greedily, starting from the first item in the ordered list

The total items/total value of items picked by the greedy algorithm is denoted by V_{greedy}. The value of $V_{opt} - V_{greedy}$ is 16

All combinations:
- <1> = 60
- <2,3> = 48
- <2,4> = 52
- <3,4> = 44

n=4, M=11.
N_{opt} = 60 Splitting not allowed!

Greedy Strategies

$\frac{P_1}{w_1} = \frac{60}{10} = 6$ $\quad \frac{P_3}{w_3} = \frac{20}{4} = 5$

$\frac{P_2}{w_2} = \frac{28}{7} = 4$ $\quad \frac{P_4}{w_4} = \frac{24}{2} = 12.$

$x_4 = 1$ <2+4>
$x_3 = 1$ = Profit = 44 } By greedy approach

$\therefore V_{opt} - V_{greedy} = 60 - 44$
$\Rightarrow 16$

If splitting allowed:

$x_4 = 1$ $\quad <2 + \frac{9}{10} \times 10> = 11.$

$x_1 = 9/10$ $\quad = 24 + \frac{9}{10} \times 60$

Profit $\Rightarrow 78$

Problem ② Job Sequencing with deadlines

→ Given a single CPU, using non-preemptive scheduling

→ Given a set of n jobs (process)/tasks/programs

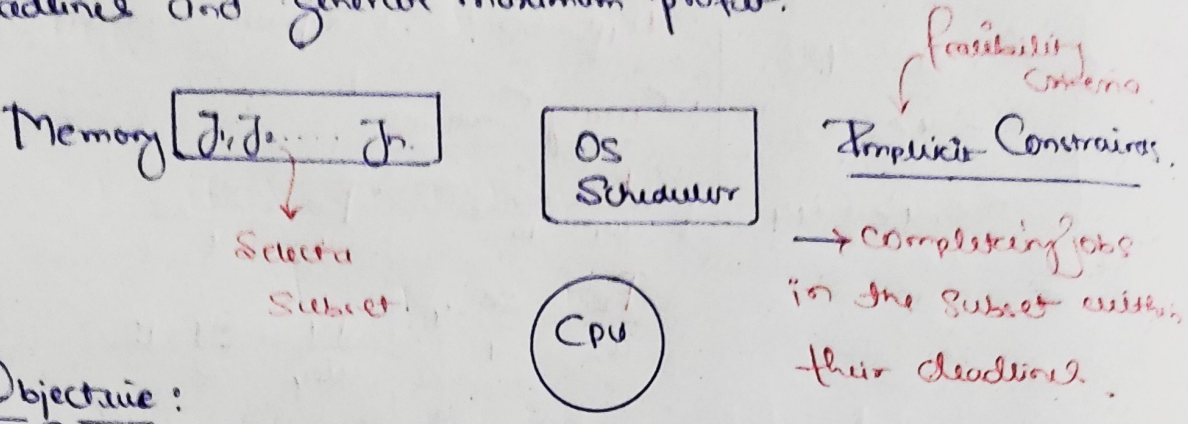
↓

J_i, $i = 1$ to n.

- A.T.
- B.T (1 unit)
- Deadline d_i (integer) > 0
- Profit P_i

→ If job is completed within the deadline, you get the profit

Problem definition: Select a subset of 'n' given jobs, such that the jobs in the subset are completable within their deadlines and generate maximum profit.



Objective:

function: Maximise the profit.

Size of Solution Space: No. of subsets = $\{2^n\}$

eg: $n=4$ $\langle J_1 \ldots J_4 \rangle$

$\langle d_1 \ldots d_4 \rangle = \langle 2, 1, 2, 01 \rangle$

$\langle p_1 \ldots p_4 \rangle = \langle 100, 15, 25, 40 \rangle$

A.T = 0

B.T = 1.

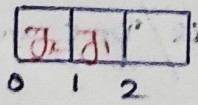
$J \leftarrow \phi$. 1> $|J| =$ zero (feasible)

2> $|J| = 1$.

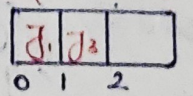
$J \leftarrow \{J_1\}$ or $\{J_2\} \ldots \{J_4\}$ (feasible)

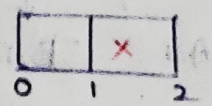
— all subsets with 1 job are feasible.

3> $|J| = 2$ $J \leftarrow \{J_1, J_2\}$ (feasible)

CPU 

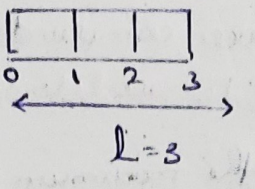
= 115.

$|J| = 2$, $J \leftarrow \{J_1, J_3\}$ (feasible) CPU 

$J \leftarrow \{J_2, J_4\}$ ✗ (not feasible) 

$J \leftarrow \{J_4, J_3\}$

4) $|J| = s$ $\quad J \leftarrow \{J_1, J_2, J_3\}$



Max Jobs in a feasible Solution

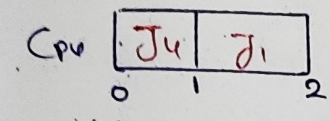
$\Rightarrow \max(d_i)_{i=1,n}$

↳ B.T(J_i) = 1.

eg. $n = 4$, $\langle J_1 \ldots J_4 \rangle$

$\langle d_1 \ldots d_4 \rangle = \langle 2, 1, 2, 1 \rangle$

$\langle p_1 \ldots p_4 \rangle = \langle 100, 15, 25, 40 \rangle$ ✓ × × ✓

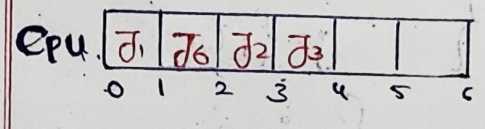


Final profit = 140

eg. $n = 6$, $\langle J_1 \ldots J_6 \rangle$

$\langle d_1 \ldots d_6 \rangle = \langle 2, 3, 4, 2, 4, 3 \rangle$

$\langle p_1 \ldots p_6 \rangle = \langle 14, 18, 12, 10, 8, 5 \rangle$ ✓ ✓ ✓ × × ✓

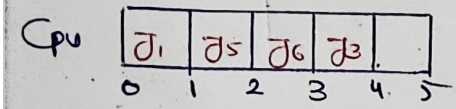


Profit = 59

eg: $n = 6$ $\langle J_1 \ldots J_6 \rangle$

$\langle d_1 \ldots d_6 \rangle : \langle 2, 1, 5, 1, 2, 3 \rangle$.

$\langle p_1 \ldots p_6 \rangle : \langle 28, 12, 5, 18, 30, 20 \rangle$ ✓ × ✓ × ✓ ✓

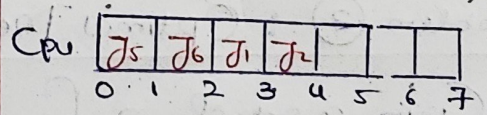


Profit = 83

eg. $n = 7$,

$\langle d_1 \ldots d_7 \rangle : \langle 4, 4, 4, 4, 3, 4, 3 \rangle$

$\langle p_1 \ldots p_7 \rangle : \langle 70, 85, 12, 18, 80, 60, 10 \rangle$



Profit = 265

Q1.) We are given 9 tasks $T_1 \ldots T_9$. The execution of each task require one unit of time. We can execute one task at a time. Each task T_i has a profit P_i and deadline d_i, Profit P_i is earned if the task is completed before the end of deadline.

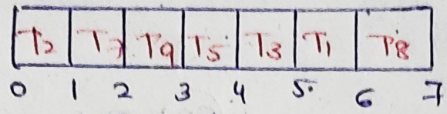
Task	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
Profit	15	20	30	18	18	10	23	16	25
Deadline	7	2	5	3	4	5	2	7	3

a) Are all tasks completed in schedule that gives max. profit?

→ T_4 and T_6 are left out

b) What is the maximum profit?

→ 147



Max profit = 147

Algorithm GreedyJob(d, t, n)

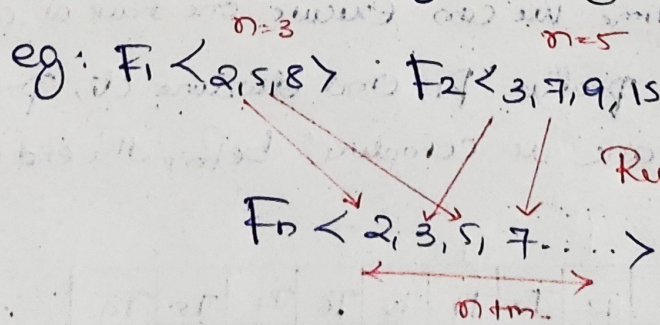
```
{
  J = {1};
  for i = 2 to n do:  ⟹ O(n)
  { if (all jobs in J ∪ {i} can be completed
        by their deadlines)        } O(n)
      then J = J ∪ {i};
  }
}
```

Time Complexity = $O(n^2)$

Problem ③ : Optimal Merge Patterns

→ Given a set of n-files $\langle F_1, F_2 \ldots F_N \rangle$

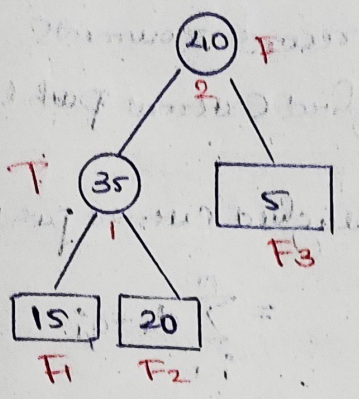
Each file contains a set of records in sorted order. It is required to merge the given N files, to get a single file in sorted order, using 2-way merging.



Metric: no of record movements

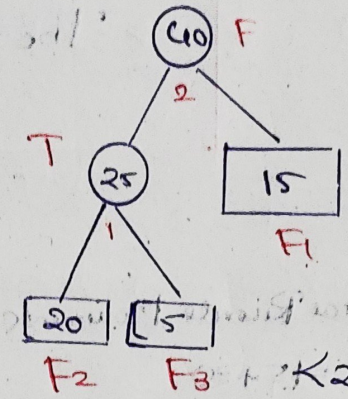
9/8/2023

→ $F_1 = 15$, $F_2 = 20$, $F_3 = 5$



<1, 2, 3>

Total records mov. = 35+40 = 75.



<2, 3, 1>

Total record movements = 65.

→ We want that pattern that generates minimum no of records movements. (Optimal)

* Solution Space : n!

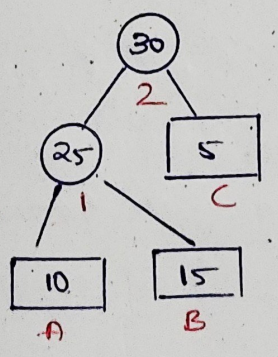
→ Given two files 'A' and 'B' having 'n' and 'm' records respectively. To merge them, using 2-way merging it required (n+m) record movements.

→ Given 'n' files, it is required to merge them using 2-way merging, such that the total no. of record movements is minimum.

* for n-files it required (n-1) merge operations.

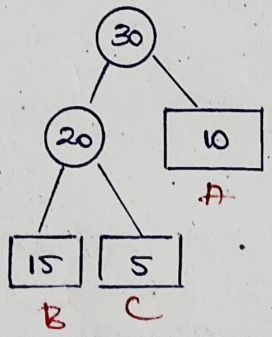
eg: A=10, B=15, C=5.

(2-way binary merge tree)



⇒ 25+30 = 55

<A B C>



⇒ 20+30 = 50

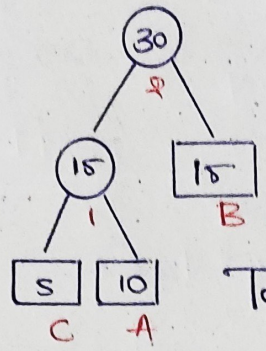
<B C A>

→ We want the pattern which is generating minimum no. of record movements.

→ Solution Space = n!

Greedy Strategy: - A=10, B=15, C=5.

→ The no. of record movements = weighted external path length



Total Records Movements.
= 15 + 30
= 45

Weighted external path length

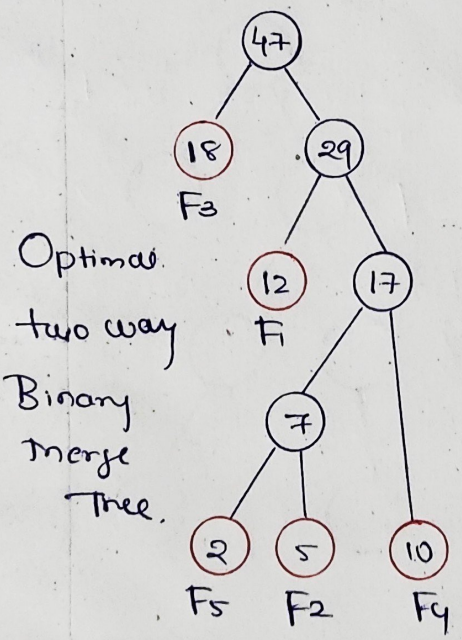
$$= \sum_{i=1}^{n} d_i + q_i$$

d_i = distance from root to F_i

q_i = frequency / size of F_i.

= 2×10 + 1×15 + 2×5
= 45

Q1) $\langle F_1 \ldots F_5 \rangle$: $\langle 12, 5, 18, 10, 2 \rangle$.



Optimal two way Binary merge Tree.

No. of records movements

7 + 17 + 29 + 47 = 100

Weighted extended path lengths = $\sum_{i=1}^{n} d_i + q_i$

= 2×12 + 4×5 + 1×18 + 3×10 + 4×2

⇒ 100

```
tree node = record { tree node * lchild;
                     tree node * rchild;
                     integer weight; }
```

Defination of tree node data struct

Algorithm Tree(n)

```
{                                  // list is a global list of single node
  for i=1 to n-1 do
  {  pt = new treenode;  // Get a new tree node
     (pt → lchild) = least(list);  // merge two lists
     (pt → rchild) = least(list);  // Gets smaller lights.
     (pt → weight) = ((pt → lchild) → weight + ((pt → rchild)
                                  → weight);
     Insert(list, pt);
  return least(list);  // tree left in the list is the
}                                  merge tree
```

Time Complexity: for n-files.

→ time complexity depends upon implementation of list.

Linked list

: for 'n' files

least : O(n)

Insert : O(1)

→ The overall time complexity = $O(n^2)$

Heap (non-linear)

: assume we have heap with 'n' elements

Insert :
Delete :
dec/inc : key.

} O(log n) time complexity.

→ Overall time complexity = $O(n \log n)$

Space Complexity:: Input: 'n'

Space complexity: O(n)

Huffman Coding: (An application of optimal merge pattern)

↳ {data encoding + data compression technique.}

↓ representing an element with some code.

eg: 4 characters/elements

Straight forward / normal encoding: 2 bits

- 00 - a
- 01 - b
- 10 - c
- 11 - d

eg: 100 characters

→ 7 bits/character

$\lceil \log_2 N \rceil$ bits required.

* Data compression is never possible with uniform encoding.
* Compression is possible with non uniform encoding.
* Codes are assigned based on frequency [non-uniform encodings]

↓

Huffman Coding

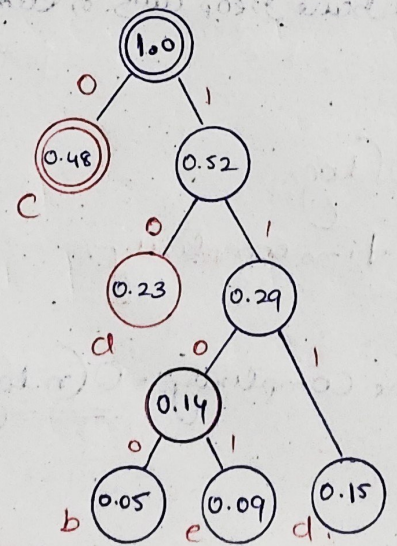
$L = \Sigma(a, b, c, d, e)$ → uniform encoding: 3 bits/character.

→ non-uniform.

* we are given frequency

= <0.23, 0.05, 0.48, 0.15, 0.09>

(↳ we need to apply optimal merge pattern algorithm)



a → 10 (2 bit)
b → 1100 (4 bit)
c → 0 (1 bit)
d → 111 (3 bit)
e → 1101 (4 bit)