Jan 7, 2020 TUESDAY Lecture 2

Data Structures

RECAP

Clever way of arranging data in memory so that update/query operations can be performed efficiently.

RANGE MINIMA PROBLEM

Given an array $A[1 \ldots n]$, indices i, j

Output: Minimum element in $A[i \ldots j]$

Strategy 1 Run a loop from $A[i]$ to $A[j]$ and find the minimum element.

Time: $O(n)$

Storage: $O(1)$ extra words

Strategy 2 Precompute a $n \times n$ matrix B such that

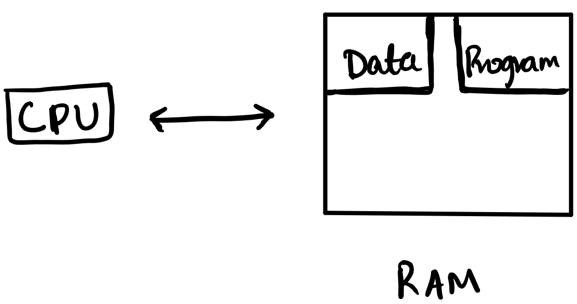
$$B(i,j) = \text{min elt. from } A(i) \text{ to } A(j)$$

Time: $O(1)$

Storage: $O(n^2)$

HW: Can you use a data structure of size $O(n)$ and get an efficient algorithm?

MODEL OF COMPUTATION



How are instructions executed?

- Decode the instruction
- Fetch the operands
- Performs arithmetic/logical operations
- Write back the answer to memory

WORD RAM MODEL   1 word : Smallest entity of RAM

- Basic entity of a problem is a word.
- Fetching any word from the RAM takes the same amount of time.
- 1 instruction is a few clock cycles.

$\boxed{\text{GRAPH MODEL}}$
(later)

FIBONACCI NUMBER

Input: n, m (both int)

Output: $F(n) \bmod m$

1 min, 10 min, 10 hour

HW: Implement IFib and RFib, note the time.

IFib: $5 + 4(n-1) + 1$ instructions

$\approx 4n$ Running time

RFib: let $G(n)$ be the time taken to compute $RFib(n, m)$

if $n = 0$ $\quad G(0) = 1$

$n = 1$ $\quad G(1) = 2$

$n \geq 2$ $\quad G(n) = 3 + G(n-1) + G(n-2)$

$$G(n) > F(n) \quad \forall n$$

Prove: $F(n) > 2^{n-2/2}$??

$G(n) > 2^{n-2/2}$

WARM UP EXAMPLE

Powering:

Input: x, n, m

Output: $x^n \bmod m$

Strategy 1 Run a loop for n iterations and compute $x^n \bmod m$

$\approx 3n$

$$x^n = \begin{cases} \text{if } n = 0 & x^n = x^0 = 1 \\ \text{if } n \text{ is even}, & x^n = x^{n/2} \times x^{n/2} \\ \text{if } n \text{ is odd}, & x^n = x^{n/2} \times x^{n/2} \times x \end{cases}$$

Jan 8, 2020 WEDNESDAY Lecture 3

RECAP

Powering:

Input: x, n, m

output: $x^n \bmod m$

```
Power (x, n, m):
    if (n = 0) then return 1
    temp ← Power (x, n/2, m)
    else if (n mod 2 = 0)
        return (temp × temp) mod m
    else return (temp × temp × x) mod m
```

Let $T(n)$ = no. of inst executed to compute Power (x, n, m)

$$T(n) = 1 + T(n/2) + 4$$
$$= 5 + T(n/2) = 5 \lg n$$

– Can we express $F(n)$ as a^n for some const a?
→ NO!

$$\begin{pmatrix} F(n) \\ F(n-1) \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F(n-1) \\ F(n-2) \end{pmatrix}$$

$$\vdots$$

$$= \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n-1} \begin{pmatrix} \cancel{F(1)}\ 1 \\ \cancel{F(0)}\ 1 \end{pmatrix}$$

<u>Hw:</u> Implement this to get a algo for $F(n)$ mod m.
call this CleverFib.

COMPLEXITY OF AN ALGORITHM

The worst case number of instructions executed while running the algorithm over a fixed input size as a function of the input size or some parameter of the input size.

Example

	Input size
– $F(n)$ mod m	$\lg n + \lg m$
– Given an array A Check if A is sorted	n
– Given a $n \times n$ matrix, output its square	n^2

MatMult

MatMult(C[n,n], D[n,n])

for $i = 0$ to $n-1$
  for $j = 0$ to $n-1$
    $M(i,j) \leftarrow 0$ — 1
    for $k = 0$ to $n-1$
      $M(i,j) = M(i,j) + C(i,k) * D(k,j)$ — 2

Return M — 3

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

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

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

$$= 2n^3 + n^2 + 1$$

COMPARING FUNCTIONS

When comparing two functions we only care about large values of n.

f	g
$2n^2 + 5n + 100$ ✓	$n^2 + 2n + 50$
$2n^2 + 5n + 100$ ✓	$100n + 1000$

f	g	h
$5n^2+n+100$	n^2+10	$10n^{1.5}+1000$

$$\lim_{n\to\infty}\frac{g}{f} = \frac{1}{5}$$

$$\boxed{\lim_{n\to\infty}\frac{h}{f} = 0}$$

→ only this can be used to predict which f is larger ??

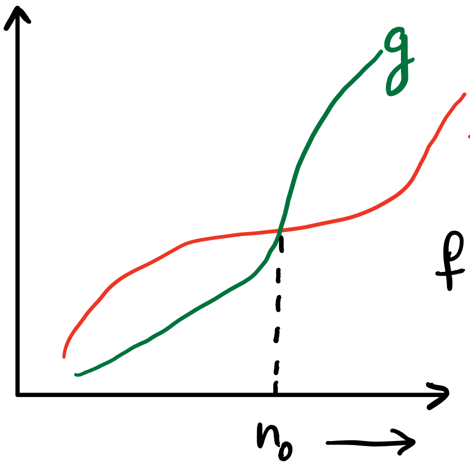
ORDER NOTATION

Let f and g be two increasing functions.

We say f is of the order of g (denoted as $f = O(g)$)

if $\exists$ constants n_0, C s.t

$$\forall n \geq n_0,\ f(n) \leq C \cdot g(n)$$



$f = O(g)$

$O(5n^2+n+100) = O(n^2)$ Loose bound

$O(n^2+10) = \underline{O(n^2)} = O(n^3)$

$O(10n^{1.5}+1000) = O(n^{1.5})$

tight bound

HW: Prove the following statements

1) If $f = O(g)$ and $g = O(h)$ then show that $f = O(h)$
2) If $f = O(h)$ and $g = O(h)$ then $f+g = O(h)$
3) Is $3^n = O(2^n)$? Prove.   DONE!

Jan 10, 2020   FRIDAY   Lecture 4

Maximum Sum Subarray Problem —

Input: An array A

Output: A subarray of A with the maximum sum

```
Trivial Algo(A) {
    max ← A[0]
    for i = 0 to n-1 {
        for j = i to n-1 {
            temp ← compute_sum (A, i, j)
            if max < temp    max ← temp
        }
    }
    return max
}
```

```
Compute_sum (A, i, j) {
    sum ← A[i]
    for k = i+1 to j {
        sum ← sum + A[k]
    }
    return sum; }
```

O(n) time algorithm

```
MaxSum (A) {
    S[0] ← A[0]          ← O(1)
    for i=1 to n-1 {     ← n-1 repitions
      if S[i-1] > 0  then  S[i] ← S[i-1] + A[i]  } O(1)
              else  S[i] ← A[i]
    }
    Scan S to find max entry → O(n)
}
```

Time complexity: O(n)

```
MaxSum Improved (A) {
        S_i = A[0]
        S_i-1 = A[0]
        for i = 1 to n-1 {
            S_i = max (A[i], S_i + A[i])
            S_i_1 = max (S_i_1, S_i)
        }
        return S_i_1
}
```

HW: Solve the problem in O(n) time using O(1) space

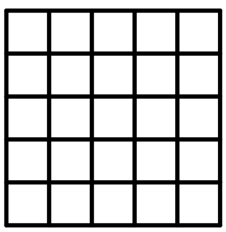
Strategy → Efficiency → Proof of correctness.

Proof of correctness

→ For every possible input the algorithm gives a correct answer.

Prove correctness of the above algorithm.

FINDING LOCAL MINIMA IN A GRID



$n \times n$ matrix consisting of distinct no.

A cell $A(i,j)$ is said to be a local minima if it is smaller than its 4 neighbors.

Input: $n \times n$ matrix A.

output: (i,j) s.t $A(i,j)$ is a local minima

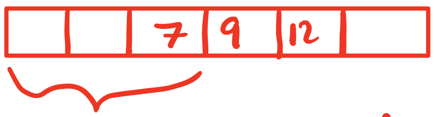
- There always exists a local minima.
  Global minima is a local minima.

Strategy 1

```
Explore (A)
  c ← some cell of A;
  while (c is not local minima) {
        c ← smallest neighbor of c;
  }
  output c;
```

1. Every idea is important
2. Can we solve a simpler variant of the problem?

Local minima in a 1D Array



there exists a local minima

HW: Give a $O(\lg n)$ time algo for computing local minima in a 1D array.

Jan 14, 2020 TUESDAY Lecture 5

PROOF OF CORRECTNESS

- The algorithm outputs the correct answer for all possible inputs.
- Loop invariant

Eg.: // compute $\sum_{i=1}^{n} i$

```
Sum(n) {
  s ← 0;
  for i = 1 to n {
        s ← s+i; }
  Output s;
```

$P(i)$: At the end of the i^{th} iteration of the for loop,

$$S = \sum_{k=0}^{i} k$$

Loop invariant

Proof by induction on i

Base case $i=0$

$S=0$ by definition

$$= \sum_{k=0}^{0} k = 0$$

Assume $P(i-1)$ is true

$P(i)$: The value of s at the end of $(i-1)^{th}$ iteration $= \sum_{k=0}^{i-1} k$ — By induction hypo.

Now, at the end of the i^{th} iteration,

$$S = \sum_{k=0}^{i-1} k + i = \sum_{k=0}^{i} k$$

Hence proved.

→ Loop invariant is an assertion that is true at the end of each iteration of the loop.

MAXIMUM SUM SUBARRAY PROBLEM

A: input array

S:

$P(i)$: At the end of the i^{th} iteration of the loop,
$S(i) =$ the sum of the max subarray, ending at $A(i)$.

Theorem: If $S(i-1) > 0$ then $S(i) \leftarrow S(i-1) + A(i)$
else $S(i) \leftarrow A(i)$

$\boxed{\text{HW: Prove } P(i).}$

Jan 15, 2020 WEDNESDAY Lecture - 6

Theorem: A local minima in a 1D array containing n distinct numbers can be found in lg n time.

```
LocMinInGrid (M) {
 L ← 0 , R ← n-1; found ← false;
 while (not found) {
     mid ← (L+R)/2 ;
     min ← index of the min. elt in M(*, mid);
     if (min is a local min) then found ← true;
     else if M(min, mid) > M (min, mid+1)
           L ← mid+1;
     else R ← mid-1;
 }
 return M (min, mid);
```

Proof of correctness:

P(i) : $\exists$ a local minima between M(*, L) and M(*, R).

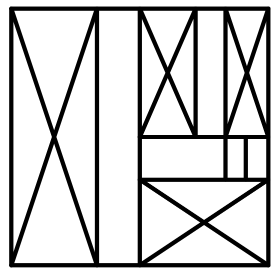
$\Updownarrow$

$\exists\ j$ s.t $M(j, L) < M(*, L-1)$ and

$\exists\ j'$ s.t $M(j', R) < M(*, R+1)$

HW: Give a counter example.

Does not work →



HW:
O(n) implementation for this problem.

Jan 17, 2020 FRIDAY Lecture 7

GCD $a \geq b > 0$

```
GCD (a,b) {
  while (b≠0)
  { t ← b;
    b ← a mod b;
    a ← t; }
}
return a;
```

Euclid's Theorem: If $a \geq b > 0$ then $\gcd(a,b) = \gcd(b, a \bmod b)$

Proof of correctness of GCD algorithm

P(i): Let a_i and b_i be the values stored in the variables a, b at the end of the i^{th} iteration. Then

$$\gcd(a,b) = \gcd(a_i, b_i)$$

Base case: $\gcd(a,b) = \gcd(a,b)$. trivial

Induction hypothesis:

$$\gcd(a,b) = \gcd(a_{i-1}, b_{i-1})$$

Now at the end of i^{th} iteration

$$a_i = b_{i-1} \; ; \; b_i = a_{i-1} \bmod b_{i-1}$$

$$\gcd(a_i, b_i) = \gcd(b_{i-1}, a_{i-1} \bmod b_{i-1})$$

$= \gcd(a_{i-1}, b_{i-1})$ [By Euclid's Theorem]

$= \gcd(a, b)$ [By induction hypothesis]

BINARY SEARCH

Binary Search (A, key)

HW:
1) Write the pseudocode of binary search
2) Show the time complexity of binary search
3) Devise a loop invariant and prove the correctness of binary search.

RANGE MINIMA PROBLEM

- Given an array A storing n numbers.
- **AIM**: To build a data structure such that queries of the following type can be answered efficiently

Range_Minima$(i, j) \longrightarrow$ outputs

$\min(A(i), A(i+1), \ldots A(j))$

Our Aim is to optimise the following :—

- Query time | $O(\lg n)$
- Space used | $O(1)$
- Preprocessing time | $\underbrace{O(n \lg n)}$

Dictionary problem

$n \simeq 10^6$, #queries $\approx 10^7 \equiv m$

Brute force

Space: $O(1)$

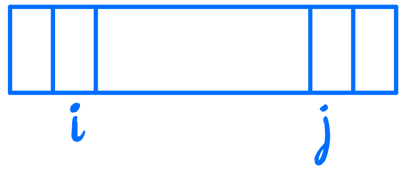
Time: $O(nm)$

$\simeq 10^{13}$

Pre compute $\forall\ i, j$

Space : $O(n^2) \quad \simeq 10^{12}$

Time : $O(1 \cdot m) \simeq 10^7$.



Aim: Each query $\rightarrow$ $O(1)$ time

Fix i,
Require a data structure of size $O(n)$.

Data structure: $O(n \lg n)$

"Efficient" implementation:

Data structures used:

- $Pow(m) = 2^k$ s.t $2^k \leq m$
  $0 \leq m \leq n-1$
- $Log(m) = \lfloor \lg m \rfloor$
  $0 \leq m \leq n-1$
- $B(i,k) = \min(A[i], A[i+1], \ldots, A(i+2^k))$

Range_Minima (i,j)

$L \leftarrow j-i$;
$t \leftarrow$ Pow (L) ;
$k \leftarrow$ Log (L) ;
if $(L = t)$ output $B(i,k)$
else output
  $\min(B(i,k), B(j-t,k))$

$O(1) \longrightarrow$ For each query
$O(n \lg n) \rightarrow$ additional space
$O(n^2 \lg n) \rightarrow$ Preprocessing. Can we do it in $O(n \lg n)$ time?

HINT: FIBONACCI

Jan 21, 2020 TUESDAY Lecture 8

Data structures

Aim: To manage data in an efficient manner so that queries can be answered efficiently.

Abstract Data Structures

Two things:

1) Give mathematical model of the data structure.
   ↳ A Formal definitions of the various operations

2) Implementation: Efficient way to design the various operations.

e.g. - List of students in this class
- list of items in a shop
- list of cases pending in Supreme court

Array based solution

- Can use arrays
- We need an upper bound on the size
- array, current number of elements in it.
- Blocking a large chunk of memory.

Operations on a list

- Query operations
  1. IsEmptyList (L): checks if a list is empty
  2. Search (L, x): checks if the element x is present in L.
  3. Successor (p, L): returns the element in list L after position p.

e.g. L: $x_1, x_2, \ldots, \underset{\uparrow p}{x_i}, x_{i+1}, \ldots, x_n$.

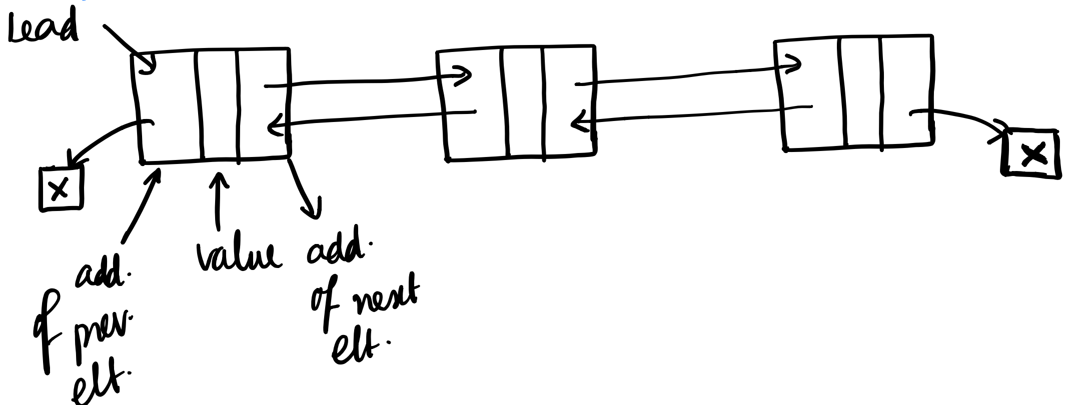
Successor $(p, L) \rightarrow x_{i+1}$

4. Predecessor (p, L): returns the previous element in list L before position p.

- Update operations
  - Create EmptyList (L)
  - Insert (x, p, L): inserts element x into the list L at position p.
  - Delete (p, L): delete the element at position p in L.
  - MakeList Empty (L).

Operations		Array	DLL
Is Empty List		$O(1)$	$O(1)$
Search		$O(n)$	$O(n)$
Successor		$O(1)$	$O(1)$
Predecessor		$O(1)$	$O(1)$
CreateEmpty List		$O(1)$	$O(1)$
Insert	motivation for linked list	$O(n)$	$O(1)$
Delete		$O(n)$	$O(1)$
Make List Empty		$O(1)$	$O(1)$

Linked list (DLL)



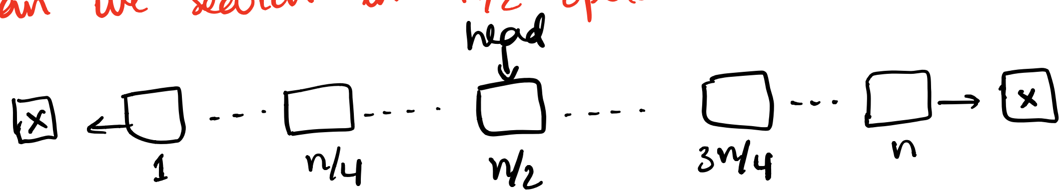
HW: Implement a list data structure. | Quiz on 9^{th} Feb Sunday

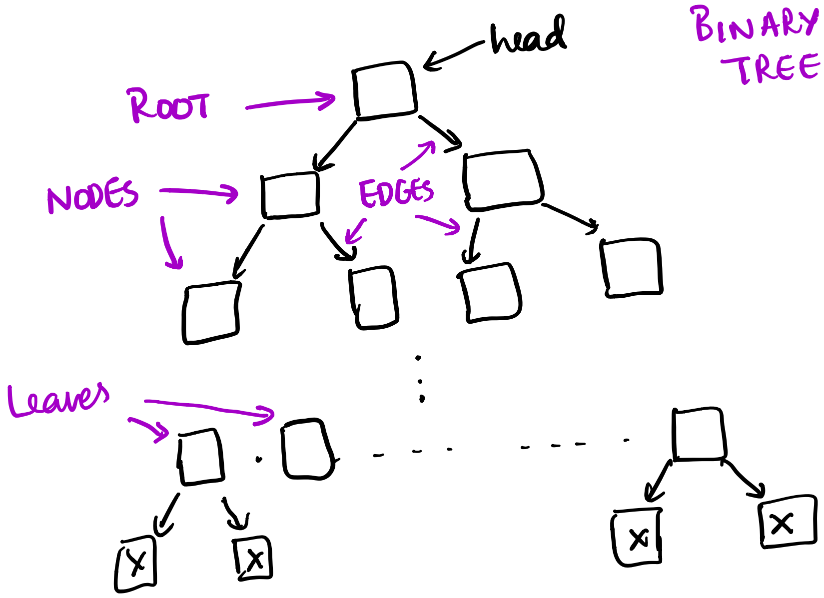
Jan 22, 2020 WEDNESDAY lecture 9

Implementing a telephone directory

Operation	Sorted array	Sorted list
SEARCH	$O(\lg n)$	$O(n)$
INSERT / DELETE	$O(n)$	$O(1)$

Can we search in $n/2$ operations in a sorted list?





BINARY TREE Start at a root, grow and branch.
At the end of branches you have leaves.
Inverted tree structure.

A binary tree is a collection of nodes connected by edges s.t

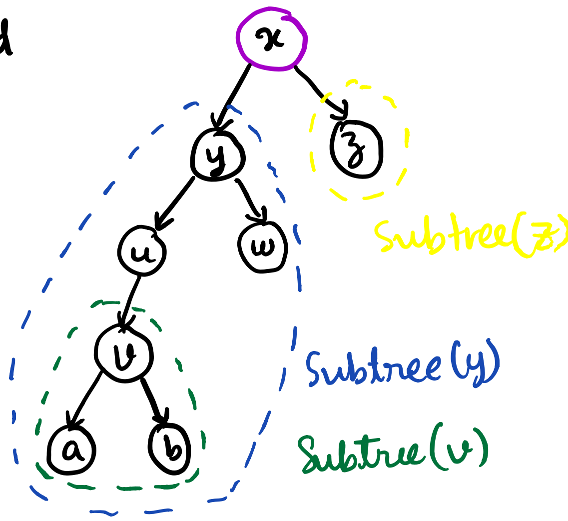
1. Every node has at most 2 outgoing edges
2. There is one node with 0 incoming edges (called root)
3. Every node except for the root has 1 incoming edge.

4. Between every pair of nodes there is at most 1 edge

Some Terminologies

- if there is an edge from u to v, then u is called the parent of v and u is a child of u

- the height of a binary tree is the maximum number of edges from the root to a leaf node



- We say binary Tree T is perfectly balanced if for all nodes v in T,

$$||\text{subtree}(\text{left}(v))| - |\text{Subtree}(\text{right}(v))|| \leq 1$$

Parent(y) = x
child (y) = {u, w}
child (u) = {v}
Height (T) = 4

THEOREM: Let T be a perfectly balanced binary tree containing n nodes. Then Height $(T) \leq \lg n$

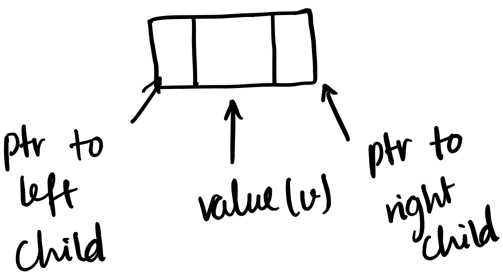
Complete the formal proof

BINARY SEARCH TREE

A binary search tree is a binary tree having values at every node, such that for all nodes $v \in T$,

value $(v) \le$ value of every node in the right subtree of v

value $(v) \ge$ value of every node in the left subtree of v

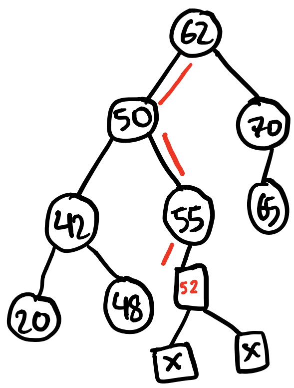


HW: Write pseudocode.

Jan 24, 2020 FRIDAY

Binary search trees

- Search and Insert in a BST



INSERT (52)

Every insertion happens at a leaf node.

HW: Write the pseudocode for Search(T, x) and Insert(T, x) in a BST.

$O(H)$ where H is the height(T)

Definition: A binary tree is said to be nearly balanced if at every node v,

$$|\text{subtree}(\text{left}(v))| \le \frac{3}{4}|\text{subtree}(v)| \quad \text{and}$$

$$|\text{subtree}(\text{right}(v))| \le \frac{3}{4}|\text{subtree}(v)|$$

Let, $H(n)$ be the height of a nearly balanced binary tree T, with n nodes

$$H(1) = 0$$

$$H(n) \le H\left(\frac{3n}{4}\right) + 1$$

$$\le H\left(\left(\frac{3}{4}\right)^2 n\right) + 2$$

$$\vdots$$

$$\le \log_{4/3} n = \frac{\lg n}{\lg 4/3} = O(\lg n)$$

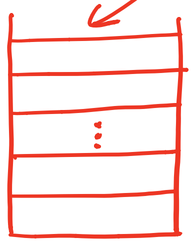
- Now suppose we start with a perfectly balanced BST and keep inserting / deleting nodes then the tree will remain nearly balanced for sometimes.

- We can maintain a fourth field at every node v,

$$|\text{subtree}(v)| \rightarrow \text{size of subtree rooted at } v.$$

Stack

top of the stack



S

- Linear
- Insert / delete at the top of the stack only
  push / pop
- we can only query the top of S.

Operations on a stack (Mathematical model)

Query

- IsEmpty(S) : checks if S is empty.
- Top(S) : returns the top element of S.

Update

- Push(S, x) : inserting x at the top of the stack.
- Pop(S) : deleting the top element from S.

8 queens problem $\rightarrow$ Can we place 8 queens in a chessboard so that no queen kills any other queen?
$\rightarrow$ Stacks

Jan 28, 2020 TUESDAY

RECAP

Stack

- linear data structure
- insert / delete / query happens on one end (call top)
- $\underbrace{\text{IsEmpty, Top,}}_{\text{query}}$ $\underbrace{\text{Push, pop}}_{\text{update}}$

Evaluating an arithmetic expression

$$9*8^2-1+3/6 = 9*64-1+3/6$$

Precedence of operators

1. ^
2. /, *
3. +, -

↓ High to low

$2^\wedge\underbrace{3^\wedge 2} = 512$ | Power is right associative

- Associativity : in an expression with multiple operators of the same precedence which operator gets executed first.

^ is right associative

/, - are left associative

Simpler Problem

- No parenthesis
- Every operator is left associative

Algo do multiple scans and solve each operator one by one

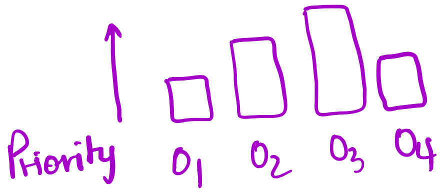
Disadvantages - multiple scans
- can only handle a fixed set of operators
- case based

$$\underbrace{n_1\, o_1\, n_2\, o_2\, \overbrace{n_3\, o_3\, n_4}^{n'}}\, o_4 \,\cdots$$

for each operator o_i there is a function priority(o_i) s.t

if priority$(o_i) >$ priority(o_j)

then o_i has higher precedence than o_j



$$E: \; n_1\, o_1\, n_2\, o_2\, \underbrace{n_3\, o_3\, n_4}_{n'}\, o_4\, n_5$$

$$n_1\, o_1\, \underbrace{n_2\, o_2\, n'}_{n''}\, o_4\, n_5$$

– Use two stacks — one for operators, one for operands.
N- stack O- stack

When can we execute o_i immediately?

→ Priority $(o_i) >$ Priority (o_{i-1})

Priority $(o_i) \geq$ Priority (o_{i+1})

$$E: \; n_1\, o_1\, n_2\, o_2 \,\ldots\, n_k$$

```
// Insert a symbol into O_stack of least priority
while (           ) {
    x ← next_token () ;
    if ( x is an operand)  Push (x, N_Stack) ;
    else {
        while (Priority (x) ≤ Priority (top(O_stack)))
        {   o ← Pop ( O_stack);
            t1 ← Pop ( N_stack);
```

```
            t2 ← Pop (N_Stack);
            t3 ← t2 o t1 ;
            Push (t3, N_Stack)
         }
         Push (x, O_Stack) ;
      }
   }
```

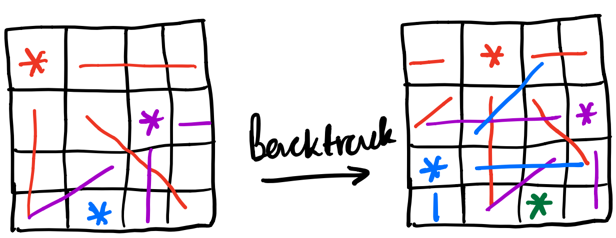
HW: Allow parentheses / Allow associativity $\Big\}$ Re-implement the algo

Jan 29, 2020 WEDNESDAY

8 queens' problem

4 x 4 case

Is it possible to place the 4th queen ?
→ yes !

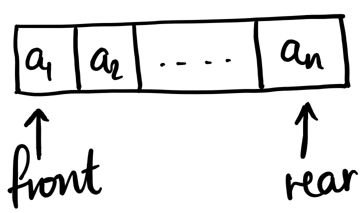


Ex: Write a program to implement a stack and using the stack solve the 8 queens' problem.

Queue

- Linear data structure
- Insertion happens at the rear and removal happens from the front.
- FIFO

queue



Mathematical modelling of a queue

query
- IsEmpty (Q): checks if a queue is empty
- Front (Q): Outputs the element at the front of the queue. $a_1, \dots, a_n$ ↑front

update
- Enqueue (x, Q): Inserts x at the rear of Q
- Dequeue (Q): Deletes the element at the front and outputs it.

eg: Q: $a_1, \dots, a_n$

Enqueue (x, Q) — $a_1, \dots, a_n, x$

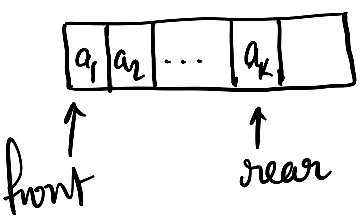
Dequeue (Q) — $a_2, a_3, \dots, a_n, x \rightarrow a_1$

Dequeue (Q) — $a_3, a_4, \dots, a_n, x \rightarrow a_2$

Implement a queue using arrays

- Size of the queue is at most n.

Q :




```
Enqueue (x, Q) {
    rear ← rear + 1
    Q[rear] ← x
    size ← size + 1
}
```

```
Dequeue (Q) {
    front ← front + 1
    size ← size - 1
    return (Q (front - 1))
}
```

- This implementation might go beyond the array boundary after several enqueue and dequeue.

Work around

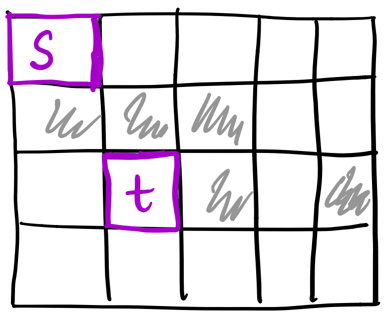
```
Enqueue (x, Q) {
    rear ← (rear + 1) mod n
    Q[rear] ← x
    size ← size + 1
}
```

```
Dequeue (Q) {
    temp ← front
    front ← (front + 1) mod n
    size ← size - 1
    return (Q(temp))
}
```

Ex: Implement the Front (Q).

Finding the shortest path in a grid

n



n

Some cells are blocked

2D array filled with 1's & 0's s.t 0's are the blocked cells.

- Given two more cells — Start, target (s, t)
- Horizontal, vertical moves allowed

Problem: Find the shortest length path $s \to t$ using available cells.

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

$$T(1,1) = 1$$

Jan 31, 2020 FRIDAY

Shortest path in a grid

$G(i,j) = 1$ if (i,j) is not an obstacle
$\quad\quad\;\; = 0$ otherwise

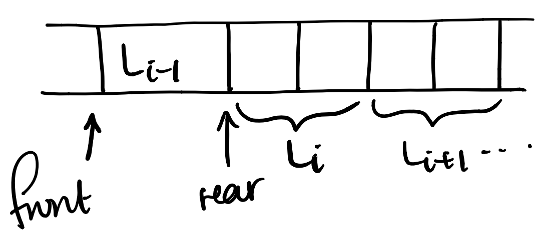
- A cell is at a distance i from s if $\exists$ a neighbour that is at a distance $i-1$, and there is no neighbor at distance $< i-1$.

```
Enumerate_next_layer (G, L_{i-1}) {
  for (each cell c in L_{i-1}) {                    O(n)
     for (each neighbor b of c that is "empty") {   O(1)
         Add b to L_i
     }
  }
  Return L_i;
}
```

```
Dist_in_Grid (G, s) {
   T <- {s};
   for (i=1 to n^2 ) {                               O(n^2)
       T <- Enumerate_next_layer (G, T)  -> O(n)
   }
```

Time complexity $= O(n^3)$

$L_0, L_1, \ldots\; L_{i-1}, L_i, L_{i+1}, \ldots$



- use a queue to simplify algorithm.

```
Shortest_Dist_in_Grid (G, s) {
    for (each cell c in G that is not an obs.) dist(c) ← ∞      O(n^2)
    dist(s) ← 0;    // Enqueue, enumerate on dequeue
    Enqueue(s, Q);
    while (IsEmpty(Q) is false) {                               O(n^2)
        x ← Dequeue(Q);
        for (each neighbor y of x s.t y is
              not an obstacle and dist(y) = ∞) {                 O(1)
            dist(y) ← dist(x) + 1;
            Enqueue(y, Q);
        }
    }
}
}
```

Every cell is enqueued at most once and in every iteration of the while loop, one cell gets dequeued.

Time Complexity : $\underline{\underline{O(n^2)}}$

Proof of correctness

– At the end, dist (c) stores the shortest distance from s to c.

$P(i)$: dist correctly stores the shortest distance to every cell that is at a distance i from s.

$P(0)$: True

Assume $P(i-1)$ is true & prove $P(i)$

Complete the proof!

Feb 4, 2020 Tuesday

Majority Problem

Given a multiset containing n elements, output an element that occurs $> n/2$ times if it exists.

Idea: Given an element, checking whether it is majority element or not can be done in $O(n)$ time.

Algo 1: check for every element, whether or not it's majority
$O(n)$

Algo 2: - sort
- pick middle element and check whether or not it's the majority element. $O(n \lg n)$

An assumption in the previous algorithm is that we can compare two elements as $<, >, =$

- If we are only allowed $=$ & $\neq$ then algo 2 does not work.

$$S = \left\{ \begin{array}{l} \boxed{a,\ b,}\ \boxed{d,} \ \ldots, x, y \quad : \text{distinct} \\ c,\ c,\ c,\ c, \ldots, c \quad : \text{majority elt.} \end{array} \right.$$

- observation 1: If we remove 2 distinct elements from S, then the majority element is preserved in the new set.

S = | c | c | a | c | b | d | c | b | c | c |

S = | a | a | c | c | c | c | b | b | c | c |

S' = | a | c | c | b | c |

- Observation 2: If S consists of $n/2$ pairs of similar element in each pair then choosing one element per pair and creating a set S' will ensure that,

S has a majority $\Longleftrightarrow$ S' has a majority and the element is same

Algo Assumes $size(S) \leftarrow 2^n$

Input : S

1. pair the elements of S $\rightarrow O(n)$
2. If a pair has distinct elements then remove it $O(n)$
3. Create a new set by choosing one element per pair having the same element and repeat step 1
4. Finally check if the last element is majority or not.

$$cn + \frac{cn}{2} + \frac{cn}{4} + \cdots + 1 \leq 2cn = O(n)$$

- uses $O(n)$ extra space
- Aim:
  - $O(n)$ time
  - $O(1)$ extra space
  - $O(1)$ pass over the input

```
Majority (S) {
    count ← 0
    for i = 0 to n-1 {
        if (count = 0) {
            x ← A(i);
            count += 1;
        }
        else if (x = A(i)) {
            count += 1; }
```

loop invariant is something about x that does not change.

```
        else { count -= 1 }
    }
    check if x is maj. or not
}
```

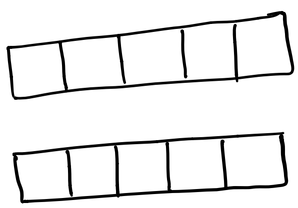
HW: Prove the correctness of the algorithm

Feb 5, 2020 Wednesday

Divide and Conquer

Has 3 parts —
1) Given a problem divide it to two or more subproblems.
2) Assume solutions to the subproblems. Also need to take care of the base case.
3) Combine solutions of subproblems to get solutions of original problem. Usually the toughest step.

Merging two sorted arrays



A: n elts
B: m elements
both sorted

How to combine them into a single sorted array?

```
Merge ( A[0,...,n], B[0,...,m], C) {        → O(n+m)
    i ← 0      // index for A
    j ← 0      // index for B
    k ← 0      // index for C
    while (i<n & j<m) {
        if (A(i) < B(j)) {
            C(k) ← A(i);
            k++; i++; }
```

```
else {
      C(k) ← B(j);
      k++; j++; }
}
while (i<n) {
      C(k) ← A(i); i++; k++; }

while (j<m) {
      C(k) ← A(j); j++; k++; }
```

```
MergeSort( A(i...j)) {
   if (i<j) {
     mid ← (i+j)/2 ;
T(n/2) ← MergeSort ( A (i, ..., mid));
T(n/2) ← Mergesort (A (mid+1, ...,j));
O(n) ← Merge ( A [i,...,mid], A (mid+1, ...,j), C);
O(n) ← Copy C into A;
   } }
```

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

$$= 2\left(2T\left(\frac{n}{4}\right) + c\frac{n}{2}\right) + cn$$

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

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

$$= \lg n \cdot cn = O(n \lg n)$$

MergeSort (A) : takes an array A and sorts it.

MergeSort : Sorts an array

- Time: $O(n \lg n)$
- Extra Space : $O(n)$. merge is not possible without this.

Multiplying two n bit integers (+ve ints assumed)

- Measure of efficiency : no. of bit operations
- Add 2 n-bit numbers. $\longrightarrow O(n)$
- $\underset{\substack{\uparrow \\ n \text{ bit}}}{X} * 2^n$ $\quad \boxed{X}\underbrace{\boxed{0}\boxed{0}\boxed{0}}_{n} \longrightarrow O(n)$

$$\underset{\substack{\uparrow \\ n \text{ bit}}}{X} * \underset{\substack{\uparrow \\ n \text{ bit}}}{Y} = Z$$

$\longrightarrow$ naive algorithm takes $O(n^2)$ bit operations.

Divide and Conquer

X: $\underbrace{\boxed{A}}_{\frac{n}{2}}\underbrace{\boxed{B}}_{\frac{n}{2}}$ $\qquad$ Y: $\boxed{C}\boxed{D}$

$$X = A * 2^{n/2} + B$$
$$Y = C * 2^{n/2} + D$$

$$X * Y = AC * 2^n + (AD + BC) 2^{n/2} + BD$$

$T(n)$: no. of bit operations to multiply 2 n-bit integers.

$$T(n) = 4\, T\left(\frac{n}{2}\right) + cn$$

$$=$$

Feb 7, 2020 Friday

RECAP

Multiplying 2 n-bit integers

- $X+Y$: $O(n)$ bit operations
- $X * 2^n$: $O(n)$ bit ops
- $X * Y$: $O(n^2)$ bit ops

Divide & Conquer Approach

$X = A * 2^{n/2} + B$ $\quad X = \boxed{A \mid B}$

$Y = C * 2^{n/2} + D$ $\quad Y = \boxed{C \mid D}$

$X * Y = AC * 2^n + (AD + BC)\, 2^{n/2} + BD$

$T(n)$: Time taken to multiply 2 n-bit ints

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

$$= 4\,(4T(n/4) + cn/2) + cn$$

$$= 4^2\, T\left(\frac{n}{2^2}\right) + cn + 2n$$

$$= 4^2\left(4T(n/2^3) + \frac{n}{2^2}\right) + cn + 2cn$$

$$\vdots$$

$$= 4^{\lg n}\, T(1) + cn + 2cn + 4cn + \cdots + 2^{\lg n - 1}\, cn$$

$$= O(n^2)$$

$$X * Y = AC\,2^n + (AD + BC)\,2^{n/2} + BD$$

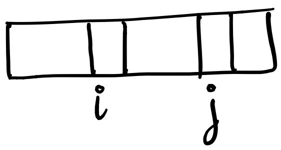
$$(A-B)(C-D) = AC + BD - AD - BC$$

$$AD + BC = AC + BD - (A-B)(C-D)$$

1. AC
2. BD
3. $(A-B)(C-D)$

$$T(n) = 3\,T(n/2) + cn$$
$$\vdots$$
$$= O(n^{\lg_2 3}) \leftarrow \text{verify.}$$

Counting number of inversions in an array

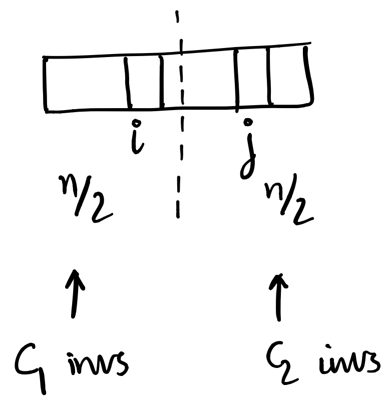
A: 

$i < j$

(i, j) is an inversion if $A(i) > A(j)$ for $i < j$

Problem: count the no. of inversions.

Max number of inversions $= {}^nC_2 = \frac{n(n-1)}{2} = O(n^2)$

A: 

Inversion1 $(A(i, \dots j))$ {

$c_3 \leftarrow 0;$

if $(i < j)$

$c_1 \leftarrow$ Inversion1 $(A(i, \dots \frac{i+j}{2}))$

$c_2 \longleftarrow$ Inversion2 $(A(\frac{i+j}{2}+1, \dots, j))$

```
for (k = i to (i+j)/2) {
    for (l = (i+j)/2 + 1 to j) {
        if (A(k) > A(l))
            c3++;
    }
}
return c1 + c2 + c3
```

$$T(n) = 2T(n/2) + cn^2$$
$$= O(n^2)$$ check!

Inversion 2 (A(i, ..., j)) : Sorts A(i, .. j) and returns # of invs.

```
c3 <- 0;
if (i<j)
   c1 <- Inv2 (A(i, ... (i+j)/2))
   c2 <- Inv2 (A((i+j)/2 +1, ..., j))
   for (k = i to (i+j)/2) {
        do Bin Search in A((i+j)/2 +1, ..., j)
        to find an index l s.t
        A(i) > A(l) and
        A(i) <= A(l+1)

        c3 <- c3 + (l - (i+j)/2);
   }
return c1 + c2 + c3;
Merge (A(i, ..., (i+j)/2), A((i+j)/2, ..., j), c) }
```

else {return 0;}

$$T(n) = 2T(n/2) + \underbrace{\frac{n}{2}(\lg n) + cn}_{c' n\lg n}$$

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

```
Merge_and_count (A, i, j, C) {
  mid ← i+j/2 ; p ← i; q ← mid+1; k ← 0 ; count ← 0.
   while ( p < mid && q < j ){
     if ( A(p) < A(q)) {
               C(k) ← A(p) ;  k++;  p++; }
     else {
            C(k) ← A(q);  k++;  q++;
            count ← count + ( mid-p);
           }
   }
   return count;
}
```

Feb 12, 2020 Wednesday

RECAP

Solving Recurrences using induction

- Given $T(n)$
- Guess an appropriate $f(n)$
- Assume $T(n) \leq f(n) \ \forall \ n<m$
- Show that $T(m) \leq f(m)$
- Show $T(1) \leq f(1)$

Common Errors

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

$$T(1) = 0$$

Let, $f(n) = O(n) = an$

Assume $T(n) \leq an \quad \forall \ n<m$

$$T(m) = 2T(m/2) + 4m$$

$$= am + 4m$$

$$= O(m)$$ ✗ incorrect

Solving using the Master Theorem

- multiplicative functions.

A function $f: \mathbb{N} \to \mathbb{N}$ is said to be multiplicative if

$f(n) \cdot f(m) = f(nm)$

e.g., n^2 $\quad n^2 \cdot m^2 = (nm)^2$

$n^{1.5}$ $\quad n^{1.5} m^{1.5} = (nm)^{1.5}$

$2n^2$ $\quad 2n^2 \cdot 2m^2 = 4(nm)^2 \neq 2(mn)^2$

$$f(a^k) = [f(a)]^k$$

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

Master Theorem

Consider a recurrence of the form $T(n) = aT(n/b) + f(n)$, $T(1) = 1$ where a, b are constant ints and $b > 1$ and $f(n)$ is a multiplicative function. Assume $n = b^k$ for some k.

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

$$= a(aT(n/b^2)) + f\left(\frac{n}{b}\right)) + f(n)$$

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

$$= a^2\left(aT\left(\frac{n}{b^3}\right) + f\left(\frac{n}{b^2}\right)\right) + a f\left(\frac{n}{b}\right) + f(n)$$

$$= a^3 T(n/b^3) + a^2 f(n/b^2) + a f(n/b) + f(n)$$

$$\vdots$$

$$= a^k T(n/b^k) + a^{k-1} f(n/b^{k-1}) + \dots + a f(n/b) + f(n)$$

$$= a^k + \sum_{i=0}^{k-1} a^i f(n/b^i)$$

$$= a^k + \sum_{i=0}^{k-1} a^i \frac{f(n)}{(f(b))^i} = a^k + f(n) \sum_{i=0}^{k-1} \left(\frac{a}{f(b)}\right)^i$$

Consider <u>3 cases</u>

$$\boxed{k = \log_b n}$$

$$x^{\log_y a} = a^{\log_y x}$$

- $a = f(b)$

$$T(n) = a^k + k \cdot f(n)$$

$$= a^k + k \cdot f(b)^k$$

$$= a^k + k \cdot a^k$$

$$= a^k (k+1)$$

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

$$= n^{\log_b a} (\log_b n + 1) = O(n^{\log_b a} \cdot \log_b n)$$

$a < f(b)$

$$T(n) = a^k + f(n) \cdot \sum_{i=0}^{k-1} \left(\frac{a}{f(b)}\right)^i$$

$$= a^k + f(n) \leq f(n) + f(n) = O(f(n))$$

- $a > f(b)$

$$T(n) = a^k + f(n) \cdot \frac{\left(\frac{a}{f(b)}\right)^k - 1}{\frac{a}{f(b)} - 1}$$

$$= a^k + \cancel{f(b)^k} \cdot \frac{a^k - f(b)^k}{\cancel{f(b)^k} \underbrace{\left(\frac{a}{f(b)} - 1\right)}_{O(1)}}$$

$$= O(a^k) = O(n^{\log_b a}).$$

In summary

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

$$T(n) = \begin{cases} O(n^{\log_b a} \cdot \log_b n) & ; \quad a = f(b) \\ O(f(n)) & ; \quad a < f(b) \\ O(n^{\log_b a}) & ; \quad a > f(b) \end{cases}$$

count ← count + (mid − p)

Feb 14, 2020 **Friday**

```
Quicksort (A, i, j) {
    if (i<j) {
        k ← Partition (A, i, j);
        Quicksort (A, i, k-1);
        Quicksort (A, k+1, j);
    }
}

Partition (A, p, r) {
    x = A[r]
    i = p-1
    for j = p to r-1
        if A[j] ≤ x
            i = i+1
            exchange A[i] with A[j]
    exchange A[i+1] with A[r]
    return i+1 }
```

Worst case complexity : $O(n^2)$

Average time complexity :

$$T_{avg}(QS) = \frac{1}{n} \sum T_A(\pi)$$

π is permutations of $\{1, \ldots, m\}$

Where, $T_A(\pi)$ is the time taken by QS on array A having permutation π.

eg.,

6	8	2	3
3	4	1	2

50	100	1	10
3	4	1	2

→ Time taken is the same as the relative ordering is same.

→ We will assume WLOG that A elements from $\{1, \ldots, n\}$

P_i = all permutations that start with i.

$|P_i| = (n-1)!$

$T(n)$: Average time taken by quicksort on an array of size n. $\{1, 2, \ldots, n\}$

$G(n, i)$: Average time taken by QS on an array $\{1, \ldots, n\}$ where the first element of array is i.

$$T(n) = \frac{1}{n} \sum_{i=1}^{n} G(n, i)$$

$$G(n, i) = T(i-1) + T(n-i) + cn$$

$$\therefore \; T(n) = \frac{1}{n} \sum_{i=1}^{n} \left[T(i-1) + T(n-i) + cn \right]$$

$$= cn + \frac{1}{n} 2 \sum_{i=1}^{n-1} T(i)$$

$$\boxed{T(n) = cn + \frac{2}{n} \sum_{i=1}^{n-1} T(i)}$$

can be written as partial fractions

Using method of induction

$$T(n) \leq an \log n + b$$

$$\therefore T(1) = d \quad \because (b \geq d)$$

Assume $m < n$,

$$T(m) \leq am \log m + b$$

$$T(n) = cn + \frac{2}{n} \sum_{i=1}^{n-1} T(i)$$

$$= cn + \frac{2}{n} \sum_{i=1}^{n-1} (ai \log i + b)$$

$$= cn + \frac{2}{n} \left(\sum_{i=1}^{n/2} ai \log i + \sum_{i=\frac{n}{2}+1}^{n-1} ai \log i \right) + 2b$$

$$\leq cn + \frac{2}{n} \left(\sum ai \log \frac{n}{2} + \sum ai \log n \right) + 2b$$

$$= cn + \frac{2}{n} \left(\sum_{i=1}^{n-1} ai \log n - \sum_{i=1}^{n/2} ai \right) + 2b$$

$$= cn + \frac{\cancel{2}}{n} \left(\frac{n(n-1)}{\cancel{2}} \right) a \log n - \frac{\cancel{2}}{n} \frac{n}{\cancel{2}} \frac{(n/2 - 1)}{2} . a + 2b$$

$$\leq a\log n + b + \underbrace{b + cn - a/4}_{\leq 0}$$

$$\boxed{\frac{a}{4} \geq b + cn}$$