AA Mod 2

Page No.
Date

- Insertion Sort

i=1=8 Example :-
j=2=2

	8	2	[4	9	3	6] not sorted
	2	8	4	9	3	6
	2	4	8	9	3	6
sorted	[2	4	8	9]	3	6
	2	3	4	8	9	6
	2	3	4	6	8	9

as 1st index is alr sorted
Start from 2nd element
Compare it with the elements on the left, switch positions.
Move to next index, repeat the process.

```
Insertion-Sort (A,n)                  // n: no. of elements
  for j ← 2 to n                      // j = 2 to n, iteration loop
      do key ← A[j]                   // A[j]: key : compare it with elements on left
         i ← j-1                      // index of last element in sorted array
         while i > 0 and A[i] > key   // sorted should be > key
               do A[i+1] ← A[i]       // shift right by 1
                  i ← i-1             // move back, iterate
         A[i+1] = key                 // after finding right spot, place key in
                                         correct position.
```

1>0, 8>2 :
A[2] = A[1]
i=0, loop broken
shift 8 to index 2

j : current pos^n
i = j-1 elements

1 Best Case Time Complexity
[Ascending Order]
comparing (n-1) times
O(n) : ~~upper~~ ~~bound~~ ~~st~~ comparison
swapping : O(1)

10 20 30

Comparison	Swap	Total = 2
0	0	n = 3
1	0	(n-1)
1	0	

2 Worst Case [Descending order]

$$\frac{n(n-1)}{2} = \frac{n^2-n}{2} \quad [AP]$$

O(n^2) : comparison, swapping.

$$T(n) = \sum_{j=2}^{n} \Theta(j) = \Theta(n^2)$$

30 20 10

Comp.	Swap	Series :
0	0	n(n-1)/2
1	1	
2	2	
⋮		
(n-1)	(n-1)	

Page No.
Date / /

3 Average case complexity

$$T(n) = \sum_{j=2}^{n} \theta(j/2) = \theta(n^2)$$

Insertion sort is stable [ex. posn of duplicate values doesn't get swapped]
Insertion sort is in place [no extra space]
Fast only for less n.

- Merge Sort [Divide and Conquer]

Example :- 6 4 2 1 9 8 3 5

Divide :- | 6 | 4 | 2 | 1 | , | 9 | 8 | 3 | 5 |

| 6 | 4 | | 2 | 1 | | 9 | 8 | | 3 | 5 |

| 6 | | 4 | | 2 | | 1 | | 9 | | 8 | | 3 | | 5 |

| 4 | 6 | | 1 | 2 | | 8 | 9 | | 3 | 5 |

Divide and merge

| 1 | 2 | 4 | 6 | | 3 | 5 | 8 | 9 |

| 1 | 2 | 3 | 4 | 5 | 6 | 8 | 9 |

$T(n)$ MERGE-SORT $A[1 \ldots n]$

$\theta(1)$ if $n=1$, done.

$2T(n/2)$ → 2 lists — Recursively sort $A[1 \ldots (n/2)]$ and $A[(n/2) \ldots n]$ // list divided into half and sorted seperately.

$\theta(n)$ Merge the 2 sorted lists [We might be using a loop to divide it as well as merge? not sure]

$$T(n) = \begin{cases} \theta(1) & \text{if } n=1 \\ 2T(n/2)+\theta(n) & \text{if } n>1 \end{cases}$$

[use any method to solve this] $T(n) = \theta(n \log n)$

$\theta(n \log n)$ grows more slowly than $\theta(n^2)$

It beats insertion sort in worst case scenario. [↑ input but better performance by Merge Sort]

- **Divide and Conquer**

Divide problem into several sub-problems, solve them, then combine them.

$n \to n/2$ — soln, $n/2$ — soln $\to$ total soln

$n, n/2$: sizes

$a_0 + a_1 + \dots + a_{n-1}$

$a_0 + \dots + a_{(n/2)-1}$ $\quad$ $a_{(n/2)} + \dots + a_{n-1}$

No. of additions is same as in brute force, needs stack for recursion

May not be useful sometimes (not working)

$T(n) = aT(n/b) + f(n)$ $\quad$ Easier to solve using Master method

$$T(n) = \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log n) & \text{if } a = b^d \\ \Theta(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

$a \geq 1 \quad b > 1$

$f(n) \in \Theta(n^d)$

For adding n numbers with this, $A(n) = 2A(n/2) + 1$

1. $T(n) = 3T(n/2) + n$ $\quad$ $T(n) \in \Theta(n^{\log_2 3})$ $\quad$ as $f(n) \in \Theta(n^1)$

   $a = 3 > b^d = 2^1$ $\quad$ $= \Theta(n^{1.5850})$

2. $T(n) = 3T(n/2) + n^2$ $\quad$ $T(n) \in \Theta(n^d)$

   $a = 3 < b^d = 2^2 = 4$ $\quad$ $\in \Theta(n^2)$

3. $T(n) = 4T(n/2) + n^2$ $\quad$ $T(n) \in \Theta(n^d \log n)$

   $a = 4 \quad b^d = 2^2 = 4$ $\quad$ $\in \Theta(n^2 \log n)$

- **Merge-Sort**

ALGORITHM Merge-Sort (A[0 ... n-1]) // sorting array by recursive merge sort
if $n > a$
 copy A[0 ... ⌊n/2⌋-1] to B[0 ... ⌊n/2⌋-1]
 copy A[n/2 ... n-1] to C[0 ... ⌈n/2⌉-1]
 Mergesort (B[0 ... (n/2)-1])
 Merge sort (C[0 ... (n/2)-1])
 Merge (B, C, A)

Algorithm Merge (B[0 ... p-1], C[0 ... q-1], A[0 ... p+q-1]) sorted
 i ← 0; j ← 0; k ← 0; // set i, k, j to 0
 while i < p and j < q do (p: length of B, q: length of C)
  if B[i] ≤ C[j]
   A[k] ← B[i]; i ← i+1 // place B[i] in merged array at k and increment i

Page No.
Date

else
    A[k] ← C[j]; j ← j+1
  k ← k+1
if i=p   // all elements in B processed
    copy C[j...q-1] to A[k...p+q-1]
else
    copy B[i...p-1] to A[k...p+q-1]

Time efficiency of merge sort : Θ(n log n) : Worst case, same for average case
    input size : $n = 2^m$
It is stable but quicksort and heapsort aren't
More complex as it needs linear amount of extra memory.

• Quick Sort
  Divide and conquer, using pivot. leftmost index, rightmost index
  Algorithm quickSort (array, left, right)
    if (left < right)
      pivot Index ← partition (array, left, right) //pivot determined by partition
      quick Sort (array, left, pivot Index - 1) //apply quicksort on left
      quick Sort (array, pivot Index, right) and right side divided by pivot
  partition (array, left, right)
    set right as pivot index   // rightmost as pivot
    store Index ← left - 1   //keeps track of position where elements < pivot will be placed
    for i ← left +1 to right //iterate from left +1 to right
    if element[i] < pivotElement
      swap element [i] and element [store Index] should be placed
after each swap Store Index ++ //to keep track of position where next element < pivot
      swap pivot Element and element [store Index +1] //pivot placed in correct position, left < right
  return Store Index +1
    → represents index where pivot is positioned

Page No.
Date

Another Algorithm (from notes):

quicksort: Algorithm Quicksort (A [l ... r])
if l < r // if l < r, we need to sort
    s ← HoarePartition (A [l ... r]) // s split posⁿ
    Quicksort (A [l .. s-1]) } // exclude s
    Quicksort (A ~~[l s+1 r]~~ [s+1 ... r]) }

Partition:
Algorithm Hoare Partition (A [l ... r]) // represent subarray
p ← A[l] // p = pivot as index l
i ← l ; j ← r+1 : i leftmost j rightmost +1 posⁿ of subarray
repeat {
    repeat i ← i+1 until A[i] ≥ p // left ≤ p
    repeat j ← j-1 until A[j] ≤ p // right ≥ p
    swap (A[i], A[j]) } // greater on right & lesser on left
until i ≥ j // loop stops
swap (A[i], A[j]) // undo last swap when i ≥ j
swap (A[i], A[j]) // to place pivot in correct position
return j

| p | | → | ≤ p | p | ≥ p |

i → ← j
| p | |

if A[i] < p, continue incrementing, stop when A[i] ≥ p
if A[j] > p, continue decrementing, stop when A[j] ≤ p

Example :-

24 (p, l)	9	29	14	19 (r)	27	as p > r, swap both of them
19	9 (l)	29	14	24 (rp)	27	p > l, no swap, just increment
19	9	29 (l)	14	24 (rp)	27	p < l, swap
19	9	24 (pl)	14	29 (r)	27	p < r, no swap, decrement
19	9	24 (pl)	14 (r)	29	27	p > r swap
19	9	14 (l)	24 (pr)	29	27	algo starts from left
19	9	14	24 (prl)	29	27	termination → subarrays

$\underbrace{19 \quad 9 \quad 14}_{lsa}$ 24 $\underbrace{29 \quad 27}_{rsa}$

↳ pivot in middle

sa : subarray

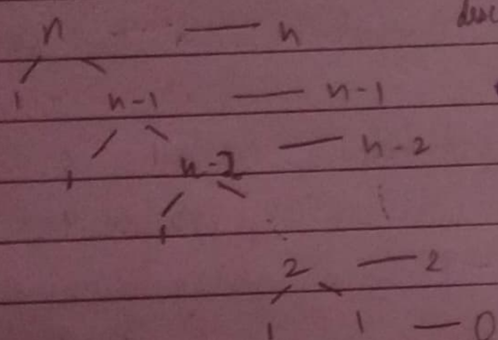
Page No.
Date

lp r lp
19 14 19 29 27 r
9 14 l 19 rp 27 29 pr l
9 14 19 24 27 29 : sorted array

Best case : O(n log n) : Average case
Worst case : $O(n^2)$

$T_{best}(n) = \begin{cases} 2T_{best}(n/2) + n & \text{for } n > 1 \\ 1 & \text{otherwise} \end{cases}$ → termination

when pivot in middle as we divide it into sub problems (n/2). There are 2 of them so its 2 * O(n/2)



desc or asc
worst case analysis [when array is ordered]
used when pivot gives the most unbalanced partitions

$T(n) = T(n-1) + n$ → from 1st step where we scanned full array
$= O(n^2)$
or just calculate from recursion tree

On average, the algo roughly divides array into roughly equal halves at each partition ('balanced'). Same as best case.

improvement
- Randomized quick sort : selects a random element as pivot.
- Insertion sort on subarrays (small), median of 3 as pivot (l m r), 3 way partition

Quicker than Insertion sort, Selection Sort, Merge sort
Not stable. Requires stack to store subarrays. Space efficiency worse than Heapsort.

- Strassen's Matrix Multiplication [7 ×, 18 + or −]

$$\begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} \begin{bmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{bmatrix} = \begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{bmatrix} = \begin{bmatrix} m_1 + m_4 - m_5 + m_7 & m_3 + m_5 \\ m_2 + m_4 & m_1 + m_3 - m_2 + m_6 \end{bmatrix}$$

(a, b, d, c for $a_{00}, a_{01}, a_{11}, a_{10}$; e, f, g, h for $b_{00}, b_{01}, b_{10}, b_{11}$; 1+4−5+7, 3+5, 2+4, 1+3−2+6)

$m_1 = (a_{00} + a_{11})(b_{00} + b_{11})$
$m_2 = (a_{10} + a_{11})(b_{00})$ etc.

Page No.

Date / /

Algorithm Strassen (A, B, n) // input A, B $n \times n$ matrices, output $C = A * B$

if $n = 1$

return $C = A * B$

else

Partition $A = \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix}$ and $B = \begin{bmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{bmatrix}$

where the blocks A_{ij} and B_{ij} are $(n/2)$-by-$(n/2)$

$M_1 \leftarrow$ Strassen $(A_{00} + A_{11}, B_{00} + B_{11}, n/2)$ // $(a+d)(e+h)$

$M_2 \leftarrow$ Strassen $(A_{10} + A_{11}, B_{00}, n/2)$ // $(c+d)(e)$

$M_3 \leftarrow$ Strassen $(A_{00}, B_{01} - B_{11}, n/2)$ // $(a)(f-h)$

$M_4 \leftarrow$ Strassen $(A_{11}, B_{10} - B_{00}, n/2)$ // $d(g-e)$

$M_5 \leftarrow$ Strassen $(A_{00} + A_{01}, B_{11}, n/2)$ // $(a+b)(h)$

$M_6 \leftarrow$ Strassen $(A_{10} - A_{00}, B_{00} + B_{01}, n/2)$ // $(c-a)(e+f)$

$M_7 \leftarrow$ Strassen $(A_{01} - A_{11}, B_{10} + B_{11}, n/2)$ // $(b-d)(g+h)$

$C_{00} \leftarrow M_1 + M_4 - M_5 + M_7$

$C_{01} \leftarrow M_3 + M_5$

$C_{10} \leftarrow M_2 + M_4$

$C_{11} \leftarrow M_1 + M_3 - M_2 + M_6$

return $C = \begin{bmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{bmatrix}$

Recurrance for multiplications :- $T(n) = \begin{cases} 7T(n/2) & \text{for } n > 1 \\ T(1) = ? \end{cases}$

For $n = 2^m$ $\quad T(n) = 7T(n/2) = 7^2 T(n/2^2)$

$T(n) = 7^m T(n/2^m) = 7^m = 7^{\log n}$

$n^{\log 7} = n^{2.807}$

Recurrance for add^n / sub^s

$T(n) = \begin{cases} 7T(n/2) + 18(n/2)^2 & \text{for } n > 1 \\ T(1) = 0 & n = 1 \end{cases}$ use master theorem