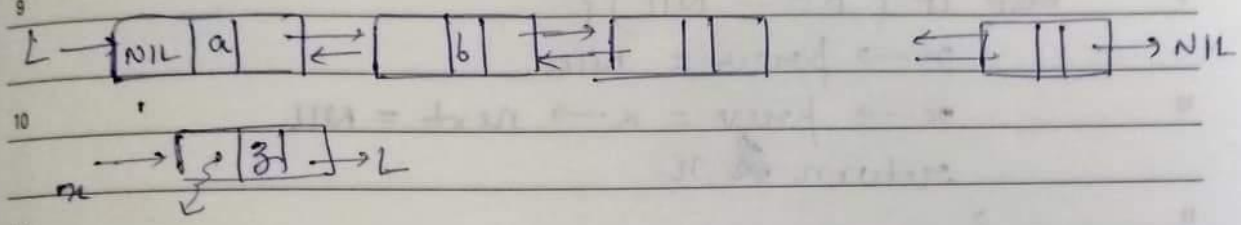
Search (L, k)

Insert (L, x) // At front



⇒ LIST Insert_front (L, x) {

if (L == NIL) return x

else {

L → prev = x

x → next = L

x → prev = NIL

return x

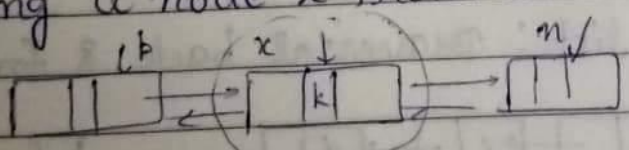
}

}

O(1)

⇒ Use L = Insert_front (L, x)

Deleting a node x from L :-



SUNDAY 03

⇒ LIST Delete (L, x) {

p = x → prev

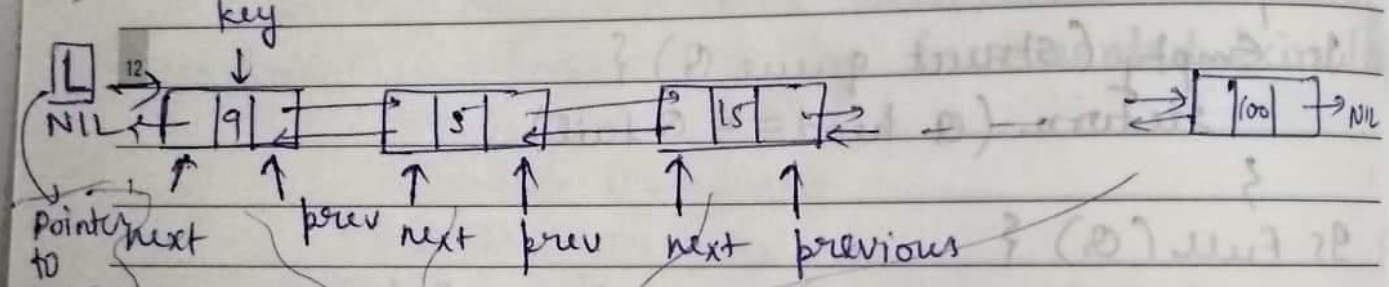
n = x → next

if (L == NIL) "PROBLEM"

2013

```
    x = Q[Q.head]
    head = (head+1) % (N+1)
    return x;
}
}
```

(Doubly) Linked list



List nodes

```
Struct Listnode {
    int key; // Any type
    Struct list node * next;
    Struct list node * prev;
};
typedef struct Listnode * list
```

O(1)

Search for a key k in a list L

```
Struct listnode * search (List L, key){
    x = L
    while (x != NIL and x -> key != k)
        x = x -> next
    return x
```

```
GLOBAL int N
struct queue {
    int Q[0.....N]; // Queue data
    int head = 0;
    int tail = 0;
} Queue
Is full (struct queue Q) {
return (Q.head == Q.tail)
}
Is Empty (struct queue Q) {
    return (Q.head == Q.tail)
}
Is Full (Q) {
    return (Q.tail + 1) % (N+1) == Q.head
}
Enqueue (Q, x) {
    if Is full (Q) {
        return "Error_Q_tailt"
    }
    Else {
        Q.[Q.tail] = x
        tail = (tail + 1) % N+1
    }
}
Dequeue (Q) {
    if Is Empty (Q) {
        return "Error_Q_Empty" {
    else {
```

```
Is Empty (Q) {
    return Q.head == Q.tail
}
Is Full (Q) {
    Q.tail + 1 == Q.head % N+1
}
```

→ Deletion happens from head, head incremented

→ Insertion happens at top incremented ⇒ circul

Insertion ≡ Enqueue

Deletion ≡ Dequeue

Conventions:

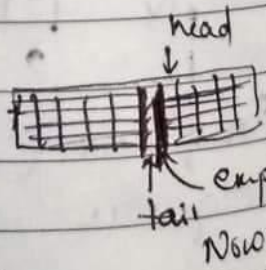
head ≡ index of the earliest (FIFO)

initially head = 0

tail ≡ index where the next item will get inser

initially tail = 0

head == tail ⇔ Empty set



(tail + 1 ≡ head) mod N+1 } Queue full

⇔ (tail + 1) % (N+1) == head }

Lose a slot, Don't be greedy :)

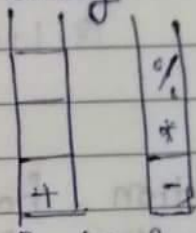
Pseudo Code for implementing head :-

Rules for Infix → Postfix conversion

1. If the current token is an operand
   → pass to output
2. Keep a stack of operators only

Ex: 2 + 3 - 5 * 7 % 5

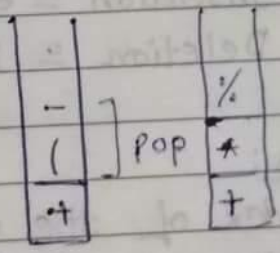
Postfix 2 3 + 5 7 5 % * -



Pop(+) Push(-)

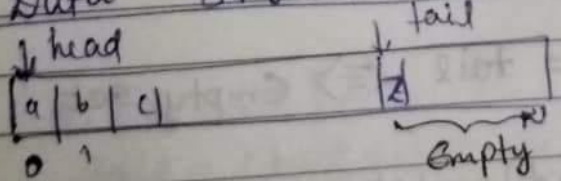
Ex 2 + (3 - 5) * 7 % 5

2 3 5 - 7 5 % * +



Time = Linear Θ(n)

Queue Data Structure — First In First Out (FIFO)



Circular N Queues

Enqueue (Q, x) // Insert
Dequeue // Delete

head == tail ≡ Empty Queue
tail + 1 == head % N + 1 is full

2013

```
Top(S){
If IsEmpty(S){
    returns false
    }
else {
    returns S.St[top-1]
    }
}
Pop(S){ // returns & deletes top item of stack
If IsEmpty{
    return "Error"
    }
else {
    val = S.St[s.top-1]
    top = top-1
    return val
    }
}
```

Infix & Postfix arithmetic exp:

Infix → 2 + (3 - 5) * -7 % 5

Postfix → 2 3 5 - 7(-) 5 % + *

2 3 5 - 7(-) 5 % + *

Infix 2 + 3

Postfix 2 3 +

2 + 3 * 5 → 2 3 5 * +

Priority		Associativity
	- (unary)	
	/ %	LA
	*	LA
	+ -	LA

→ Pop delete the top of the stack
Make_Empty (S)
Push (S, x) // S.Push(x)
Pop (S) // S.Pop
Is Empty(S)
Top (S) : returns the element top of stack.
Is Full(S) returns true if stack is full.

Implementation of a stack by an array :-

```
struct stack {
    int St[1 ... N];
    int top;
} S;
Make_Empty (S) {
    S.top = 1
// top is the next free index
}
Is_Empty (S) { returns S.top == 1 }
Is_full (S) { return S.top == N+1 }
Push (S, x) {
    If not is full (S) {
        S.St [top] = x
        top = top + 1
        return true
    }
    else return false
}
```

FRIDAY • FEBRUARY

22

053-312
WEEK 08

JANUARY

M	T	W	T	F	S	
	1	2	3	4	5	
7	8	9	10	11	12	
14	15	16	17	18	19	
21	22	23	24	25	26	
28	29	30	31			

Insert (S, x) ≡ Push (S, x) : add on top
Delete (S) ≡ Pop (S) : remove top value

Postfix

Infix

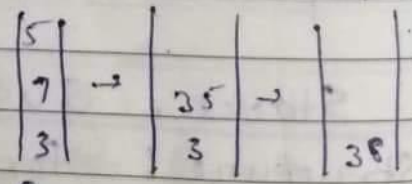
(3+5) - 4 * 6 % 4
operand (operator) operand
precedance

% 3
* 2
-, + 1

→ Stacks convert Infix to Postfix

Postfix : 3 + 5 - 4 → 3 5 + 4 -
operand operand operator
→ operator comes after operand.

Infix	Postfix
3 * 7	3 7 *
3 + 7 * 5	3 7 5 * +



Stacks: Stacks and Queues

→ Insertion into stack is called Push(S, x) places inserted item x at "the top of stack S"

2013

$T(n)$ ≤ Time taken on array of size $\leq 3n/4$ + time taken to reduce the array down to $3n/4$

Time taken on array of size n

Take Expectation on both side.

$E(T(n)) = E(\quad + \quad) = E(1) + E(\quad)$

$T(n) = T(3n/4) + \Theta(n)$

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

expected linear time algorithm.

Correctness -

```
Quick Sort (A, p, q) {
// Sort A[p ... q]
if p == q return
(l,m) = Rand partition (A, p, q, v)   v ∈ A[p] ... A[q]
Quick Sort (A, p, l)
Quick Sort (A, m+1, q) }
```

Worst Case $T(n) = T(n-1) + \Theta(n)$

$= \Theta$ Quadratic

Expected running time $= O(n \log n)$

Stacks, Queues

Data Structure: for dictionary : set of elements

⇒ Stacks, Queues both support following operation.

1) Insert 2) Data Delete

→ Stacks is last in first out structure (LIFO)

2013

WEDNESDAY • FEBRUARY

20

051-314
WEEK 08

Best case:- v = median

$$T(n) = T(n/2) + \Theta(n)$$

Almost good if $T(n) = T(3n/4) + \Theta(n) \equiv \Theta(n) = T(n)$

$T(n) = T(9n/10) + \Theta(n) \equiv \Theta(n) = T(n)$

=) examples:-

[1 2 4 6 8 10] Sorted Array

=) $T(n) = T(3n/4) + \Theta(n)$

=) v is good if v lies in 25^{th} - 75^{th} %ile.

=) The chance of choosing a "good" v is $1/2$

=) On expectation find the expected number of coin tosses of a fair coin before it shows "head"

P(a head occurs first after k trials) = $\frac{1}{2^k}$

$$E = \sum_{k=1}^{\infty} k \cdot \frac{1}{2^k} = 2$$

$E = 1 + \frac{1}{2}E$, so $E = 2$

Let $T(n)$ ≡ the expected time taken by Rand_Select (A, p, q, k) on an input of size n

2013

27/8/18

13 MARCH

M	T	W	T	F	S	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

FEBRUARY • TUESDAY

050-315
WEEK 08

19

Partition (A, p, r, v):-

Time Taken : linear $\Theta(r-p+1)$

⇒ **Randomized_Partition :-**

Choose an index q at random from p...r with equal chance. Call partition with $v = A[q]$

⇒ **Problem with using Partition to solve Select(A,p,r,k)**

let (l, m) = Partition (A, p, r, v)

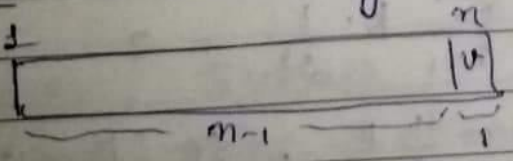
v is one of A[p] ... A[r]

$$\text{Select}(A,p,r,k) = \begin{cases} \text{Select}(A,p,l,k) & \text{if } k \le l-p+1 \\ v & \text{if } l-p+1 < k \le m-p+1 \\ \text{Select}(A,m+1,r,k-(m-p+1)) & \text{o.w} \end{cases}$$

⇒ **Use this partition to compute Median**

select (A, 1, n, $\lceil n/2 \rceil$)

Bad choice :- v is largest in array segment



Worst Case time $T(n) = \Theta(n + n-1 + n-2 + \cdots + \frac{n}{2})$

$= \Theta(n^2)$

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

2013

⇒ Selection (A, p, r)
// A[p...r] find kth smallest element in A[p..r]
if r-p ≤ 1 return A[p]

2 3 1 5 2 6 7

[2 1 2] [3] [5 6 7]

4th smallest element = 3
2nd smallest Select (A_L, 2)
5th smallest Select (A_R, 5)

Partition splits into A_L, A_M and A_R Then select (A, k)

$$= \begin{cases} \text{select } (A_L, k) & \text{if } k \le |A_L| \\ v. & \text{if } |A_L| < k \le |A_L| + |A_M| \\ \text{Select } (A_R, k) & \text{if } k > |A_L| + |A_M| \end{cases}$$

else { v = A[p] // Arbitary
(l, m) = Partition (A, p, r, v)
if (k ≤ l - p + 1)
return Select (A, p, l, k)
else if (~~k ≤ m~~) (l - p + 1 < k ≤ m - p + 1)
return v
else return (A, m+1, r, ~~k-q~~ k - m)
}

⇒ good case
Goal : → choose pivot v so ~~that~~ the splits are balanced.

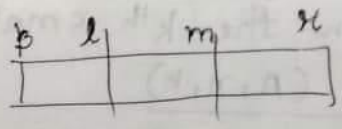
$$T(n) = T(n/2) + O(n)$$
$$= O(n)$$

Bad Case :- Pivot is the largest element

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

Random $T(n) = T(3n/4) + O(n)$
or $T(5n/6) + O(n)$

⇒ Quick Sort (A, p, r) {
// Quick Sort A[p--- r]
if r-p ≤ 1 return
v = A[r]



(l, m) = Partition (A, p, r)

Quick Sort (A, p, l)
Quick Sort (A, m+1, r)

⇒ Selection (A, p, r)
// A[p--- r] find kth smallest element in A[p-- r]
if r-p ≤ 1 return A[p]

2 3 1 5 2 6 7

[2 1 2] [3] [5 6 7]

4th smallest element = 3
2nd smallest Select (A_L, 2)
5th smallest Select (A_R, 5)

Partition splits into A_L, A_M and A_R Then select (A, k)

$$= \begin{cases} \text{select } (A_L, k) & \text{if } k \le |A_L| \\ v. & \text{if } |A_L| < k \le |A_L| + |A_m| \\ \text{Select } (A_R, k) & \text{if } k > |A_L| + |A_m| \end{cases}$$

else { v = A[p] // Arbitary
(l, m) = Partition (A, p, r, v)
if (k ≤ l-p+1)
return Select (A, p, l, k)
else if (~~k ≤ m-p~~) (l-p+1 < k ≤ m-p+1)

Eg: 4 1 5 3 9 7

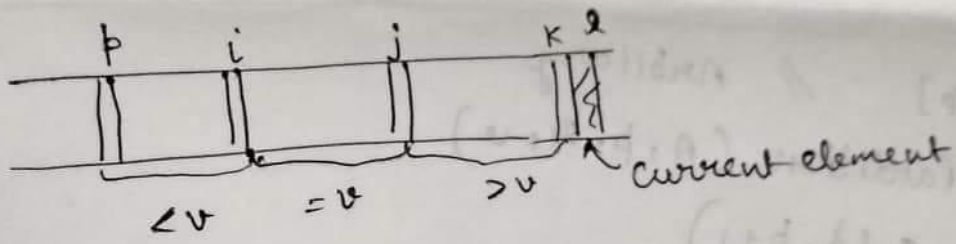
Partition around 3 into 3 parts.

1 | 3 | 4 5 9 7

$<v$ $=v$ $>v$

→ Partition splits $A[p..r]$ into 3 parts: A_L all eleme
A_M all element $= v$, A_R all element $> v$.

Invarient




```
Partition (A, p, r, v) {
    i = j = k = p-1
    for l = p to r {
        if (A[l] > v)        k = k+1 ,
        else if (A[l] == v)
            exchange A[j+1] with A[l]
                j = j+1
                k = k+1
        else if (A[l] < v)
            exchange A[j+1] with A[l]
                k = k+1
                j = j+1
            exchange A[j] with A[i+1]
                i = i+1

    } // end for
    return (i, j)
}
```

24/8/18

Selection Problem: Given an array $A[1 \ldots n]$ and an index $1 \le k \le n$, return the k^{th} smallest element of A.
denoted **Select (A, n, k)**

$k = [n/2]$ is median.

A | 4 | 1 | 7 | 2 | 9 |

A sorted | 1 | 2 | 4 | 7 | 9 |

Select (A, 5, 4) = 7

→ Solve the selection problem without necessarily sorting.
(eg $O(n \log n) \neq O(n)$)

Partition (A, p, q, v) : key or pivot
$A[p \ldots q]$ and v is some value

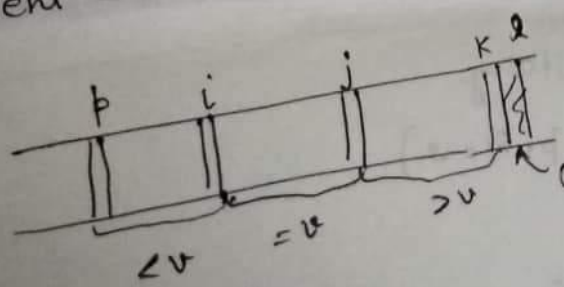
Eg: | 4 | 5 | 3 | 9 | 7 |

Partition around 3 into 3 parts.

| 1 | 3 | 4 | 5 | 9 | 7 |
$<v$ $=v$ $>v$

→ Partition splits $A[p \ldots q]$ into 3 parts: A_L all elements $< v$, A_M all element $= v$, A_R all element $> v$.

Invarient



Partition (A, p, q, v) {

A.

$n! \geq (n/e)^n$ Shorting

$\log n! \geq n \log n/2$

$= \Omega(n \log n)$

No "comparison" Sort algorithm can take time less than $(n \log n)$

$\log n! = \Omega(n \log n)$

This is a lower bound on all comparison sort Algorithms.

Median and Selection Problems :-

eg. 1 1 2 3 5 7 11 21 59

↑ Median

$n = 9$

If n = even 1 1 2 3

both Median.

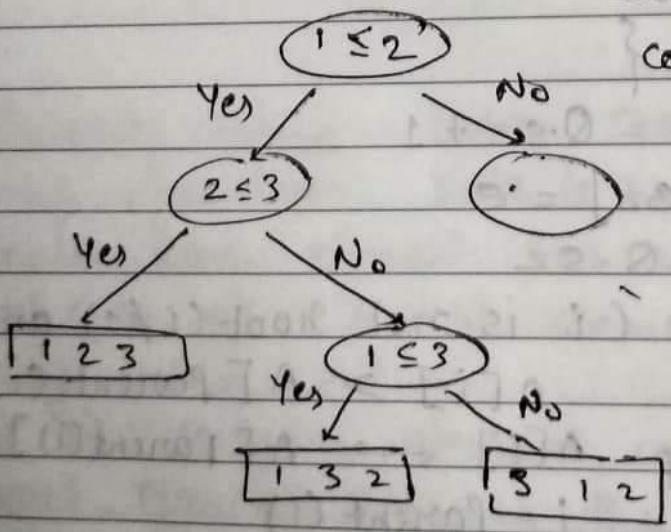
Quantiles q^{th} quantile, the element with rank $\lceil qn \rceil$

while (j is not root and A[j] > A[parent(j)]
{Exchng A[j] ⟷ A[parent(j)]
j = parent(j)
}

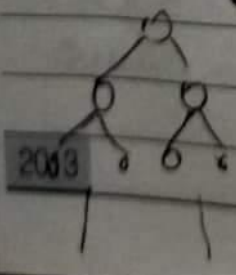
Comparison Sort: Sorting Algorithms that only compare and move data, can be represented in the form of an abstract binary tree.

A = [A[1], A[2], A[3]]

height of comparison sort tree = max no of comparisons on input of size n.



There are n! possible permutations for the sorted order.
The height of



$2^H \geq n!$

$H \geq \log(n!)$

$= \Omega(n \log n)$

Implementing Priority Queue by heap:-

Extract_Max (Q) {
// Q is represented as an array
Q [1, ... Q.sz]
max = Q[1]
Q[1] = Q [Q.sz]
Q.sz = Q.sz - 1
Max_heapify (Q, 1)
}

O(logn)

Insert (Q, e) {
Q.sz = Q.sz + 1
Q[Q.sz] = e
i = Q.sz
while (i is not root (i ≠ 1) and
A[i] > A[Parent (i)])
{ Swap A[i] ⟷ A[Parent(i)]
i = Parent (i)
}}

O(logn)

Increase_key (Q, i, v)
// Increase the priority of i
// in Q to
j = i, Q[i].priority = v

2013

$$\leq n \underbrace{\sum_{h=0}^{\log n} \frac{h}{2^h}}_{\text{Constant}} = \boxed{O(n)}$$

⇒ Heap Sort (A, n) {
    heapsize = n
    Build Heap (A, n)
    for i = 1 to n-1 { O(n)
        Exchg A[1] ⟷ A[Heapsz] ×
        heapsz = heapsz - 1
        Max_Heapify (A, i, heapsz) O(log n)

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

Time ⇒ (Max) Priority Queue :- Data structure support following operations. Data is a collection of elements. Each element has a priority value.

O(log n) 1) Insert (Q, e) : Insert element e with priority value e ∈ v

O(1) 2) Max (Q) : Returns element with max priority

O(log n) 3) Extract_Max(Q) : Returns the largest priority element and remove it from Q.

4. Increase_key (Q, e, v)
    // e priority = e priority + v

Note ! A natural correspondence between max priority Queue and its implementation by a max heap

Build_Heap(A, n) — Time

// n is no of elements in A.

for $i = [n/2]$ down to 1 — $O(n) \times$
    Max_heapify (A, i) — $O(\log n) =$ $O(n \log n)$

Loop invariant - The node i+1 ... to n form max_heap
→ Initially holds bez initially $[n/2] \to [\frac{n}{2}+1] \ldots n$ are leaf indices ⇒ Max_heap.
Inductive step, The premise for Max_heapify is subtrees rooted at 2i, 2i+1 must be max heap by induction, this holds.
So by property of Max_Heapify
// node i is the root of a heap at the end of iteration i
So i to n are now indices for heap.
At termination $i = 0$
By the invariant,
0, i+1 ... n are roots of max heap.
1 ... n is a max heap.

Time taken by Build_Heap :-

$\sum_{h=0}^{\log n}$ (No of nodes at height h) $O(h) \leq \sum_{h} \left\lceil \frac{n}{2^{h+1}} \right\rceil \cdot h$

h = height (1)

MONDAY • FEBRUARY

11

042-323
WEEK 07

JANUARY

M	T	W	T	F	S	S
		1	2	3	4	5
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

(Max) Heap & Priority Queue:-

- Nearly complete binary tree free from left.
- Array represent level wise

Max_Heapify (A, i, n):

```
// n = size of heap
// i is an index in A = a node i in Heap
// the sub-tree rooted at left(i), right(i) are
   max-heaps.
// objective is to fix a possible violation of head
   property at node i
   done = false
while (i is not a leaf node and done is false)
// i leaf ≡ 2i > n
   {   if A[i] < A[left(i)]
           largest = left(i)
       if right(i) exists and A[right(i)] > A[
           largest = right(i)
   if (largest == i)
           done = true,
   else { exchg. A[i] ⟷ A[largest]
          i = largest                    }
```

Time Complexity:

If h is the height of i then time complexity $= O(h) = O(\log n)$

2013

$2^H \leq n \leq 2^{H+1} - 1$

$H \equiv \Theta(\log n)$

$1 \; - - - \; [n/2], [n/2+1], \ldots n$

leaf nodes.

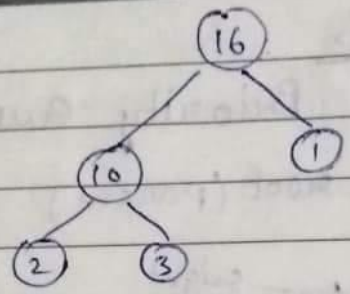
left child $2(n/2+1) > n$. not exists.

$\Rightarrow \left\lceil \frac{n}{2^{H+1}} \right\rceil$ node at height H

Pseudo Code:

```
Max_heapify (A, i, n)
// Assume that the subtree rooted at A[left(i)] & A[right(i)] satisfy max Heap property while 'i' is not a leaf node )
{   // left(i) > n
    if A[i] < A[left(i)]
        largest = left(i)
    if A[right(i)] exists and
    A[largest] < A[right(i)]
        largest = right(i)
Exchange A[i] with A[largest(i)]
    if  i ≠ largest
        i = largest
}
```

Problem ↗ if root node is largest ?



incomplete but filled from left. (Heap)

→ left child of the node of array index i
$= 2i$

→ right child of the node of array index i
$= 2i + 1$

⇒ Parent node of (i) $= [i/2]$

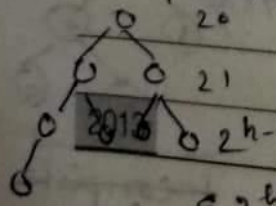
Max Heap Property : in array
For any node i, except the root
index

$A[\text{parent}(i)] \cdot key \geq A[i] \cdot key$

Define the height of a node is the length (No edges) of the longest downward path from i to a leaf node.

Height of a tree ≡ height of its root.

→ Suppose a heap has height H. How many (min/max) nodes does it have.



$n \leq 1 + 2 + 2^2 + \cdots + 2^H$

$n \geq 1 + 2 + 2^2 + \cdots + 2^{H-1} + 1$

$\leq 2^h$

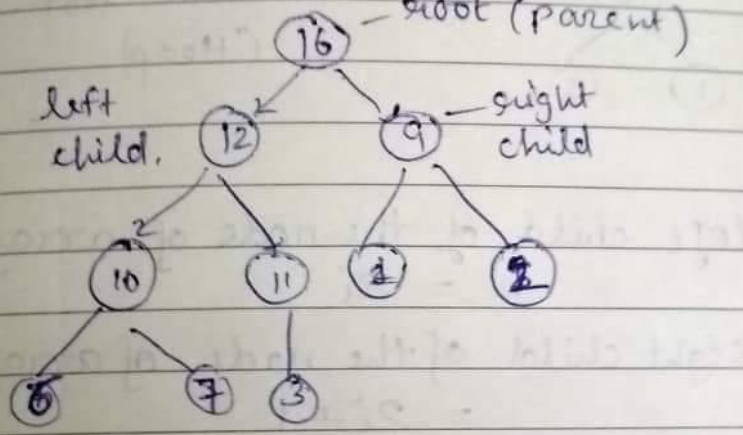
FEBRUARY • THURSDAY

038-327
WEEK 06

07

The Heap

Data Structure / Priority Queue Time represent.

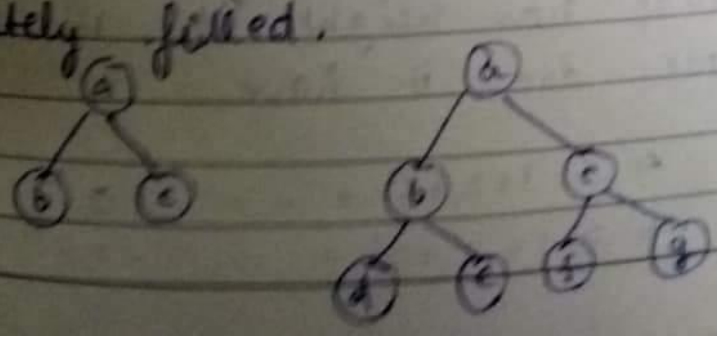


1	2	3	4	5	6	7	8	9	10
16	12	9	10	11	1	2	6	7	3

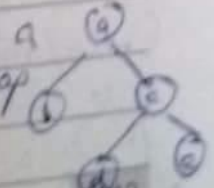
Heapsize = 10

Heap : is a ~~array~~ almost binary tree
→ filled from left

Binary tree : is a tree in which every node has 0, 1 or 2 children.
→ If a node has no children, it is a leaf node.
→ nodes that are not leaf node are called internal node.
→ **Complete Binary tree** : is a binary tree, and every node has (max child node) all levels completely filled.



Not a Heap



~~Incomplete~~ not a tree

given an array A[1...n] and an ind
... element of A.

WEDNESDAY • FEBRUARY

06

037-328
WEEK 06

JANUARY

M	T	W	T	F	S	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

1. Compute F_{n-y}
2. reverse the indices $(F_j)_{1 \ldots n-1}$

A.

$$= \frac{1 - \omega_n^{n(1-j)}}{1 - \omega_n^{(1-j)}} \qquad -(n-1) \le j \le n-1$$

$= 0$ ⇒ inverse is very easy
⇒ conjugate transpose

c) $\|(f_n)_i\|^2 = n$

$F_n^\dagger F_n = nI \Rightarrow f_n^{-1} = \frac{1}{n} f_n^\dagger$

$$f_n^\dagger = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & \bar{\omega}_n & \bar{\omega}_n^2 & \cdots & \bar{\omega}_n^{n-1} \\ 1 & \bar{\omega}_n^2 & \bar{\omega}_n^4 & \cdots & 1 \\ \vdots & & & & \\ 1 & \bar{\omega}_n^{n-1} & \bar{\omega}_n^{2(n-1)} & \cdots & \bar{\omega}_n^{(n-1)^2} \end{bmatrix}$$

(now going backward in the ⊖)

ω_n

Inv_FFT

1. Use Rec_FFT but ~~for~~ $\bar{\omega}_n = \omega_n^{-1}$ instead of ω_n

2) Divide by n

(matrix product coincides)

$$\left\{ \bar{\omega}_n^k = \omega_n^{(n-1)k} = \omega_n^{-k} = \omega_n^{n-k} \right\}$$

$$f_n^\dagger = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \cdots & \omega_n^{(n-1)^2} \\ 1 & & & & \\ 1 & \omega_n & \omega_n^2 & \cdots & \omega_n^{n-1} \end{bmatrix}$$

rows are reordered

$(F_n^\dagger)_{i\text{row}} = (f_n)_{n-i\text{row}}$ $\quad i = 1, 2, \cdots n-1$

to compute $(f_n^\dagger y)$

2013

MONDAY • FEBRUARY

04

// computed first half of the DFT.
// $A(\omega_n^{k+n/2}) = A^{even}(\omega_{n/2}^{k}) - \omega_n^{k+n/2} A^{odd}(\omega_{n/2}^{k})$
$\omega_n^{n/2} = -1$

$y_{k+n/2} = y_k^{even} - \omega \cdot y_k^{odd}$
$\omega = \omega \cdot \omega_n$ }
return y // pt value rep at the nth roots of unity
(insert to get the coeff)

$DFT(a) = F_n \cdot a$

$$F_n = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega_n^1 & \cdots & \omega_n^{n-1} \\ \vdots & & & \vdots \\ 1 & \omega_n^{n-1} & \cdots & \omega_n^{n-1} \end{bmatrix} \Rightarrow \frac{1}{n} F_n \text{ is a unitary matrix}$$

(i) The inner product of any two distinct colums is zero.
(ii) The norm of a column is 1.

* Reals $x, y \in R^n \rightarrow \langle x, y \rangle = x_1 y_1 + \cdots + x_n y_n = x^T y$
complex $\rightarrow \langle x, y \rangle = x_1 \bar{y_1} + \cdots + x_n \bar{y_n} = x^T \bar{y}$

Take two distinct colums, $\begin{matrix} 1 \\ \omega_n^i \\ \omega_n^{2i} \\ \omega_n^{(n-1)i} \end{matrix}$, $\begin{matrix} 1 \\ \omega_n^j \\ \omega_n^{2j} \\ \omega_n^{(n-1)j} \end{matrix}$

(i) $\langle (F_n)_i, (F_n)_j \rangle$
$= 1 + \omega_n^{(i-j)} + \omega_n^{2(i-j)} + \cdots + \omega_n^{(n-1)(i-j)}$
$[j - i \neq 0)$

A.

Evaluate A on $\omega_n^0, \omega_n^1 \ldots \omega_n^{n-1}$

$A^{even}(x^2)$ on $A^{even}(\omega_n^0, \omega_n^2, \ldots \omega_n^{2(n-1)})$

all the $\omega_{n/2}^k$ $k = 0 \ldots (n/2)-1$ Already calculated.

$T(n) = 2T(n/2) + \Theta(n)$

PSEUDO CODE

```
Rec_FFT (a, n) { // a[0, ..., n-1]
// A(x) = Σ_{i=0}^{n-1} a_i x^i, want [A(ω_n^0), A(ω_n^1) ... A(ω_n^{n-1})]
if n == 1 return n
Let a_odd  = [a_1, a_3, ... a_{n-1}] // deg (n/2) - 1
    a_even = [a_0, a_2, ... a_{n-2}] // deg (n/2) - 1

Let y_odd  = Rec_FFT (a_odd, n/2)  // division
    y_even = Rec_FFT (a_even, n/2) // phase
// y_odd  = [y_0^o, y_1^o, ..., y_{n/2-1}^o]
// y_even = [y_0^e, y_1^e, ..., y_{n/2-1}^e]

Let y = [y_0, y_1, ..., y_{n-1}]
Let ω_n = e^{2πi/n}, ω = 1
for k = 0 to n/2 - 1
    y_k = y_k^even + ω y_k^odd & // conquer phase
    // linear time loop
```

SUNDAY 03

2013

... an array A[1 - - - n] ... of A.

FRIDAY • FEBRUARY

01

032-333
WEEK 05

JANUARY 13

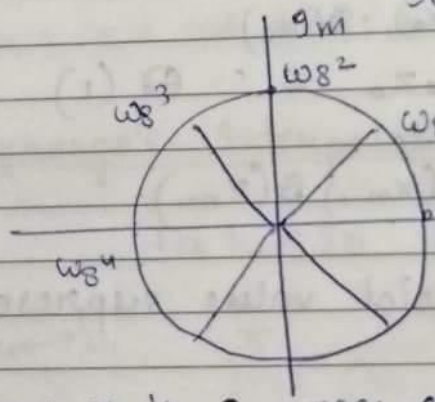
M	T	W	T	F	S	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

13 MARCH

M	T	W	T	F	S
				1	2
4	5	6	7	8	9
11	12	13	14	15	16
18	19	20	21	22	23
25	26	27	28	29	30

$x^n = 1$ for any positive integer n is known to have exactly n roots.

These form a graph generated by primitive roots $\{\omega_n^0, \omega_n^1 \dots \omega_n^{n-1}\}$



Unit circle

⇒ Assume n is a power of 2

$\{1, \omega_n, \dots \omega_n^{n/2-1}, \omega_n^{n/2} \dots \omega_n^{n-1}\}$

Square these no's — Square these no's gives second ground of roots of $n/2$ roots of unity

$(\omega_n^k)^2 = e^{\frac{2\pi i}{n} k \cdot 2} = e^{\left(\frac{2\pi i}{n/2}\right) \cdot k} = (\omega_{n/2}^k)$

$(\omega_n^k)^2$ gives $(\omega_{n/2}^k)^{th}$ root.

$A(k) = a_0 + a_1 x + a_2 x^2 - - - - a_{n-1} x^{n-1}$

$A^{odd}(x) = a_1 + a_3 x + a_5 x^2 - - - a_{n-1} x^{n/2-1}$

$A^{even}(x) = a_0 + a_2 x + a_4 x^2 - - - - a_{n-2} x^{n/2-1}$

$A(x) = A^{even}(x^2) + x A^{odd}(x^2)$

↓ $n/2^{th}$ root — again $n/2^{th}$ root of unity

2013

$A(x_0) \quad \ldots \quad A(x_{n-1})$

$= y_0 \quad \ldots \quad = y_{n-1}$

$B(x_0) \quad \ldots \quad B(x_{n-1})$

$z_0 \quad \ldots \quad z_{n-1}$

$C(x_0) = A(x_0) \cdot B(x_0)$

$= y_0 z_0 = \quad$ ⊕ (1)

$C(x_{n-1}) = A(x_{n-1}) B(x_{n-1})$

⇒ Multiplication in point value representation takes Θ(n)

$A(x) = [a_0 \ldots a_{n-1}]$

$B(x) = [b_0 \ldots b_{n-1}]$

Multiply Θ(n²) → $C(x) = A(x) \cdot B(x)$

$C_k = \sum_{i=0}^{n-1} a_i b_{k-i}$

Evaluate ↓ Θ(n log n) time FFT

Interpolation Θ(n log n) ↑

$A(\omega_{2n}^0), A(\omega_{2n}^1), \ldots A(\omega_{2n}^{2n-1})$

Point-Value Multiplication Θ(n) → $C(\omega_{2n}^0), C(\omega_{2n}^1) \ldots C(\omega_{2n}^{2n-1})$

↓x ↓x

$B(\omega_{2n}^0) \quad B(\omega_{2n}^1) \quad B(\omega_{2n}^{2n-1})$

⇒ $f(n) = 0 g(n)$

$g(n) = 0 f(n)$

⇒ $f(n) = \Theta g(n)$

$$V(x_0, x_1 \cdots x_{n-1}) \begin{bmatrix} a_0 \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} y_0 \\ \vdots \\ y_{n-1} \end{bmatrix}$$

$$\det V(x_0 \cdots x_{n-1}) = \prod_{i<j} (x_i - x_j)$$

If x_i's are all distinct $\Rightarrow$ unique soln.

Lagrange's Interpolation –

Fix $k \in \{0, \cdots, n-1\}$

Polynomial $\rightarrow$ $\prod_{\substack{j=0 \\ j \neq k}}^{n-1} \frac{(x - x_j)}{(x_k - x_j)}$ $\rightarrow$ non zero and constant

$$f_k(x) = A(x) = \sum_{k=0}^{n-1} y_k \prod_{\substack{j=0 \\ j \neq k}}^{n-1} \frac{(x - x_j)}{(x_k - x_j)}$$

Exercise: Compute $[a_0, a_1, \cdots a_{n-1}]$ in $\Theta(n^2)$

[Coefficient Rep]

Evaluation ↓

↑ Interpolation : $\Theta(n^2)$

↳ find coeff rep from

[point value Rep] → pt value at n pt

$f_k(x)$ is a poly. of deg. $n-1$ which is 1 at x_k and 0 at $x_1, \cdots, x_{k-1}, x_{k+1}, \cdots x_{n-1}$

Polynomials

$A(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1}$

degree $= n-1$ deg. bound $= n$

coefficient representation

$A \equiv [a_0 \; a_1 \; \cdots \; a_{n-1}]$

Store array of coeff Point value representation

1) choose n distinct pts $x_0, x_1, \cdots x_{n-1}$

2) Solve $y_0 = A(x_0)$, $y_1 = A(x_1)$ $\cdots$ $y_{n-1} = A(x_{n-1})$

$y_k = A(x_k)$ $\quad k = 0, 1, \cdots n-1$

Time Complexity

1) $A(x)$ is computed in $\Theta(n)$ times

$C(x) = A(x) \cdot B(x)$

degree bound n

$C(x) = C_0 + C_1 x + \cdots + C_{2n-2} x^{2n-2}$

$C_k = \sum_{i=0}^{k} a_i b_{k-i}$

Convolution $c_1, c_2 \cdots c_{2n-2}$ get computed in time $\Theta(n^2)$

In coeff^n rep, Prod takes $\Theta(n^2)$

2013

$C(x) = [c_0, c_1 \text{ ---- } c_{2n-2}]$

$= O(n^2)$ for computation

⇒ **Point Value Representation :-**

$A(x) = a_0 + a_1 x + \text{- - -} + a_{n-1} x^{n-1}$

choose n distint points

$x_0, x_1, \text{- - - } x_{n-1}$

$(x_0, y_0) = (x_0, A(x_0))$

$(x_1, y_1) = (x_0, A(x_1))$

$(x_{n-1}, y_{n-1}) = (x_0, A(x_{n-1}))$

$a_0 + a_1 x_0 + \text{- - -} + a_{n-1} x_0^{n-1} = y_0$

$a_0 + a_1 x_1 + \text{- - -} + a_{n-1} x_1^{n-1} = y_1$

$a_0' + a_1 x_{n-1} + \text{- - -} + a_{n-1} x_{n-1}^{n-1} = y_{n-1}$

Matrix form :

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \text{- -} & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \text{- -} & x_1^{n-1} \\ 1 & & & & 1 \\ \vdots & & & & \\ 1 & x_{n-1} & x_{n-1}^2 & \text{- - -} & x_{n-1}^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ 1 \\ 1 \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} y_0 \\ 1 \\ 1 \\ \vdots \\ y_{n-1} \end{bmatrix}$$

Vander mondes matrix

$$B(x) = b_0 + b_1 x + b_2 x^2 + \cdots + b_{n-1} x^{n-1} + \cdots$$

$$A(x) + B(x) = \sum_{i=0}^{n-1} (a_i + b_i) x^i$$

Representation of Polynomial :-

1) Coefficient Representation

$$A(x) = [a_0, a_1 \cdots a_{n-1}]$$

→ Good for Addition, and evaluating value at x_0

⇒ if $A(x) = [a_0, a_1 \cdots a_{n-1}]$

$B(x) = [b_0, b_1 \cdots b_{n-1}]$

$$A(x) + B(x) \Leftarrow C(x) = [a_0 + b_0, a_1 + b_1 \cdots a_{n-1} + b_{n-1}]$$

⇒ Evaluate A at x_0

$$A(x_0) = a_0 + a_1 x_0 + a_2 x_0^2 + \cdots a_{n-1} x_0^{n-1}$$

$$= ((a_{n-1} x + a_{n-2}) x + a_{n-3}) x + \cdots + a_0$$

$$= H(x)$$

⇒ Multiplication

$$A(x) = \sum_{j=0}^{n-1} a_j x^j$$

$$B(x) = \sum_{k=0}^{n-1} a_k x^k$$

$$C(x) = A(x) \times B(x) = \sum_{i=0}^{l} a_i b_{l-i}$$

DECEMBER
M T W T F S
31
3 4 5 6 7 8 1
10 11 12 13 14 15 2
17 18 19 20 21 22 9
24 25 26 27 28 29 16
23
30

2. Product :

$P_1 = A_{11} S_1 = A_{11}B_{12} - A_{11}B_{22}$

$P_2 = S_2 B_{22} = A_{11} B_{22} + A_{12} B_{22}$

$P_3 = S_3 \cdot B_{11} = A_{21} B_{11} + A_{22} B_{11}$

$P_4 = A_{22} S_4 = A_{22} B_{21} - A_{22} B_{11}$

$P_5 = S_5 \cdot S_6 = A_{11} B_{11} + A_{11} B_{22} + A_{22} B_{11} + A_{22} B_{22}$

$P_6 = S_7 \cdot S_8 = A_{12} B_{21} + A_{12} B_{22} - A_{22} B_{21} - A_{22} B_{22}$

$P_7 = S_9 \cdot S_{10} = A_{11} B_{11} + A_{11} B_{12} - A_{21} B_{11} - A_{21} B_{12}$

Now,

$C_{11} = P_5 + P_4 - P_2 + P_6$

$C_{12} = P_1 + P_2$

$C_{21} = P_3 + P_4$

$C_{22} = P_5 + P_1 - P_3 - P_7$

$T(n) = 7\,T(n/2) + \Theta(n^2)$

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

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

$= \Theta\, n^{2.81}$

⇒ Polynomial $A(x) = a_0 + a_1 x + a_2 x^2 + \dots a_{n-1} x^{n-1}$

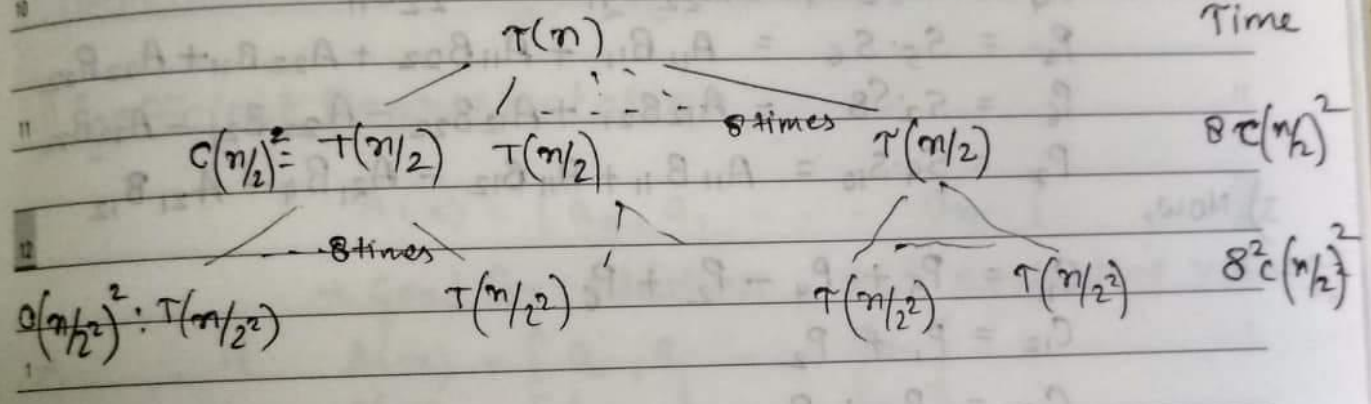
$n-1$ = degree

$a_0, a_1, \dots a_{n-1}$ = coefficient

degree bound is any no. larger than degree.

$T(n)$ is time taken to compute product of 2 $n \times n$ matrix

$$T(n) = 8\,T(n/2) + \Theta(n^2)$$



$$T(n) = cn^2 + 8c(n/2)^2 + 8^2c(n/2^2)^2 + \cdots + 8^{\log n - 1}c\left(\frac{n}{2^{\log n - 1}}\right)^2 + 8^{\log n}T\left(\frac{n}{2^{\log n}}\right)$$

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

1. Compute $S_1 \cdots S_{10}$ (sums)

$S_1 = B_{12} - B_{22}$
$S_2 = A_{11} + A_{12}$
$S_3 = A_{21} + A_{22}$
$S_4 = B_{21} - B_{11}$
$S_5 = A_{11} + A_{22}$
$S_6 = B_{11} + B_{22}$
$S_7 = A_{12} - A_{22}$
$S_8 = B_{21} + B_{22}$
$S_9 = A_{11} - A_{21}$
$S_{10} = B_{11} + B_{12}$

T(n)
n×n

$O(n/2^2)$:

↓.

*) Assume n is a power of 2

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \qquad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$n/2 \times n/2$

$$C = A \cdot B$$

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

$$T(n) = 8\,T(n/2) + \Theta(n^2)$$

No advantage of this iterative method.

Square Matrix Multiplication :-

$$C_{11} = A_{11}B_{11} + A_{12}B_{21}$$
$$C_{12} = A_{11}B_{12} + A_{12}B_{22}$$
$$C_{21} = A_{21}B_{11} + A_{22}B_{21}$$
$$C_{22} = A_{21}B_{12} + A_{22}B_{22}$$

A_{ij}, B_{ij} are $n/2 \times n/2$ size block of A, B
n is a power of 2

Iterative Method : $\Theta(n^3)$

2013

Soln: $T(n) = \Theta(n \log n)$

Improvement over Quad. time algo.

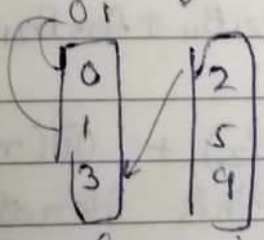
⇒ A [3 1 0 2 5 9]

No of inversions: (3,1) (3,0) (1,0) (3,2)

4 inversions.

Assume distinct nos.

Problem: Give an $O(n \log n)$ time algorithm to count the no. of inversions.



no of cross inversion

Square Matrix Multiplication

A : n×n B : n×n

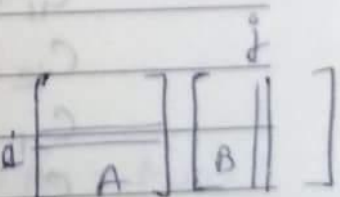
Mat Mult (A, B, n)

Create a new matrix

C : n×n

$C = A \cdot B$

$C_{ij} = \langle A_{i*}, B_{*j} \rangle$



for i = 1 to n

for j = 1 to n

c[i,j] = 0

for k = 1 to n

c[i,j] = c[i,j] + A[i,k] · B[k,j]

Time Complexity: $\Theta(n^3)$

Find_max_subarray (A, lo, hi)
~~if (lo == hi)~~
if (lo > hi) return ;
if (lo = hi) return (A[lo], lo, lo)
// lo < hi
mid = $\left\lfloor \frac{(lo + hi)}{2} \right\rfloor$

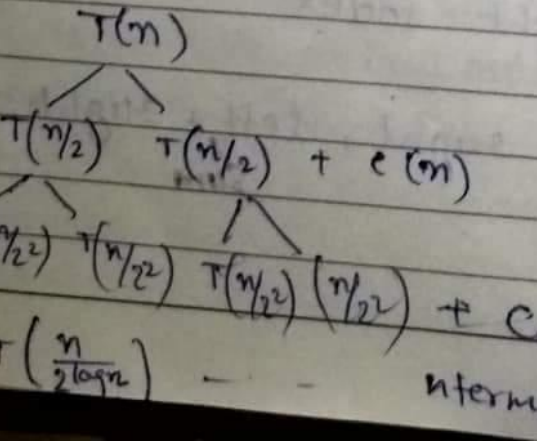
(sum 1, i_1, j_1) = Find_max_subarray (A, lo, mid)

(sum 2, i_2, j_2) = Find_max_subarray (A, mid+1, hi)

(sum 3, i_3, j_3) = Find_max_cross_subarray (A, lo, mid, hi)

if (sum 1 ≥ sum 2) && (sum 1 ≥ sum 3)
return (sum 1, i_1, j_1)
else if (sum 2 ≥ sum 1) && (sum 2 ≥ sum 3)
return (sum 2, i_2, j_2)
else return (sum 3, i_3, j_3)

Assume n is a power of 2
$T(n) = 2T(n/2) + \Theta(n)$



A.


```
Find_Max_Cross_Subarray (A, lo, mid, hi) {
// Calculate the max left subarray A[i ... mid]
    left_sum = A[mid] , left = mid
    Sum = left_sum
    for i = mid - 1 down to lo {
        sum = sum + A[i]
        if (sum > left_sum) {
            left_sum = sum
            left = i
        }
    }
}
```

Loop invariant :

At the begining of iteration i

1) left sum is the max subarray sum of A[i+1 ... mid]
2) This max subarray is A[left ... mid]

// At the end, we get
// (i) left sum + right sum = max cross subarray
(ii) left to right index.

SUNDAY 20

return (left, right, left_sum + right_sum)

FRIDAY • JANUARY

18

018-347
WEEK 03

DECEMBER 12

M	T	W	T	F	S	S
31					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

⇒ Find Maximum Subarray

Given array $A[1 \ldots n]$

A subarray is a contiguous segment $A[i \ldots j]$, $1 \le i \le j \le n$

Subarray sum of $A[i \ldots j]$:

$= A[i] + A[i+1] + \ldots + A[j]$

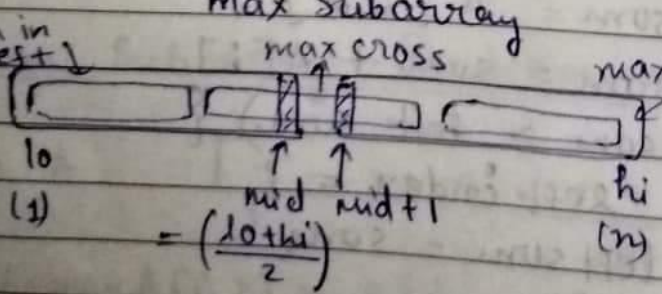
Problem: find Max Subarray

Find a subarray $A[i\,j]$ with the max sum among all subarrays of $A[i \ldots n]$

⇒ **Solution approach**:-

Divide and Conquer

Consider Max_Subarray (A, lo, hi)

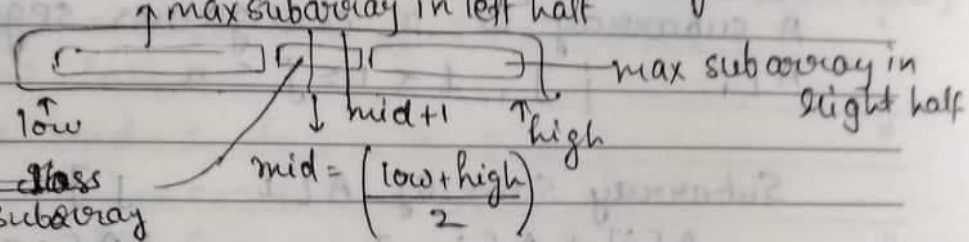


max subarray is either in (i) left half, or (ii) righ half or (iii) max cross.

2013

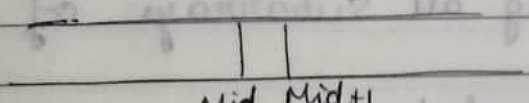
⇒ Problem :

Find a subarray with the maximum of sum.



for max subarray 3 case → Totally left, right or in btw.

⇒ Max cross subarray (A, lo., mid, hi)




```
left index = mid
left sum = A [mid]
for i = mid - 1, down to lo //  sum = A[i+1] - A[mid]
    sum = A [mid]
    sum = sum + A[i]
if ( sum > left sum) {
    left index = i
    left sum = sum
}

right index = mid + 1
right sum = A [mid + 1]
```

$$(bc + ad) = (a+b)(c+d) - (ac+bd)$$

Compute $x_L \times y_L$, $x_R \times y_R$
Compute $(x_L + x_R) \times (y_L + y_R)$
then $(x_L y_R + x_R y_L) = (x_L + x_R)(y_L + y_R) - (x_L y_L + x_R y_R)$

$$T(n) = 3\,T(n/2) + \Theta(n)$$

$$T(n) = cn + c\cdot\frac{3}{2}n + c\left(\frac{3}{2}\right)^2 n + \cdots + c\left(\frac{3}{2}\right) + 3^{\log n}\, T\left(\frac{n}{2^{\log n}}\right)$$

$$= \Theta\left(3^{\log n} = n^{\log_2 3}\right)$$

Algo complexity

$$\Theta\left(n^{\log 3} \cong n^{1.59}\right) \text{---}$$

⇒ Maximum Subarray Problem ⇐

Array A

1	5	-1	2	-2	-3	1	4

$n = 7$

Subarray $A[i,j] = A[i \cdots j]$

Subarrays sum :

$$A[i \cdots j] = \sum_{i=1}^{j} A[i]$$

Total no of Sub Arrays $= \binom{n}{2} + n$

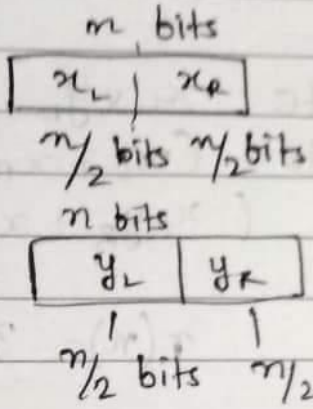
$$= \frac{n(n+1)}{2}$$

Time Taken = $\Theta(n^2)$

Divide and Conquer:

$x = 2^{n/2} x_L + x_R$

$y = 2^{n/2} y_L + y_R$



$$x \cdot y = (2^{n/2} x_L + x_R)(2^{n/2} y_L + y_R)$$
$$= 2^n x_L y_L + 2^{n/2} x_L y_R + 2^{n/2} x_R y_L + x_R y_R$$

if $T(n)$ is time to multiply 2 n bits No's then

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

$$T(n) = cn + 4(T(n/2))$$
$$= cn + 4 \cdot c(n/2) + 4^2 T(n/2^2)$$
$$= cn + 4\left(\frac{cn}{2}\right) + 4^2\left(\frac{cn}{2^2}\right) + \cdots + 4^{\log n} T\left(\frac{n}{2}\log n\right)$$
$$= c\{n + 2n + 2^2 n + \cdots 4^{\log n} \cdot c\}$$
$$= \Theta(n^2)$$

Gauss's Observation:

$$(a+bi)(c+di) = (ac - bd) + i(bc + ad)$$

MONDAY • JANUARY

14

014-351
WEEK 03

DECEMBER

M	T	W	T	F	S	S
31					1	
3	4	5	6	7	8	
10	11	12	13	14	15	
17	18	19	20	21	22	
24	25	26	27	28	29	

⇒ Multiplying n bit numbers →

$a = \bar{n} \; - \; - \; - \; \bar{2} \; \bar{1}$

bit position

$b = \underline{\quad} \; - \; - \; - \; - \; \underline{\quad} \; \underline{\quad}$

$n \qquad\qquad 2 \; 1$

Note: $(a+bi)(c+di) = (ac - bd) + i(bc + ad)$

$(bc + ad) = (a+b)(c+d) - ac - bd$

⇒ n-bit addition, multi :-

$x = 1\;1\;0\;1\;0\;1$

$y = 1\;1\;1\;0\;1\;0$

$1\;1\;0\;1\;1\;1\;1$

Elementary School Arithmatic for adding

If x and y are n bit numbers, this method computes $x+y$ on time $\Theta(n)$

Multiplication -

```
   n bit      1 1 0 1
   n bit    x 1 0 1 1
   ---------------------
   n bit      1 1 0 1
 n+1 bit    1 1 0 1
 n+2 bit  0 0 0 0
 n+3 bit  1 1 0 1
 2n bit   1 0 0 0 1 1 1 1
```

$= c(n) + c(n) + c(n) + \dots + cn$

$= \Theta(n \log n)$

Recursion tree Method :-

$T(n) = cn$ $= cn$

$T(n/2)$ $T(n/2)$ $\frac{cn}{2}$ $\frac{cn}{2}$

$T(n/2^2)$ $T(n/2^2)$ $T(n/2^2)$ $T(n/2^2)$

$T(n/2^3)$ $T(n/2^3)$ $T(n/2^3)$ $T(n/2^3)$ ...

⇒ Binary Search :-

Sorted Array | 1 | 3 | 4 | 7 | 8 | 10 |

Question : Does 11 belong to Array ?

divide and check

Time Complexity : $\Theta(\log n)$

give $\Theta(n \log n)$ Algorithm

Exercise : Given a list of numbers S, Given another no x. Does there exist two numbers $a_i, a_j \in S$ s.t. $a_i + a_j = x$?

| a_1 | a_2 | — | — | — | a_n |

2013

j=1; for k = p to r {

if L[i] ≤ R[j] {

A[k] = L[i]

i = i+1

}

else { // L[i] > R[j]

A[k] = R[j]

j = j+1

}

Time Complexity

if r-p+1 = n

line 1 to 19 takes Θ(n) time

No of times for loop runs is r-p+1 = n

line 20 to 30 Θ(n) time

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

$T(n) = cn = cn$

$T(n/2)\ T(n/2) \quad 2T(n/2)$

$= cn + 2c(n/2) + 2^2 T(n/2^2)$

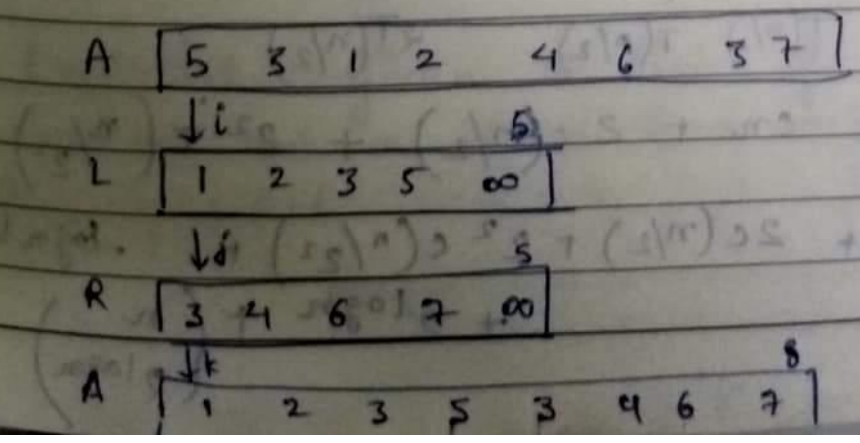
$T(n) = cn + 2c(n/2) + 2^2 c(n/2^2) + \ldots \log n$ time

$+ 2^{\log n} T(n^{\ldots})$

```
Merge-Sort (A, p, r) {
// Sorts A [p ..... r]   // top level call MergeSort (A, 1, n)
    if (r > p) {
        q = (r+p)/2 ;
        MergeSort (A, p, q)
        Merge-Sort (A, q+1, r)
        Merge (A, p, q, r)
    }

Merge (A, p, q, r) {
// A[p ... q] is sorted
// A [q+1 ..... r] is sorted
// let L [1 ..... q-p+2] be a new array
// R [1 ..... r-q+1] be a new array

Copy A[p .. q] into L [1 ... q-p+1]
     L [q-p+2] = ∞
Copy A [q+1 .. r] into R [1 ... r-q]
     R [r-q+1] = ∞
```



WEDNESDAY • JANUARY

09

009-356
WEEK 02

DECEMBER 12

M	T	W	T	F	S	S
31					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

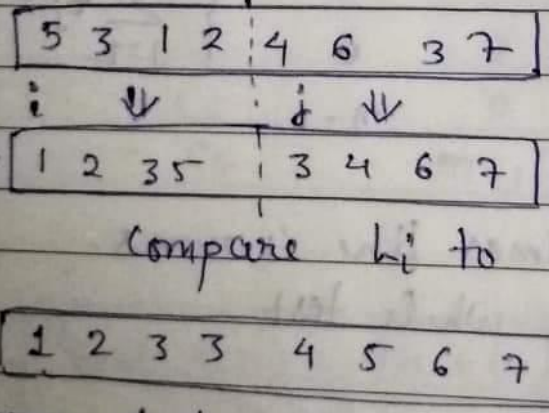
$\exists\ n_0, c_1, c_2 \ .\ \forall\ n \geq n_0$

$$c_2 g(n) \leq T(n) \leq c_1 g(n)$$

1/8/18

Divide and Conquer Approach to Algorithm Design :-

Problem : Given a list of n numbers . sort them in ascending .



Divide into 2 parts and sort them seperately

(For array of size n)

∴ By this Algorithm time taken in sorting is

$$T(n) = T(n/2) + T(n/2) + \Theta(n)$$

$$T(n) = 2T(n/2) + \Theta(n) \quad cn$$

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

$\exists\ c_1, c_2, n_0\ \forall\ n \geq n_0$

eg: $2n \leq (3n-5) = T(n) \leq 3n$

WORST CASE TIME COMPLEXITY

	c_1	$n-1$
	c_2	$n-1$
	c_3	$n-1$
	c_4	$\sum_{j=2}^{n} t_j$
	c_5	$\left.\begin{matrix}\\ \\ \end{matrix}\right\} \left(\sum_{j=2}^{n} t_j\right) - 1$
	c_6	
	c_7	$n-1$

t_j = No of times line 4 runs.
// inner while test

upper bound

$t_j \leq j$

$$c_4 \sum_{2}^{n} j + (c_5 + c_6) \sum_{2}^{n} (j-1) = c_4 n\left(\frac{n+1}{2}\right) - 1 + (c_5 + c_6)\frac{n(n-1)}{2}$$

$T_{worst}(n) = An^2 + Bn + C$

$T_{worst}(n) = \Theta(n^2)$

→ for loops take constant time.

MONDAY • JANUARY

07

007-358
WEEK 02

A[i+1] = key

RAM model of computation

→ Best Case Time

for j=2 to n.	Time	No of time it repeats
key = A[j]	c_1	$n-1$
i = j-1	c_2	$n-1$
while( ){	c_3	$n-1$
A[i+1] = A[i]	c_4	$n-1$
i = i-1 }	c_5	0
	c_6	0
A[i+1] = key	c_7	$n-1$

$$T_{Best}(n) = an + b$$
$$\propto n$$

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

⇒ Assymptotically as $n \to \infty$
Thr exists c_1 & c_2 s.t. $T(n)$ is sandwitch btw $c_1 n$ & $c_2 n$

$$c_2 n \leq T(n) \leq c_1 n$$

⇒

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

As $n \to \infty$; $\forall$ c_1 & c_2

$$c_2 g(n) \leq T(n) \leq$$

30/7/18 Lecture 1

JANUARY • SATURDAY

005-360
WEEK 01

05

ALGORITHMS.

→ Gets applied in all walks of life

→ In all branches of science & Eng.

Insertion Sort :

Consider an Array

1	2	3	4	5
6	2	3	5	1

$n = 5$.

→ $j = 2$

6	2	3	5	1

key = 2

2	6	3	5	1

$j = 3$

2	6	3	5	1

key = 3 = A[j]

2	3	6	5	1

and so on.

⇒ Insertion_Sort (A, n) {

Sort A[1 ... n]

for j = 2 to n {

key = A[j]

i = j - 1

while i ≥ 1 and (A[i] > key) {

A[i+1] = A[i]

i = i - 1 }

SUNDAY 06

2013