sorting techniques

Critera for Analysis

1. Number of comparisons
2. Number of swaps
3. Adaptive
4. stable
5. Extra memory.

- Number of comparisons decided the time complexity of an sorting algorithom.
- How many swap in sorting Analysis.
- if the elements are sorted then (Adaptive) that one.
- some sorting Algo need to perform to sort extra space requaired.

- stable

if same order should be same

C E
6 6

sort

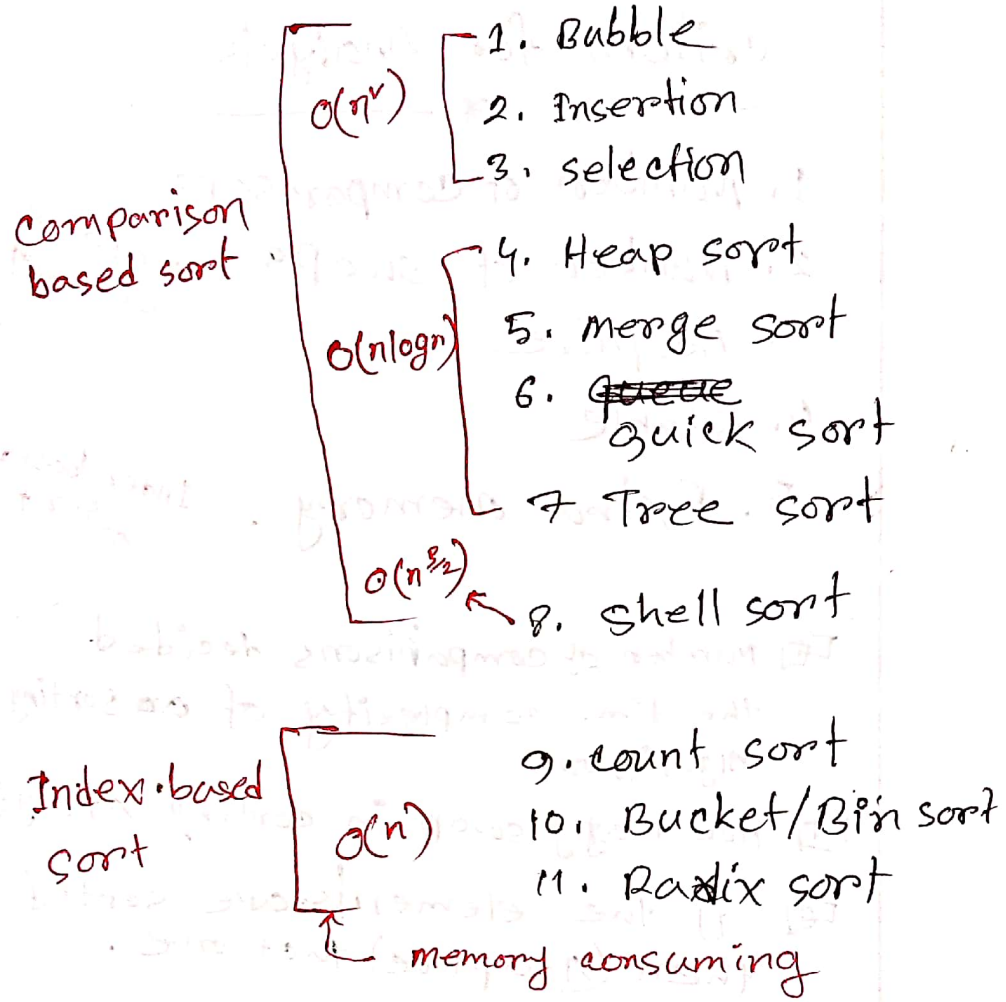
Name →	A	B	C	D	E	F	G	← students name
Marks →	5	8	6	4	6	7	10	

Name ⇒	D	A	C	E	F	B	G
Mark ⇒	4	5	6	6	7	8	10

Various sort tech

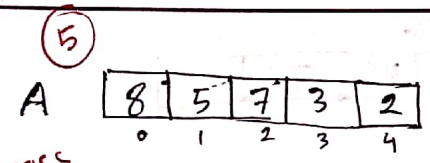
Comparison based sort:
1. Bubble
2. Insertion
3. Selection
4. Heap Sort
5. merge sort
6. quick sort
7. Tree sort
8. shell sort

Index based sort:
9. Count sort
10. Bucket/Bin so
11. Radix sort

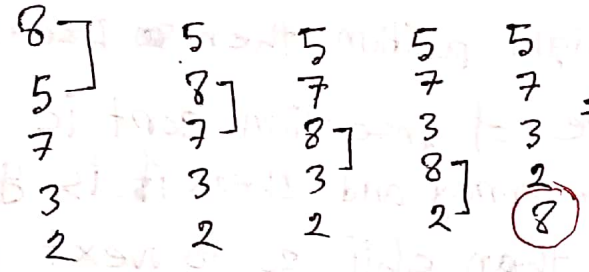


Lecture 2

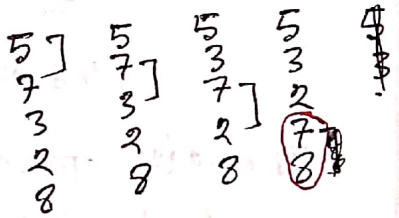
Bubble sort



1st pass



2nd pass



Compare all possible pair of element

In First pass one (8) is sorted

4 · comp

4 - swap

3 - comp

3 - swap

$n = 5$

No of pass : 4 $\Rightarrow (n-1)$

No of comp : $1+2+3+4 \Rightarrow 1+2+3+4 \cdots (n-1) = \frac{n(n-1)}{2} = O(n^2)$

max No of swap : $1+2+3+4 \Rightarrow 1+2+3+4 \cdots (n-1) = \frac{n(n-1)}{2} = O(n^2)$

1 pass $\Rightarrow$ 1 getter element

2 n $\Rightarrow$ 2 n n

3 n $\Rightarrow$ 3 n n

k n $\Rightarrow$ k n n

Using this Algo we get selected Naimber of sorted element

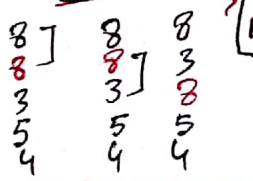
```
void BubbleSort(int A[], int n)
{
    int flag;
    for(i=0; i<n-1; i++)
    {
        flag=0;
        for(j=0; j<n-1-i; j++)
        {
            if(A[j] > A[j+1])
            {
                flag=1;
                swap(A[j], A[j+1]);
            }
        }
        if(flag==0)
            break;
    }
}
```

Min complity $\Rightarrow O(n)$

max complity $\Rightarrow O(n^2)$

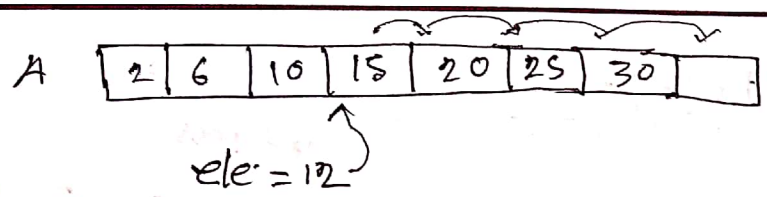
check it Adaptiv or Not

stable



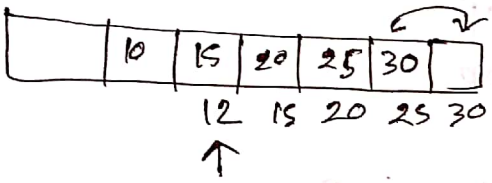
it is stable

Insertion sort



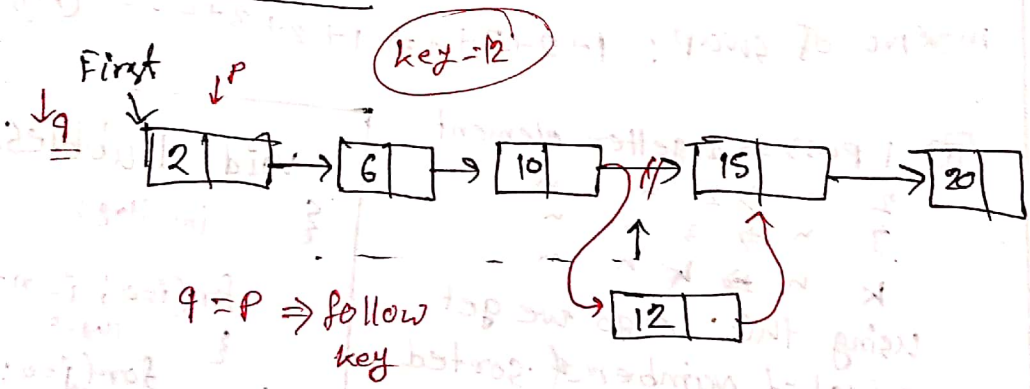
Find the Right position then Insert:

exeict procdure of Insertion sort is
Last of the array and check if is getter than key then shif 30 to next positon again and again



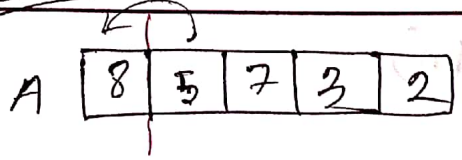
O(n) ⇒ max
O(1) ⇒ min

Insertion in Linked list



Insertion sort

Lectures

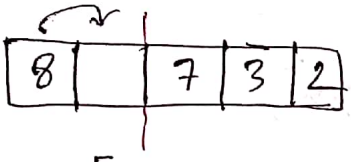


insertion sort is more usefull in link list

1. we assuming that first element (8) is sorted

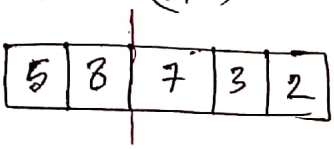
1. Insert 5

1st pass

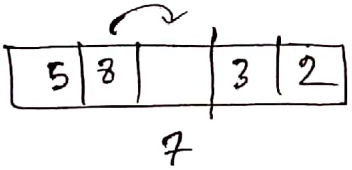


1. com
1. swap

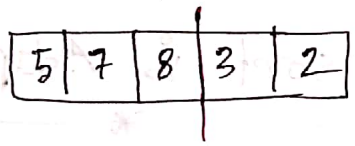
check (5, 8)



2. pass



2 comp
2. swap



$n = 5$

No of pass ⇒ 4

$n \Rightarrow (n-1)$ pass

No of comp = $1+2+3+4 \cdots (n-1)$

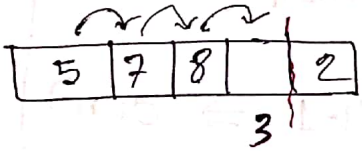
$= \frac{n(n-1)}{2} = O(n^2)$

$O(n^2)$

No of swap = $\frac{n(n-1)}{2}$

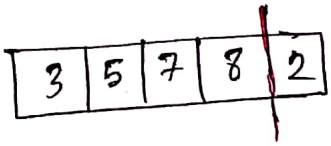
$= O(n^2)$

3. pass

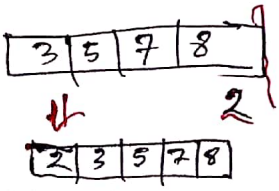


compair with 3

3 compair
3 swap



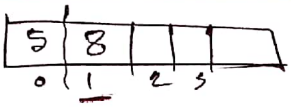
3 pass



4 com
4 swap

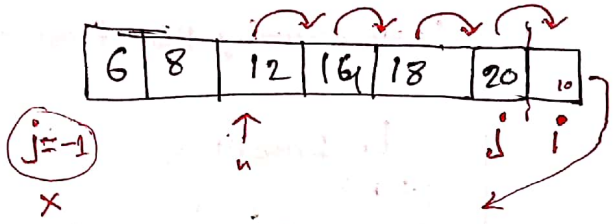
lecture6

Insertion code



insertion sort start from 1

```
void insertionSort(int A[], int n)
{
    for(i=1; i<n; i++)
    {
        j= i-1;
        x= A[i];

        while(j>-1 && A[j]>x)
        {
            A[j+1] = A[j];
            j--;
        }

        A[j+1] = x;
    }
}
```



x = 10

x = 2
x corner case

Lecture 7 insertion sort is Adaptive or Not

A | 2 | 5 | 8 | 10 | 12 | Adaptive

No of comp → n-1 ⇒ O(n)

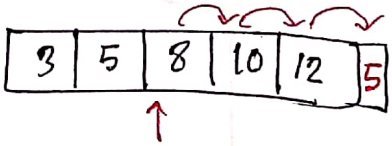
No of swap → 0 ⇒ O(1)

Insertion sort

min → O(n)

max → O(n²)

Stable



after black Red (5) is inserted

Lecture 9

Comparing Bubble and Insertion

	Bubble	Insertionsort	
min comp	$O(n)$	$O(n)$	Assinging
max comp	$O(n^2)$	$O(n^2)$	deciending
min swap	$O(1)$	$O(1)$	Ascending
max swap	$O(n^2)$	$O(n^2)$	Descending
Adaptive	✓	✓	
stable	✓	✓	
Link list	NO	yes	
k pass	yes	NO	

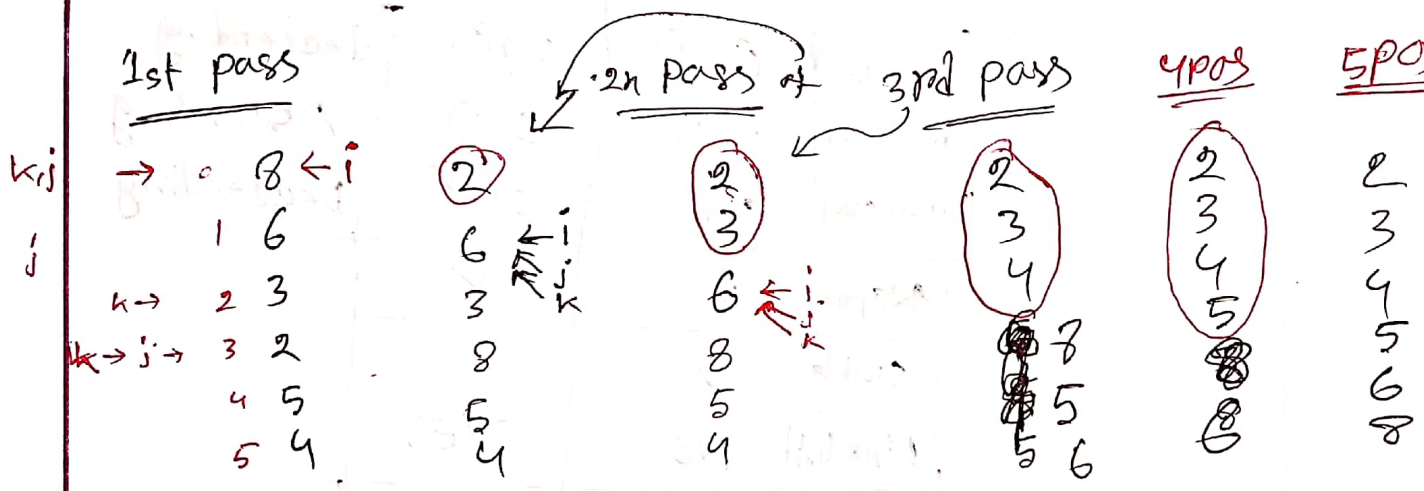
getter

Selection sort

Lecture 10

A

8	6	3	2	5	4
0	1	2	3	4	5



① step: select a position to find out element of this position (i=0)

② step: two pointer (k, j) ⇒ k is point small element and j move again and again ⇒ k follow j
if j is smaller than k ⇒ move k to j (k → j)

③ step: inter change (j to k)

No of comp = $1+2+3+4+\cdots+(n-1) = \frac{n(n-1)}{2}$

$= O(n^2)$

No of swap = $O(n)$

1 pass → 1 swap

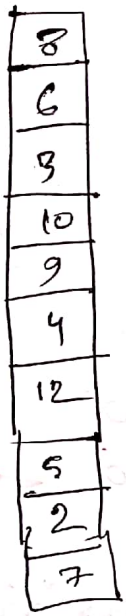
min number of swap

1 pass 1 small element

2 pass 2 " "

k pass k " "

Lecture 11




```
void selectionSort(int A[], int n)
{
    for(i=0; i<n-1; i++) //pass
    {
        for(j=k=i; j<n; j++)
        {
            if(A[j] < A[k])
            {
                k=j;
            }
        }
        swap(A[i], A[k]);
    }
}
```

Lecture 12

Adaptive

0 2
1 4
2 8
3 10
4 12
5 16

✗

we can not Find out any method to tell this Array is sorted or not

stable

0 8
1 3
2 5
3 8
4 4
5 2
6 7

✗

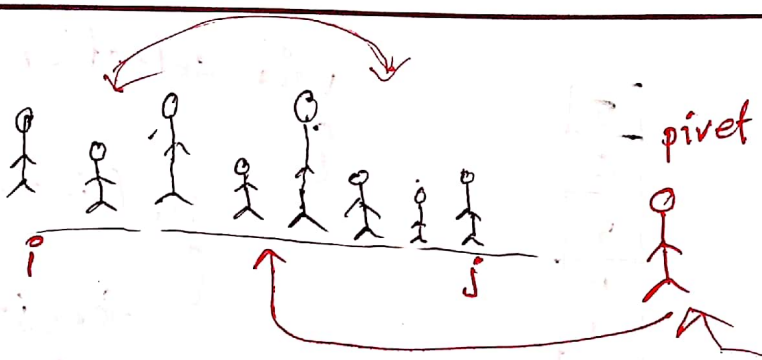
after change so it is not stable

Quick sort

Lecture 14

quick ① teacher ② student

Find the sorted position



→ 10 30 20 70 . 40 . 90 80

→ 80 70 40 30 20 10 90

→ (40 30 20) 50 (90 70 80) ← This the team

You want to find your won sorted possition.

Lecture 15 ∴ procidured

i j

50 70 60 90 40 80 10 20 30

① step: selected the pivet 50

② step: All the small element of 50 is left side

small 50 large

③ step: check j is small element the interchang with i element then 50

i check elemel is getter then 50

1st pass ⇒ (50) 30 60 40 80 10 20 70 ∞
i → i j ← j

2nd pass (50) 30 20, 40 80 10 60 70
i → i → i j ← j

3rd pass (50) 30 20 40 10 80 60 70
j i

stop (i) is getter and equal
if (j) is getter continue
if (j) is smaller stop
interchang pivot and j th element

(10 30 20 40) 50 (80 60 70)
j
↑
split here

Left side and Right side
⇒ we have to sort again

Lecture 16 Quick sort work

(20) 10 30 ∞
i → i → i j
j ← j ←

(10) 20 (30)

quick sort work one two element
* one element Not work

(10) 20 ∞ (two element)
i → i j
j ← j ←

interchang j with pivot same

Lecture 16 Quick sort code

```
int partition(int A[], int l, int h)
{ int pivot = A[l]
  int i=l, j=h;

  do{
     do{i++;} while(A[i] ≤ pivot)
     do{j--;} while(A[j] > pivot)
     if(i<j)
       swap(A[i], A[j]);
  } while(i<j)
     swap(A[l], A[j]);
  Return j;
}
```

* For Assending and desinding time
  Worst case $O(n^2)$

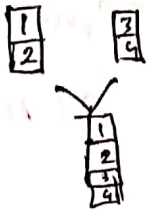
Divide two part

* time complexity $O(n \log n)$
  Average
  Best →

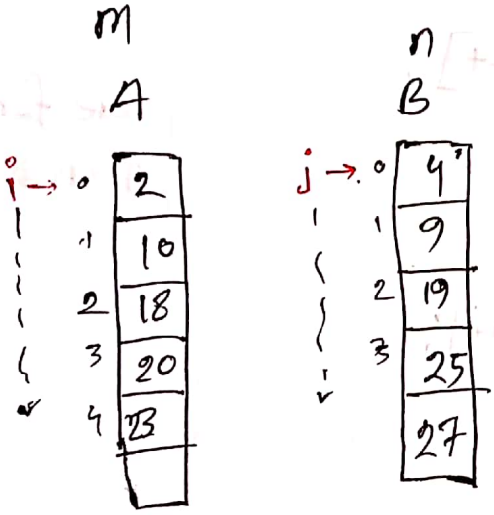
merging

Lecture 19

merging is a process where two sorted list in a single list

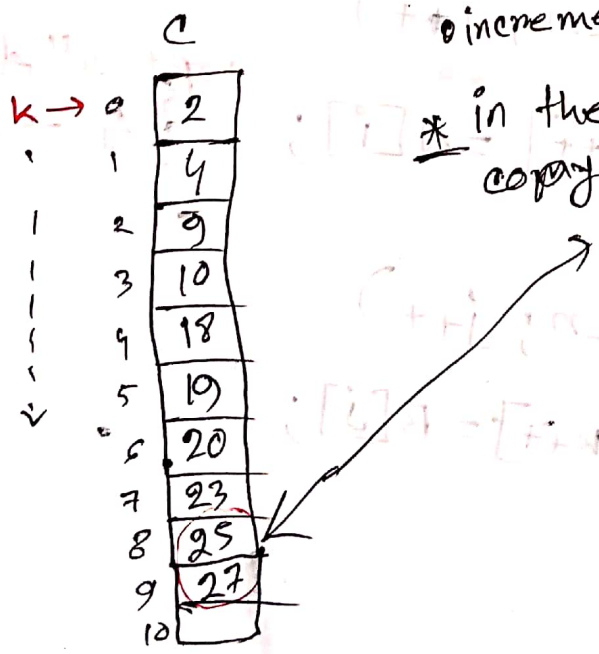


1. merging 2 list
2. merging 2 List in single Array
3. merging multiple list.



who's elemel is smaller copy in C
increment (i or j) and k also

* in the last A or B Not empty copy all the element to C -



.Little code — Array in global (kept)

```
void merge(int A[], int B[], int m, int n)
{
    int i,j,k;
    i=j=k=0;

  while(i<m && j<n)
  {  if (A[i] < B[j])
       {C[k++] = A[i++];
       }
     else
       {
         C[k++] = B[j++];
       }
  }
  for( ; i<m; i++)
    { C[k++] =A[i];
    }
  for( ; i<n; j++)
    { C[k++] = B[j];
    }
}
```

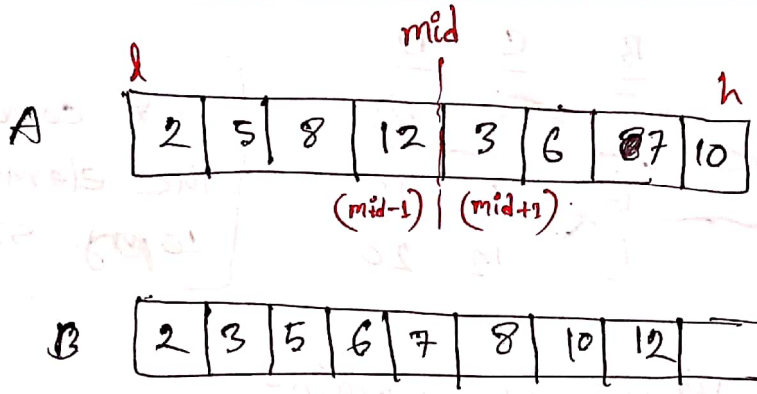
Time complexity

O(m+n)

code for merge A & B ⇒ C

copy all elemel from A → is not empty

concept 1 single Aarry marge How?



key

This Array divided in middle then behave like two array

pass parameter modification 1 Aarray pass l and h and mid

we use same code to single Array marge

2 Array code

```
void singleArray marge (int A[], int l, int mid, int h)
{   int i, k, j;
    i = l; j = mid+1;
    k = l; // where we don't know to marge
    int B[h+1]
while (i <= mid && j <= h)
  {
      if(A[i] < A[j])
      {
          B[k++] = A[i++];
      }
      else
      {   B[k++] = A[j++];
      }
  }
for( ; i<= mid; i++)
  { B[k++] = A[i];
for( ; j <= h; j++)
  { B[k++] = A[j];
}
```

```
for(i=l; i<=h; i+
{
    A[i] = B[i]
}
```

marge multiply Array

A	B	C	D
2	3	5	8
5	6	9	16
15	18	12	20

First comapair all the element and copy small one

4 way marging

m-way maging

1 methord

E
2
3
5
5
6
8
9
12
1

2nd method

A	B	C	D
2	3	5	8
5	6	9	16
15	18	12	10

This two

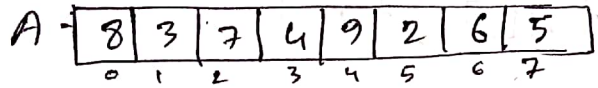
AB ⇒ (AB)C ⇒ (ABC)D

gef

2 way marging

2 way merge sort

Lecture 20 — Iterative merge sort

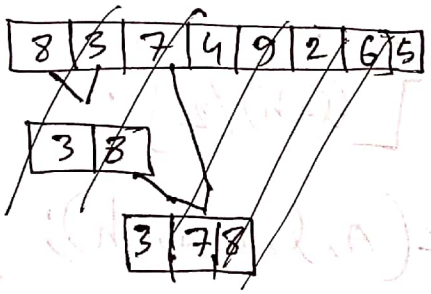


① Statement : If have an Array containing 8 element in Array we have to sort them.

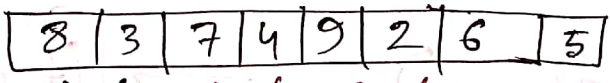
② Apply merge sort ⇒ change the statement ⇒ There is an Array → containing 8 Lists of element

8 | 3 | 7 — each another list

if 1 list ⇒ it ~~merge~~ is sort it self



2 way merge Apply



1st pass: 3 8 | 4 7 | 2 9 | 5 6 — n — 4 list — 2 element

2nd pass: 3 4 7 8 | 2 5 6 9 — n — 2 list

3rd pass: 2 3 4 5 6 7 8 9 — n — Sorted

element pass

$O(n \log n)$

$(\log n)$

$(\log_2 8) = 3$

code for merge sort

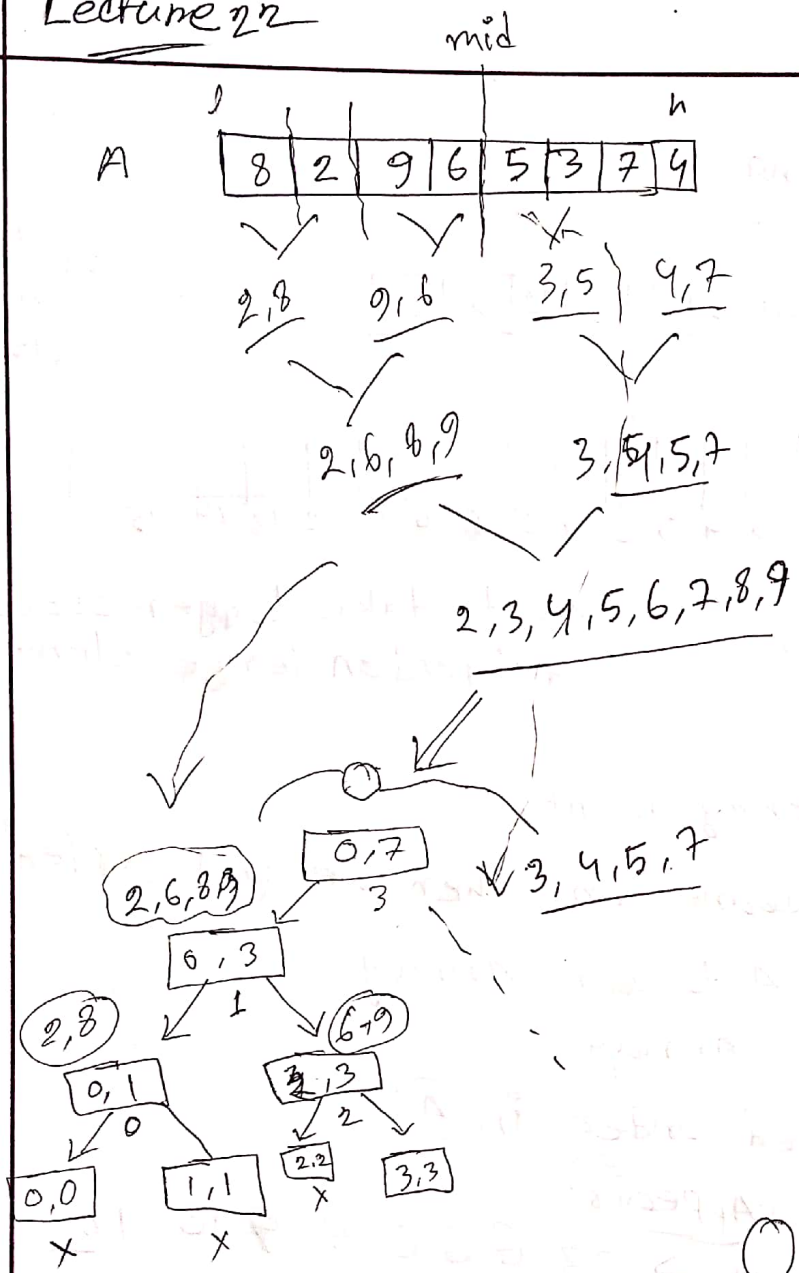
8

```
void ImergeSort(int A[], int n)
{
  int p, i, l, mid, h;

  for(p=2; p<=n; p=p*2)
  {
    for(i=0; i+p-1<n; i=i+p)
    {
      l=i;
      h=i+p-1;
      mid = ⌊(l+h)/2⌋;

      merge(A, l, mid, h);
    }
  }
  if(p/2<n)
  { merge(A, 0, p/2, n-1)
  }
}
```

→ if odd number elem ent

Recursive merge sort

Lecture 22



mid → left and Right

if 1 element in the list then it is sorted

```
void MergeSort(int A[], int l, int h)
{
    if (l < h)
    {
        mid = (l+h)/2;
        MergeSort(A, l, mid);
        MergeSort(A, mid+1, h);
        Merge(A, l, mid, h);
    }
}
```

$O(n \log n)$

Merging is done post order

$A \rightarrow n$

in merge auxiliary $B \rightarrow n$

stack $\rightarrow \log n$

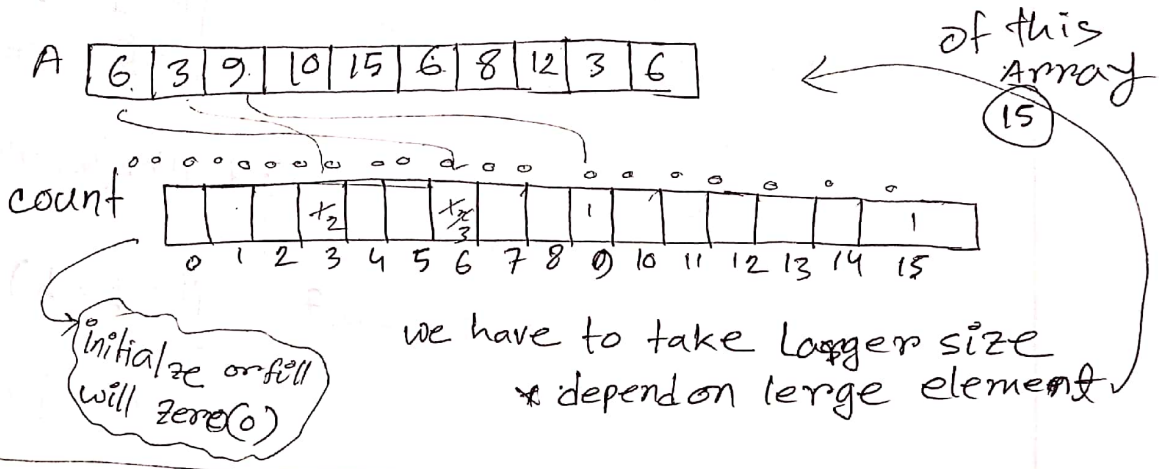
$2n + \log n$

Space $\Rightarrow (n + \log n)$

Lecture 24

Count sort

* Fast sort
* space consuming



Like frequency count

we treverse count Array then we get sorted Element

1. After Assing A to count Array
2. scan count Array
3. back sorted order in A
4. frecquency Appear

A ⇒ 3 3 6 6 6 8, 9 10 12 15

$(n+n) \Rightarrow O(n)$

Space $O(n)$

```
void countsort(int A[], int n)
{   int max
    max = finMax(A, n)
    int *c;
    c = new int [max+1];
    for(i=0; i<max+1; i++)
      {
        c[i]=0;
      }                                  } n

  for(i=0; i<n; i++)
    {  c[A[i]]++; // indexing            } n
    }
    (i=0;) j=0;
  while(i<max+1)
  {                                      } n
    if(c[i]>0)
    {  A[j++]=i;
       c[i]--;
    }
     else
        {  i++;                          O(n)
        }
  }
}
```

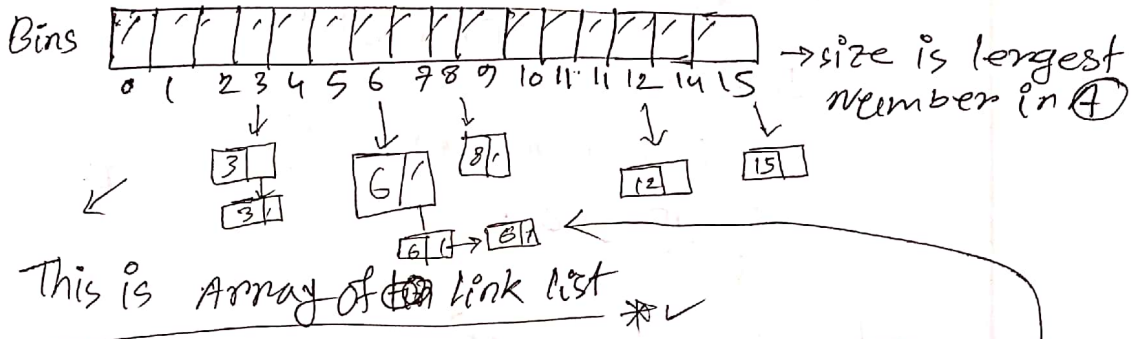
Bucket/Bin sort

similar to count sort

26

A: | 6 | 8 | 3 | 10 | 15 | 6 | 0 | 12 | 6 | 3 |
0 1 2 3 4 5 6 7 8 9

initial null pointer



1. Insert A element is index postion in Bins.
2. scean Bins then insert again → A and Notice that repetation → link list

O(n) time

space O(n)

Bucket sort / Bin

```
void BinSort(int A[], int n)
{
  int max, i, j;
  Node **Bins;
  max = findMax(A, n);
  Bins = new Node[max+1]

  for(i=0; i<max+1; i++)
    { Bin[i] = Null;
    }
  for(i=0; i<n; i++)
    { Insert(Bins[A[i]], A[i])
    }
  i=0; j=0;
  while(i<max+1)
  { while(Bins[i] != Null)
    { A[j++] = Delete(Bin[i])
    }
    i++
  }
}
```

O(n) + total (n)

Radix Sort

Lecture 27

similer to Bin/Bucket

348 → gives sort list element

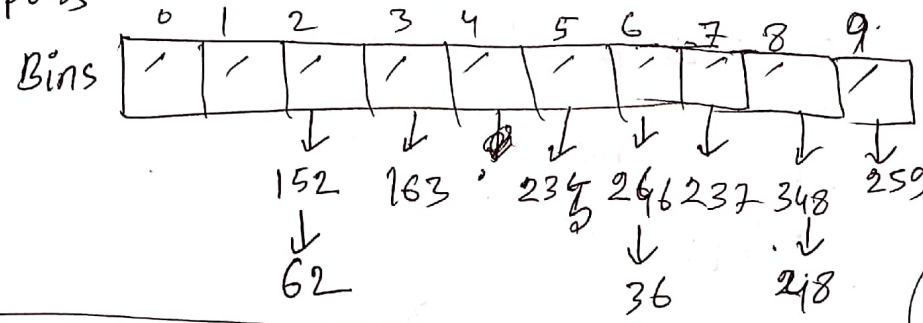
A

237	246	254	348	152	163	235	48	36	62

1st pass : 152, 62, 163, 235, 246, 36, 237, 348, 48, 259

2nd pass : 235, 36, 237, 246, 348 - - - - -

3rd pass - - - - - - - - - -



Deimal number (10) Array Size

Follow FIFO Fast in Fast out

The main concept is that large number have how many digit that number of pass 4 have to perform

① just check last digit of num
237 ⇒ 7
A[i]%10 = 7

348
- 1st pass ⇒ A[i]%10
- 2nd pass ⇒ (A[i]/10)%10
- 3rd pass ⇒ (A[i]/100)%10

that number of pass to bins

3 pass in the box

get sort element

Space O(n)

O(dn) ⇒ O(n)