Lecture - 1

Worst Case : Upper Bound
Best Case : Lower Bound
Average Case : Prediction

$T(n)$ is $O(f(n))$ if $c>0, n_0 \geq 0$ such that $T(n) \leq c \cdot f(n)$ for all $n \geq n_0$. $O() \leq$

$T(n)$ is $\Omega(f(n))$ if $c>0, n_0 \geq 0$ such that $T(n) \geq c \cdot f(n)$ $\forall$ $n \geq n_0$. $\Omega() \geq$

if $c_1 f(n) \leq T(n) \leq c_2 f(n)$
$T(n)$ is $\Theta f(n)$ $c_1, c_2 > 0; n \geq n_0$

$1 \leq \log n \leq n \leq n\log n \leq n^2 \leq n! \leq 2^n \leq n^n$

$o() \to <$, $\omega() \to >$

Theorem :→ $f(n) = \Theta(g(n))$ if $f(n) = O(g(n))$ & $f(n) = \Omega(g(n))$

$f(n) = \Theta(g(n)), g(n) = \Theta(h(n)) \Rightarrow f(n) = \Theta(h(n))$
Transitivity

Reflexivity
$f(n) = \Theta(f(n))$. same for O, Ω

Symmetry : $f(n) = \Theta(g(n))$ only if $g(n) = \Theta f(n)$
Transpose : $f(n) = O(g(n))$ " " $g(n) = \Omega f(n)$

Masters Theorem

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

$a \geq 1, b > 1$ & $f(n) > 0$

Case 1 : if $f(n) = O(n^{\log_b a - \epsilon})$, then $T(n) = \Theta(n^{\log_b a})$

Case 2 : if $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$

Case 3 : if $f(n) = \Omega(n^{\log_b a + \epsilon})$ $\epsilon > 0$, & if $af(n/b) \leq cf(n)$ for some $c < 1$ & all sufficiently large n, $T(n) = \Theta(f(n))$.

$\lg^k n = (\lg n)^k$, $\lg\lg n = \lg(\lg n)$

$\log n^y = y \log x$, $\log xy = \log x + \log y$

$\log \frac{x}{y} = \log x - \log y$, $a^{\log_b x} = x^{\log_b a}$

$\log_b x = \log_a x / \log_a b$

$\boxed{\text{Lecture-2}}$

A.P $\Sigma = \frac{n(n+1)}{2}$; G.P $\sum_{k=0}^{n} x^k = \frac{x^{(n+1)} - 1}{x - 1}$

G.P $\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}$ $\{|x| < 1\}$

$\boxed{\text{Summation Series}}$

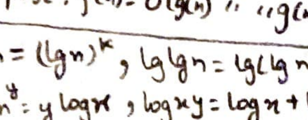
$\sum_{k=1}^{n} \frac{1}{k} = \ln n$; $\sum_{k=1}^{n} \lg k = n \log n$

$\sum_{k=1}^{n} k^p = 1^p + 2^p + \dots n^p = \frac{n^{p+1}}{p+1}$

Recurrence Time

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

Tree Method



Ht. $= \log_2 n$ $\quad \Sigma = n \times Ht$

RunTime $= n \times \log n = O(n\log n)$

Substitution Method :→

Prove $T(n) = \Theta(n\log n)$, $O(n\lg n)$

Assume $T(k) \leq c_1 \cdot n \log n$ $k < n$

Given $T(n) = 2T(n/2) + c_2 n$

$\Rightarrow T(n) \leq c_1 \times 2 \times \frac{n}{2} \log \frac{n}{2} + c_2 n$

$c_1 n\log n - c_1 n \log 2 + c_2 n \leq c_1 n \log n$

$n(c_2 - c_1 \log 2) \leq 0$, $c_2 \leq c_1 \log 2$

$\Rightarrow$ if $c_1 \geq c_2$ then $T(n) \leq c_1 n \log n$

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

$T(k) = O(n^2 - n)$, $T(n/2) \leq c_1 \left(\frac{n}{2}\right)^2 - \frac{n}{2}$

$T(n) \leq 4 \times c_1 \frac{(n^2 - 2n)}{4} + n$

$T(n) \leq c_1 \frac{(n^2 - 2n)}{4} + n$

$T(n) \leq c_1 n^2/4 + n(1 - \frac{c_1}{2})$

$1 - c_1/2 \leq 0$, $c_1 \geq 2$ then $T(n) = O(n^2 - n)$

$\leq c_1(n^2 - n) \Rightarrow T(n) = O(n^2 - n)$

Sorting

Bubble Sort
N = len (A)
while (A not sorted)
for j from 1 to N-1
if A[j-1] > A[j]
swap(A[j-1], A[j])
end

8 4 6 9 2 3
i=1 … j
8 4 6 9 2 3
i=1 … j
8 4 6 2 9 3
i=1 … j
2 8 4 6 9 3
i=2 … j

→ Sorts in Place
Best Case $\Theta(n)$, Worst $\Theta(n^2)$

Insertion Sort

Insertion-Sort(A)
for j = 2 to A.length
Key = A[j]
i = j - 1
while i > 0 & A[i] > Key
A[i+1] = A[i]
i = i - 1
A[i+1] = Key

5 2 4 6 1
2 5 4 6 1
2 4 5 6 1
2 4 5 6 1
1 2 4 5 6

Avg & Worst: $\Theta(n^2)$
Best Case $O(n)$

Loop Invariant :→ A property used to prove correctness of algorithm. Define a value that doesn't changes before & after the algo is run.

For Insertion :→ A[1 … j-1], this array remains same in size, just that it becomes sorted after algo.

Steps :→ Initialization, j = 2
A[1] = A[1] after algo.
Maintainence, Inductive step.
Termination: After for loop ends j = n+1 ⇒ j-1 = n
Before loop j-1 = n
Hence Algo correct

⇒ $n^2/2$ Comparisons
⇒ In place sorting

Runtime $= n + C \sum_{j=2}^{n} t_j + c_1 \sum_{j=2}^{n} (t_j - 1)$
$+ c_2 \sum_{j=2}^{n} (t_j - 1) + n$

Best Case $= n + (n-1) + n$

Worst Case $\sum_{j=2}^{n} t_j = (n)(n-1) + (n)(n-1) = \Theta(n^2)$

Divide & Conquer

Divide & Conquer :→ Divide the problem into sub problems till problem size becomes very small. Conquer → Combine the divided small problems to get solution. Maximum Sub Array Problem. $\Theta(n\log n)$

Find-Max-Cross-Subarray(A, low, mid, high)
left-sum = -∞
sum = 0
for i = mid to low
sum = sum + A[i]
if sum > left-sum then
left-sum = sum
max-left = i
right-sum = -∞
sum = 0
for j = mid+1 to high
sum = sum + A[j]
if sum > sum + A[j]
if sum > right-sum
then right-sum = sum
max-right = j
return (max-left, max-right, left-sum + right-sum)

Kadane

initialize :
max_so_far = INT_MIN
max_end_here = 0
Loop for each element of arr
max_end_here = max_end_here + a[i]
if (max_so_far < max_end_here)
max_so_far = max_end_here
if (max_end_here < 0)
max_end_here = 0
return max_so_far

Merge-Sort

Merge-Sort :→ Doesn't sort in place

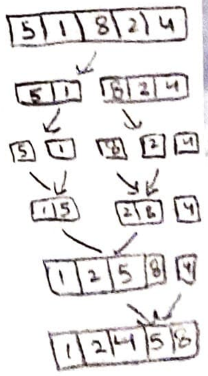
Divide time $= \Theta(1)$
Conquer time $= 2T(n/2)$
Combine $= \Theta(n)$

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

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

Disadvantage ≈ n

A(p, r)
Div.
A(p, q), A(q+1, r)
Combine
A(p, r)



Stable

Selection Sort

Selection Sort :→ $O(n^2)$

SelSort (A[n])
for (i = 1 to n (exclu. n))
for (j = i+1 to n (inclu. n))
if (arr[i] > arr[j])
{ min = arr[j];
arr[j] = arr[i];
arr[i] = min;
}
end
end

Sorted Array
In place, Not stable

Heap-Sort: Sorts in place & worst case time $= O(n \lg n)$

Binary-Tree → Full binary tree, complete binary tree

Full :→ BT in which each node is either a leaf or has a exact degree of 2.

Complete :→ BT in which all leaves are on same level and are internal nodes have degree 2.

Depth $= \lfloor \lg n \rfloor$, Total nodes $= 2^{d+1} - 1$ $\{d = \text{depth}\}$.

Nodes at level 'l' $= 2^l$.

Max-Heap :→ Complete Binary Tree where parent > child

Root is the max element.

Min-Heap :→ Complete BT where parent < child

Root is min element. Root $A[1]$

Node i: $A[i]$, left child $i = A[2i]$, right child i: $A[2i+1]$

Parent of node i $= A[\lfloor i/2 \rfloor]$

Heap size[A] < length[A]

New nodes always inserted at bottom

Root is deleted, then max/min heap property is restored

```
Max-Heapify (A, i)
{
    l = left(i); r = Right(i);
    if (l ≤ heap-size(A) && A[l] > A[i])
        largest = l;
    else
        largest = i;
    if (r <= heap-size(A) && A[r] > A[largest])
        largest = r;
    if (largest != i)
        swap (A, i, largest);
        Heapify (A, largest);
}
```

$\boxed{\text{Run Time } O(\lg n)}$, No. of comp $= 2h$

$h = \lfloor \log n \rfloor$. $T_n \le T(2n/3) + \Theta(1)$

```
BUILD-MAX-HEAP
n = length [A]
i ← ⌊n/2⌋ downto 1
do MAX-HEAPIFY (A, i, n) → O(lg n)     } O(n)
```

Total Time $= O(n \lg n)$ → Correct Upper Bound

Not a tight bound. $\boxed{\text{Tighter Bound } O(n)}$

```
→ BUILD-MAX-HEAP(A)                  O(n)
→ for i ← length [A] to 2                      }
      do exchange A[1] ↔ A[i]                  } n-1 times
         MAX-HEAPIFY (A, 1, i-1)     O(lg n)   }
```

$O(n) + (n-1)\, O(\lg n) = \boxed{O(n \lg n)}$ ← Heap Sort

Priority Queues :→

if Smaller number has higher priority use a min Heap. $O(\log n)$

if larger number has higher priority use a max Heap. $O(\log n)$

Higher priority element deleted first

INSERT(S, x): Insert x into set S.

EXTRACT-MAX(S) :- Remove highest priority element from S.

MAXIMUM (S) :- Return element with largest key

Increase Key(S, x, k): Increase value of element x's key to K {Assume $k > x$}.

```
HEAP-EXTRACT-MAX (A, n)
if n<1
then error "heap underflow".
max ← A[1]
A[1] ← A[n]
MAX-HEAPIFY(A, 1, n-1)
return max
```

$\boxed{\text{Run Time } O(\lg n)}$

```
HEAP-INCREASE-Key (A, i, key)
if Key < A[i]
then error "new key is smaller than curr".
A[i] ← key
while i>1 & A[parent(i)] < A[i]
   do exchange A[i] ↔ A[parent(i)]
      i ← PARENT(i)
```

$\boxed{\text{Run time}: O(\lg n).}$

```
MAX-HEAP-INSERT :→ (A, key, n)
heap-size [A] ← n+1
A [n+1] ← -∞
HEAP-INCREASE-KEY (A, n+1, Key)
```

$\boxed{\text{Run Time } O(\lg n)}$

Counting sort: Overall Time $= \Theta(n+r)$

used when $r = O(n)$ ⇒ Run Time is $\Theta(n)$

Counting is stable, Not in place.

Example A, B, C

A = orig. Arr, C → Range Arr. B → final arr

C → 1st freq → Cumm freq → Redu freq.

```
Counting-Sort (A, B, n, r)
for i ← 0 to r
   do C[i] ← 0
for j ← 1 to n
   do C[A[j]] ← C[A[j]]+1 → C[i] contains no. of element equal to i
for i ← 1 to r
   do C[i] ← C[i] + C[i-1] → C[i] contains no. of element ≤ i
for j ← n to 1
do B[C[A[j]]] ← A[j]
   C[A[j]] ← C[A[j]] - 1
```

Quick Sort :- (Hoare)

→ sorts in place
→ avg case : $O(n \lg n)$
→ worst case : $O(n^2)$

Alg : QUICKSORT (A, p, r)
  if p < r
   then
    q ← PARTITION (A, p, r)
    Quicksort (A, p, q)
    " " (A, q+1, r)

Initially : p = 1, r = n
Recurrence : $T(n) = T(q) + T(n-q) + f(n)$ → depends on partitⁿ
$f(n) = n$

1. HOARE'S PARTITION

Alg. PARTITION (A, p, r)
  x ← A[p]
  i ← p-1
  j ← r+1
  while TRUE
   do repeat j ← j-1
    until A[j] ≤ x
   do repeat i ← i+1
    until A[i] ≥ x
   if i < j
    then A[i] ↔ A[j]
   else
    return j

*Running Time : $\Theta(n)$
$n = r-p+1$

Example : pivot = 5
① 5 | 3 | 2 | 6 | 4 | 1 | 3 | 7   (i, j)
② 5 | 3 | 2 | 6 | 4 | 1 | 3 | 7   (i, j)
③ 3 | 3 | 2 | 6 | 4 | 1 | 5 | 7   (i, j)
④ 3 | 3 | 2 | 6 | 4 | 1 | 5 | 7   (i, j)
⑤ 3 | 3 | 2 | 1 | 4 | 6 | 5 | 7   (i, j)
⑥ 3 | 3 | 2 | 1 | 4 | 6 | 5 | 7   (i, j)
A [p..q]   A[q+1..r]

QS worst case = $\Theta(n^2)$
best " = $\Theta(n \lg n)$

2. LOMUTO'S PARTITⁿ

Alg. PARTITⁿ (A, p, r)
  x = A(r)
  i = p-1
  for j = p to r-1
   if A[j] ≤ x
    i = i+1
    A[i] ↔ A[j]
  A[i+1] ↔ A[r]
  return (i+1)

Example Lomuto :
① 2 | 8 | 7 | 1 | 3 | 5 | 6 | 4   (p, i, j, r)
② 2 | 8 | 7 | 1 | 3 | 5 | 6 | 4
③ 2 | 8 | 7 | 1 | 3 | 5 | 6 | 4
④ 2 | 8 | 7 | 1 | 3 | 5 | 6 | 4
⑤ 2 | 1 | 7 | 8 | 3 | 5 | 6 | 4
⑥ 2 | 1 | 3 | 8 | 7 | 5 | 6 | 4
⑦ 2 | 1 | 3 | 8 | 7 | 5 | 6 | 4
⑧ 2 | 1 | 3 | 8 | 7 | 5 | 6 | 4
⑨ 2 | 1 | 3 | 4 | 7 | 5 | 6 | 8

Lomuto Alg : same as Hoares
but (A, p, q) ↔ (A, p, q-1)

3. RANDOMIZED QS (using Lomuto's)

Alg : R-QS (A, p, r)
  if p < r
   q ← R..PART(A, p, r)
   R-QS (A, p, q-1)
   R-QS (A, q+1, r)

partⁿ called atmost n times

Partition :
Alg : Randomized Part.. (A, p, r)
  x ← A[r]
  i ← p-1 } O(1) - constant
  for j ← p to r-1
   do if A[j] ≤ x
    i ← i+1
    A[i] ↔ A[j]
  } # comparisⁿˢ = X (unknown)
  A[i+1] ↔ A[r] } O(1) - const
  return i+1
* Total work = C + X

AVG CASE ANALYSIS OF QS (when pivot is chosen randomly)

- Total # comparisⁿˢ in all calls to partitⁿ = X (rand. var.)
- Total work = $O(n + X)$
- Need to estimate $E(X)$
- $E(X)$ = avg # comparisⁿˢ
- z_i = ith smallest element
- set $z_{ij} = \{z_i, z_{i+1}, \dots z_j\}$
- $x_{ij} = I\{z_i$ compared to $z_j\}$
- Total # comparisⁿ :
$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}$$

i → n-1
i+1 → n

- $E(X) = E\left[\sum\sum x_{ij}\right]$
by linearity of exp
$= \sum\sum E(X_{ij})$ ← ind. var
$= \sum\sum Pr\{z_i \text{ comp. to } z_j\}$

Case 1 : Pivot x chosen such as $z_i < x < z_j$
→ z_i & z_j will never be compared
Case 2 : z_i or z_j is the pivot → z_i & z_j will be compared
→ only if one of them is chosen as pivot before any other element in $\{z_i .. z_j\}$

→ $E(X) = \sum\sum Pr(\dots)$
→ $Pr\{z_i \text{ comp. } z_j\} = Pr\{z_i$ is the 1st pivot chosen from $z_{ij}\}$
$Pr\{z_i$ is the 1st ... $\}$ + $Pr\{z_j \dots\}$
$= \frac{1}{j-i+1} + \frac{1}{j-i+1} = 2/(j-i+1)$

→ $E(X) = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} 2/(j-i+1)$
(set $j - i = k$)
$= \sum_{i=1}^{n-1}\sum_{k=1}^{n-i} 2/(k+1) < \sum_{i=1}^{n-1}\sum_{k=1}^{n} 2/k$
$= \sum_{i=1}^{n-1} O(\lg n)$ (harmonic)
→ $E(X) = O(n \lg n)$

ORDER STATS : ith order stat = ith smallest el.

RANDOMIZED SELECTION :
↳ randomized partitⁿ on only one sub-array
→ run-time : $O(n)$

Alg : Rand Select (A, p, r, i)
  if (p == r) : return A[p]
  q = Rand Part (A, p, r)
  k = q - p + 1
  if (i == k) : return A[q] (pivot)
  if (i < k) :
   return Randselect (A, p, q-1, i)
  else
   return " (A, q+1, r, i-k)

Worst case : 0 : n-1 partitⁿ
$T(n) = O(n^2)$

Best case : 9 : 1 partitⁿ
$T(n) = O(n)$

Avg Case : for upper bound assume ith element always falls in larger side
$T(n) \le \frac{1}{n}\sum_{k=0}^{n-1} T(\max(k, n-k-1)) + \Theta(n)$
$\le \frac{2}{n}\sum_{k=n/2}^{n-1} T(k) + \Theta(n)$ (show $T(n) = O(n)$ by substitutⁿ)

- Assume $T(k) \le ck$ ($k < n$) for large c :
$T(n) \le \frac{2}{n}\sum_{k=n/2}^{n-1} T(k) + \Theta(n) \le \frac{2}{n}\sum ck + \Theta(n)$
$= \frac{2c}{n}\left(\sum_{k=1}^{n-1} k - \sum_{k=1}^{n/2-1} k\right) + \Theta(n)$
$= \frac{2c}{n}\left(\frac{1}{2}(n-1)n - \frac{1}{2}\left(\frac{n}{2}-1\right)\frac{n}{2}\right) + \Theta(n)$
$= c(n-1) - \frac{c}{2}\left(\frac{n}{2}-1\right) + \Theta(n)$
→ Assume $T(n) \le cn$ for large c :
$T(n) \le cn - c - \frac{cn}{4} + \frac{c}{2} + \Theta(n)$
$= cn - \frac{cn}{4} - \frac{c}{2} + \Theta(n)$
$= cn - \left[\frac{cn}{4} + \frac{c}{2} + \Theta(n)\right]$
⇒ $\boxed{T(n) \le cn}$ (if c is big)

Radix Sort :→ if Key = 15

$Key_{10} = 15$, $d = 2$, $k = 10$ $0 \le n_i \le 9$

$key_2 = 1111$, $d = 4$, $k = 2$ $0 \le n_i \le 1$

Assumption :→ $d = \Theta(1)$, $k = O(n)$

→ First sort least significant digit using stable sort

→ Continue sorting on next least significant digit until all digits have been sorted

Run Time : $O(d(n+k))$

326	751	608	326
453	453	326	435
608	835	835	453
835 →	435 →	435 →	608
751	326	751	751
435	608	453	835

Direct Addressing Op :-) $T[n] = n$. So no. of slots is high.

if there are no elements with key k in set, $T[k]$ is empty.

Alg : Direct-Address Search (T, K)
return T[K].

Alg : Direct Address Insert (T, n)
T[key[n]] ← n

Alg : Direct Address Delete (T, n)
T [key [n]] ← NIL

when key space |k| is less than universe |U|. Then use Hash Table.

Key k is hashed to Table T at T[k]. $\boxed{\text{Avg. search } O(1)}$

Two key want same slot in Hash Table → collision.

Resolution → Chaining

Chained - Hash - Insert (T, n) → insert n at head of list T[h(key[n])]

Would take addn. search to see it n had already been added.

Chained Hash Delete (T, n) → delete n from list T[h(key[n])]

Need to find element to be deleted

Chained Hash Search (T, k) → search for element with key K in T[h(k)]

Run time ∝ length of list in slot h(K)

Avg value of $n_j = \alpha = \frac{n}{m}$

→ Avg. length of list at all slots = α.

n = total elements

m = total slots.

Prob. of collision = 1/m

Proof :-) Search unsuccessful for any key k, need to search till end of list T[h(k)]

Expected Length of list = α.

Total Time = $O(1) + \alpha = \Theta(1+\alpha)$

Successful Search :→ $\Theta(1+\alpha)$

Expected no. of elements examined in a successful search is

$1 + \alpha/2 - \alpha/2n$

Division Method :→ $h(k) = k \% m$

Adv. Fast req. only one operation.

Disadv → Certain m are bad like power of 2

Mult. Method :→

$h(k) = \lfloor m(\underbrace{kA \bmod 1}_{\text{Fractional part of } kA}) \rfloor$

$0 < A < 1$

$= kA - \lfloor kA \rfloor$

Slower → Disadv

Adv → m is not critical

Universal Hash function

$Z_p = \{0, 1, .. p-1\}$

$Z_p^* = \{1, 2 ... p-1\}$.

$h_{a,b}(k) = ((ak+b) \% p) \% m$

$a \in Z_p^*$, $b \in Z_p$

chance of collision ≤ 1/m of a collision if h(k) & h(l) were selected randomly

BST.

Inorder Tree Walk :

if n ≠ NIL

Inorder Tree - Walk (left [n])

print key [n] $\boxed{\Theta(n)}$

Inorder TREE Walk (right [n])

Tree - Search (n, k)

if n = nil or k = key [n]
return n

if k < key [n] O(h) h - ht. of tree

then return TREE - SEARCH (left [n], k)

else return TREE - " (right [n], k)

Tree - Minimum (n)

while left [n] ≠ NIL
do n ← left [n] O(h)

return n

Tree - Max (n) → Same as above replace left with right.

Successor (n) :→ Smallest no. greater than key [n].

(-I → right (n) is not empty

successor (n) = min in right (n)

(-II → TREE - Successor (n)

if right [n] ≠ nil
return TREE - MINIMUM (right [n])

y ← P [n]

while y ≠ NIL & n = right [y]
do n ← y
y ← P [y] O(h)

return y h → ht. of tree

PREDECESSOR :→ opp. of successor.

C-II Go up tree until current node is right child. Pred(n) is parent of current node.

Can't go further : n is smallest

Tree Insertion

y ← NIL

n ← root [T]

while n ≠ NIL
do y ← n
if key [z] < key [n]
then n ← left [n]
else n ← right [n]

P[z] ← y

if y = NIL $\boxed{O(h)}$

then root [T] ← z

else if key [z] < key [y]
then [y] ← z
else Right [y] ← z

Deletion

C-I just del.

C-II just del & replace

C-III del & replace with successor

→ only ... pivot before any ... in {zi .. zj}