Sub :

Day

Time :

Date : / /

Searching

Linear Search

Al- 4: Linear (DATA, N, ITEM)

1. Set K = 1 and LOC = 0 → location

2. Repeat steps 3 and 4 while LOC = 0 and K ≤ N.

3. If ITEM = DATA [K], LOC = K

4. Set K = K+1
[End of Step 2 loop]

5. [Successful ?]
If (LOC = 0) then
write : Item is not found.

Else, LOC is the location

Sub :

Day

Time : Date : / /

Efficient-

Alg-5 : Linear (Data, N, Item, Loc)

1. [Insert Item] Set Data[N+1] = Item

2. Set Loc = 1

Repeat while Data[Loc] = Item

Set Loc = Loc+1;

[End of loop]

4. [Successful ?] If Loc = N+1

Set Loc = 0;

complexity $f(n) = \lfloor \log_2 n \rfloor + 1$

Sub :

Day

Time : Date : / /

Binary search (sorted Array)

Alg- 6 : Binary (Data, LB, UB, Item, Loc)

1. Set Beg = LB. End = UB.
   Mid = (int)((Beg + End) / 2)

2. Repeat steps 3 and 4 while
   Beg ≤ End. and Data[Mid] ≠ item

3. If item < Data[Mid] ;
   Set End = Mid - 1
   else, Beg = Mid + 1

4. Set Mid = (Beg + End) / 2

5. If Data[Mid] = item, then
   Set Loc = Mid.
   else Loc = Null.

6. Exit.

Sub :

Day

Time : Date : / /

Steps of Binary search:

Data = 11, 22, 30, 33, 40, 44, 55, 60, 66, 77, 80, 88, 99.

Steps:

1. (11), 22, 50, 33, 40, 44, [55], 60, 66, 77, 80, 88, (99)

2. (11) 22, [30], 33, 40, (44) 55, 60, 66, 77, 80, 88, 99

3. 11, 22, 33, (33), [40] (44), 55, ...

total comparison $\frac{n(n-1)}{2}$

Day

Sub :

Time : Date : / /

Sorting Techniques

Bubble Sort

Algorithm – 7

Bubble (Data, N)

1. Repeat step 2 and 3 for K = 1, to N-1
2. Set PTR := 1
3. Repeat while PTR ≤ N-K
   a. If Data [PTR] > DATA [PTR+1]
      interchang Data [PTR] and data [PTR+1]
   [End of if structure]
   b. Set PTR = PTR + 1
   [End of inner loop]
   [End of outer loop]
4. Exit.

Time complexity

Avg and Worst = n^2

Best = n (already sorted)

worst case = reverse sorted

Sub :

Day

Time : Date : / /

Steps and Passes of Bubble sort

Pass 1 given array 32, 51, 27, 85, 66, 23, 13, 57

(a) compare A1, A2, 32 < 51, no change [N.C]

(b) compare A2, A3, Interchange [I.C.]
32, (27), (51), 85, 66, 23, 13, 57

(c) compare A3, A4, NC

(d) " A4, A5, 85 > 66, IC
32, 27, 51, (66), 85, 23, 13, 57

(e) " A5, A6 85 > 23, IC
32, 27, 51, 66, (23), (85), 13, 57

(f) " A6, A7 85 > 13, IC.
32, 27, 51, 66, 23, (13), (85), 57

(g) " A7, A8 85 > 57, IC
32, 27, 51, 66, 23, 13, (57), (85).

Sub :

Day

Time : Date : / /

Pass 2:

(57) (33) 51, 66, 23, 13, 57, 85

(57), 33, 51, (23), (66), 13, 57, 85

57, 33, 51, 23, (13), (66), 57, 85

57, 33, 51, 23, 57, (66), 85

Pass 3: 57, 33, (23), (51), 13, 57, 66, 85

57, 33, (13), (57), 57, 66, 85

Pass 4: 57, (23), (33), 13, 57, 57, 66, 85

57, 23, (13), (33), 57, 57, 66, 85

Pass 5: (23) (57), 13, 33, 51, 57, 66, 85

23, (13), (57), 33, 51, 57, 66, 85

Pass 6. (13), (23), 57, 33, 51, 57, 66, 85.

Sub :

Day
Time : Date : / /

Selection Sort

get the min, put it on 1st

Alg-8

Min (A, K, N, Loc)

1. Set min = A[K]. Loc = K
2. Repeat J = K+1, K+2, ... N.
   if min > A[J]. Set min = A[J]
   Loc = J.
3. Return Loc.

Selection (A, N)

1. Repeat 2 and 3 for K = 1, ... N-1
2. call Min (A, K, N, Loc)
3. swap A[K] and A[Loc]
   Set Temp = A[K], A[K] = A[Loc]
   and A[Loc] = Temp
4. Exit.

Time complexity

Best, avg, worst = n^2

→ 77, 33, 44, 11, 88, 22, 66, 55

→ 33, 77, 44, 11, ---

Sub: 33, 44, 77, 11, 88, 22 ---

11, 33, 44, 77, 88 — —

Day

Time: Date: / /

Insertion sort

Let him decid

Algorithm ⇒

Insertion (A, N)

1. Set A[0] = −∞
2. Repeat steps 3 to 5 for k = 2, 3, ... N
3. Set temp = A[k]. and PTR = k − 1
4. Repeat while temp < P[PTR]
   - (a) Set A[PTR+1] = P[PTR]
   - (b) Set PTR = PTR − 1

   End of Loop
5. Set A[PTR+1] = Item

   End of Loop.
6. Return.

77, 33, 44, 11, 88, 22, 66, 55

11, 33, 44, 77, 88, 22, 66, 55

Sub :

Time : Date :

Quick Sort → Give a Number its own position

A - 10 Quick (A, N, BEG, END, LOC)

1. Set Left = Beg, Right = End, Loc = Beg
2. [Scan from right to left]
   - (a) Repeat while A[Loc] ≤ A[Right] and Loc ≠ Right
     Right = Right − 1.

   [End of loop]

   - (b) If Loc = Right, then Return.
   - (c) If A[Loc] > A[Right], then
     - (i) Interchange.
     - (ii) Set Loc = Right
     - (iii) [End of If]
     - (iv) Go to step 3.

---

complexity

worst case = sorted or reverse sorted $O(n^2)$

Best and avg = divide into two array equally, $O(n \log n)$

Sub :

Day

Time : Date : / /

3. [Scan from left to right]

(a) Repeat while A[left] ≤ A[loc] and left ≠ loc

left = left + 1

[End of Loop]

(b) If loc = left, then, Return

(c) If A[left] > A[loc], then

(i) [Interchange A[left] and A[loc]

A[left] = Temp

(ii) Set loc = left

(iii) Go to step 2.

Quick sort (A, N)

1. TOP = NULL.
2. Push boundary values of A into stacks when A has 2 or more elements.

If N > 1, then TOP = TOP + 1, lower[1] = 1

upper[1] = N.

Sub :

Day

Time : Date : / /

3. Repeat steps 4 to 7 while TOP ≠ NULL.

4. [Pop sublist from stacks]
Set Beg = Lower[TOP], End = Upper[TOP]
TOP = TOP - 1

5. call Quick(A, N, Beg, End, Loc)

6. [Push a left sublist onto stacks when it has 2 or more elements]
If Beg < Loc - 1, then.
TOP = TOP + 1, Lower[TOP] = Beg
Upper[TOP] = Loc - 1
[End of If structure]

7. Push Right sublist onto stacks when it has 2 or more elements
If Loc + 1 < End, then
TOP = TOP + 1, Lower[TOP] = Loc + 1
Upper TOP = End
[End of If]
[End of step 3 loop]

8. Exit.

Sub :

Day

Time : Date : / /

showing the steps for getting pivot.

→ taking 1st element As pivot.

→ first check from right to left whether the ~~element~~ pivot is less then these. if pivot is greater, swap.

→ Second check from left to right whether pivot is greater than these. if pivot is less than, swap pivot.

① 44, 33, 11, 55, 77, 90, 40, 60, 99, 22, 88, 66

⑪ 22, 33, 11, 55, 77, 90, 40, 60, 99, 44, 88, 66

⑪ 22 33, 11, 44, 77, 90, 40, 60, 99, 55, 88, 66

⑪ 22, 33, 11, 40, 77, 90, 44, 60, 90, 99, 88, 66

⑪ 22, 23, 11, 40, 44, 90, 77, 60, 99, 55, 88, 66

So, all the elements before 44 is smaller and after is greater. 44 is in its right place.

Sub :

Day
Time : Date : / /

Merging → Split and Merge

Alg-11

Merg (in A, l, mid, h)
{ int i, j, k;
int L[mid-l+1], R[h-mid] → temp array
for (i=0; i<mid-l+1; i++) L[i] = a[l+i] → copy
for (j=0; j<h-mid; j++) R[j] = a[mid+1+j]
while (i<mid-l+1 && j<h-mid).
{ if (L[i] <= R[j]), a[k++] = L[i++]
else a[k++] = R[j++].
}
copy (a, L) while (mid-l+1).
copy (a, R) while (j<h-mid)
}.

complexity

best, avg. and worst = n Log n.

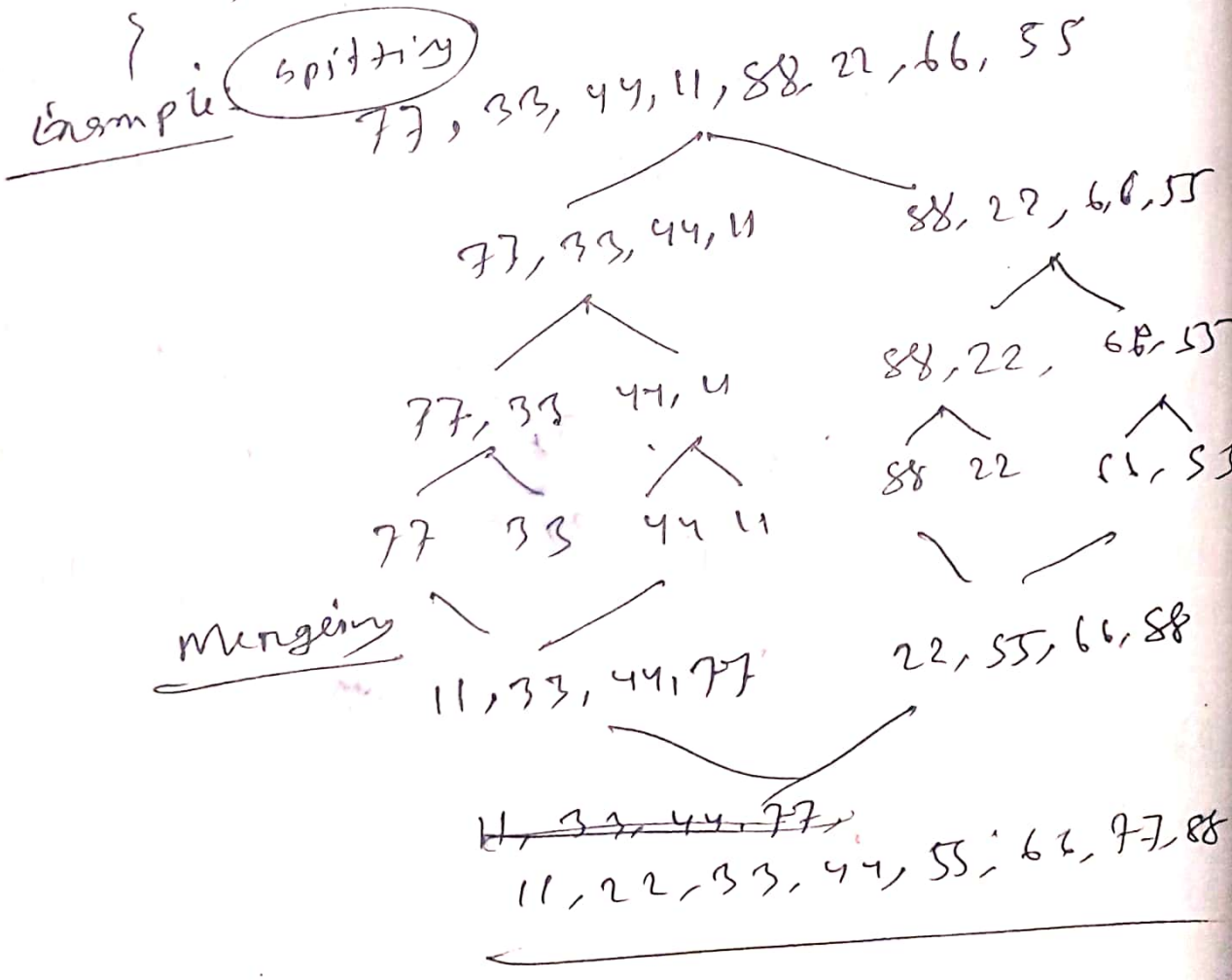
Sub :

Day

Time : Date : / /

```
Merge_sort (int a[], int l, int h)
{
    if (l < h)
    {
        int mid = (l+h)/2
        Merge_sort (a, l, mid)
        Merge sort (a, mid+1, h)
        Merge (a, l, mid, h);
    }
}
```

Example (Splitting) 77, 33, 44, 11, 88, 22, 66, 55



Sorting

Heapsort

Heap sort is a comparison based sorting technique based on Binary Heap data structure. similar to selection-sort.

full → All the height- have maximum no. of nodes

complete - Not maximum

Max heap: Every parent is greater than its child

Min heap: structure:

Array representation:

a node if is index i

left child $= 2*i$

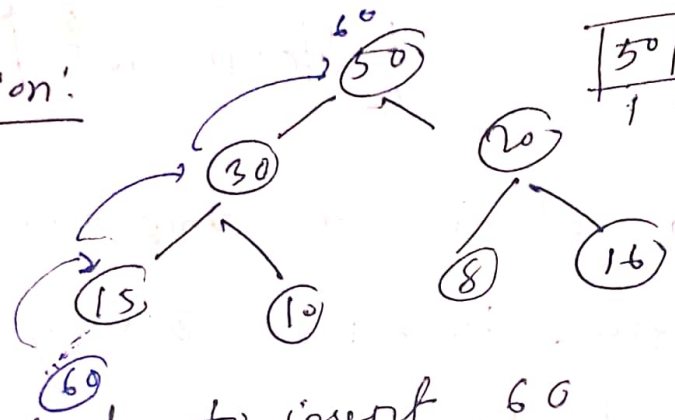
right $= 2*i+1$

parents $= \lfloor \frac{i}{2} \rfloor$

Heap → complete Binary

Max Heap:

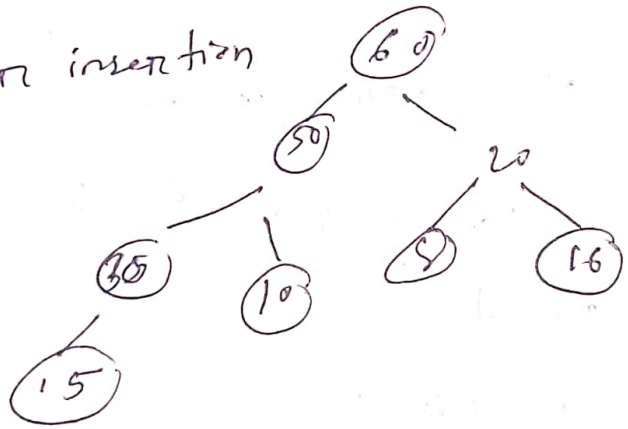
Insertion:



50	30	20	15	10	8	16	
1	2	3	4	5	6	7	8

Time (equal to swaps) (log N) if we want to insert 60 (indx 8)
parent = 4 (15)

After insertion



Delete:

Only root element can be deleted.

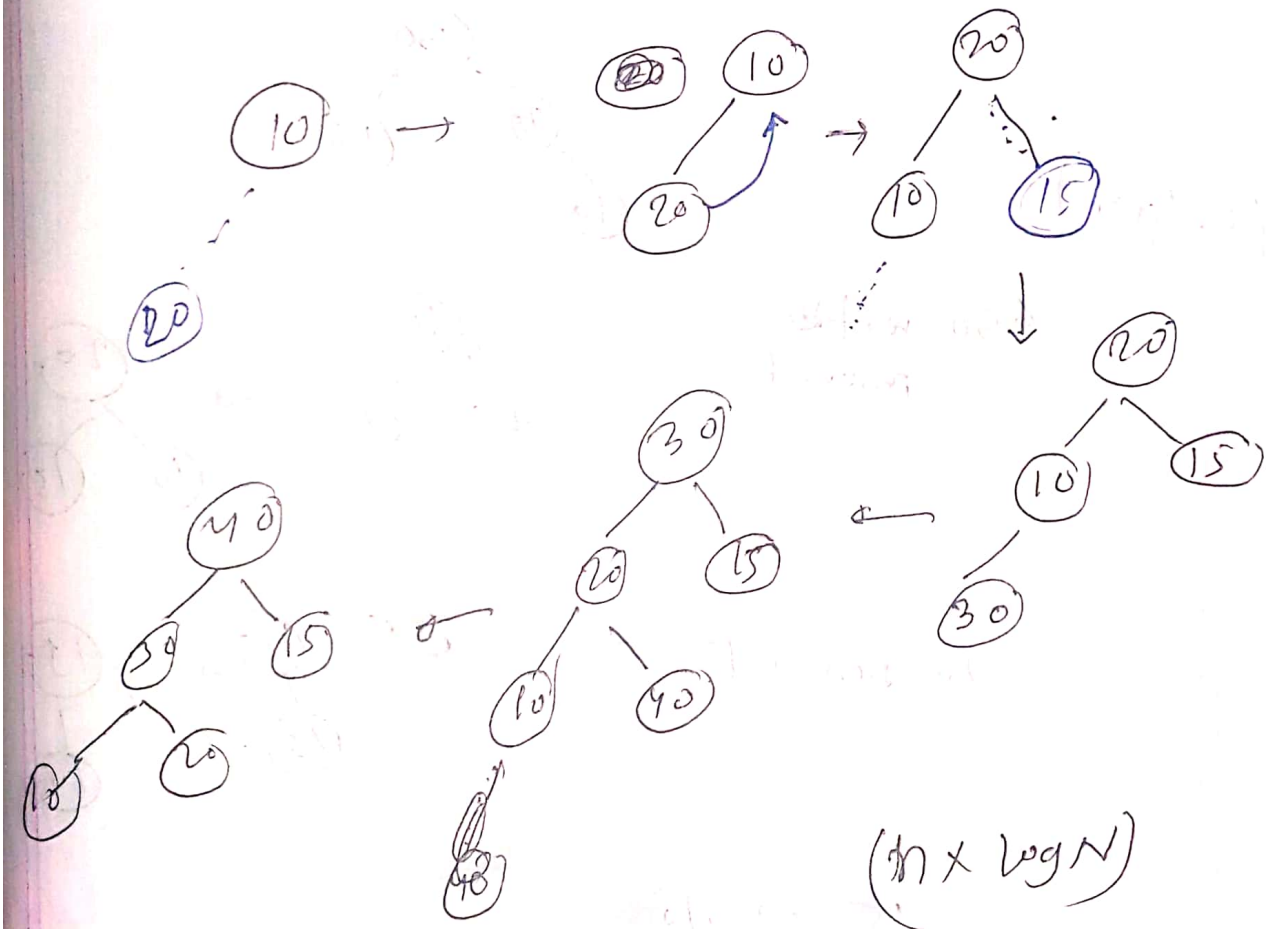
Time = log N → then only the last element will replace that place.

→ Now compare the children

Heap sort

10, 20, 15, 30, 90

step-1 : create Heap (Max/Min)

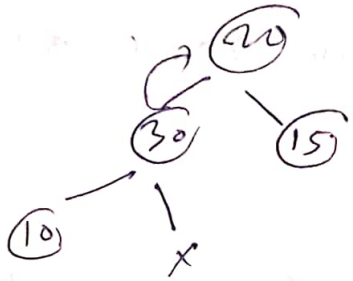


$(n \times \log N)$

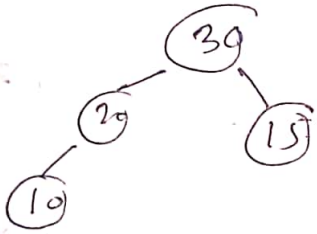
4

Step-2: Delete: (root), last element will be replaced

40 is removed →



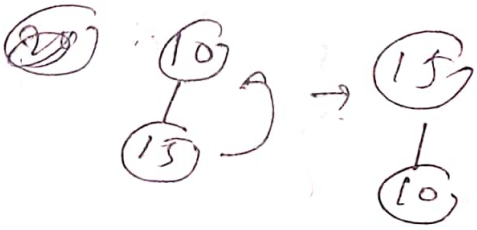
↓ Adjust



$O(N \log N)$

& 30 ~~will be~~ removed

20 removed



15 → 10

code:

```
void Heapify (int n, int i)
{
    int largest = i;
    int left = 2*i+1
    int right = 2*i+2
    if (left < n && arr[left] > arr[largest])
        largest = left.
    if (right < n && arr[right] > arr[largest])
        largest = right;
    if (largest != i)
    { swap (arr[i], arr[largest])
      heapify (n, largest));
    }
}
void heapsort (int n)
{
    for (int i = n/2-1; i >= 0; i--)
    { heapify (n, i);
    }
    for (int i = n-1; i >= 0; i--)
    { swap [a[0], a[i]);
      heapify (i, 0));
    }
```

Complexity:

Time: Best - $O(n \log n)$

Avg: "

Worst: "

Stable SORTING

counting sort

#Input with Range. (n, k) / No negative value.

Let the initial array be:

n = 17
k = 9

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	1	1	0	2	5	4	0	2	8	7	7	9	2	0	1	9

count A (k+1)

0	1	2	3	4	5	6	7	8	9
3	3	4	0	1	1	0	2	1	2

(We need to prefix sum to get the index of new element)

Why?: for example: 1 → 3 times
2 → 3 times

the actual array would be

1 1 1 2 2 2
1 2 3 4 5 6 is 3

here frequency of 1 & also means that index of 2 will start after 3.

so, prefix sum helps us to get what the starting index of a number should be

countA =

3	6	10	10	11	12	12	14	15	17

Now we need an array to store the sorted output, same size with original

output:

0																16

we will traverse original array from n-1; first we got 9. what is the position of 9?

so 1st take help from countA to get the index. Index of countA holds the actual index of an element (explained earlier)

In this case countA[9] = 17,

so index of 9 will be 17 - 1 = 16

arr[count[arr[i]] - 1] = arr[i];

We should decrement the countA[i] value also.

8

code:

```
void countsort (int n)
{   int output [n]
    int countA [RANGE+1]
    memset (&countA, 0, sizeof (countA));

for (int i=0; i<n; i++)
        countA[arr[i]] ++;

for (i=1; i<= RANGE ; i++)
    countA [i] += countA [i-1];

for (int n-1; i>=0; i--)
  { output [countA[arr[i]]-1] = arr[i];
    countA[Arr[i]] --;
  }
cout<<output;
```

complexity:

Avg case : $O(d(n+b))$

Number of passes.
No. of element
Number of possible value that each digit can have

Worst case : $O(n^2)$

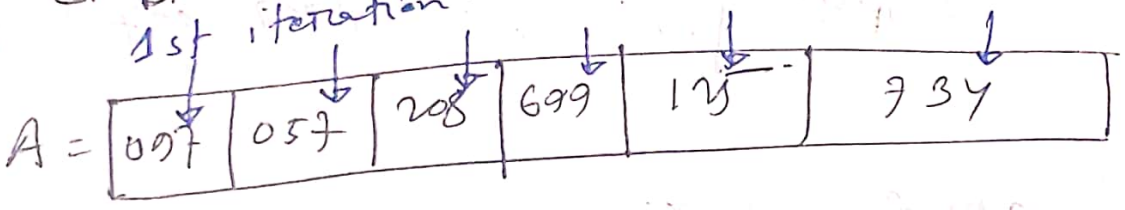
If the largest element equals to n.

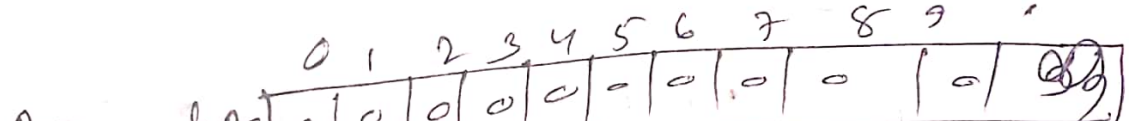
Problem of cont-sort -
if the elements are not uniform
Radix Sort waste memory

$A = \{97, 57, 208, 699, 125, 734\}$

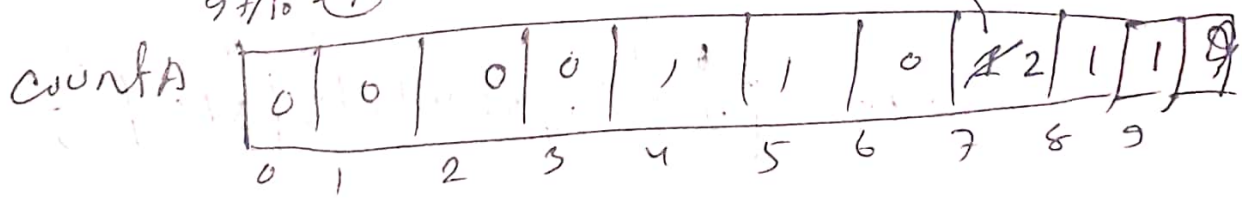
$n = 6, \quad R = 9$

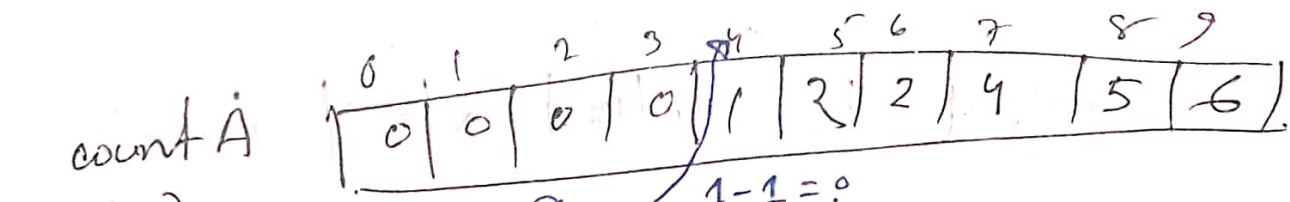
→ Works with single digit -
→ Modified version of counting sort.
→ d Bi: Unit digit of each number.





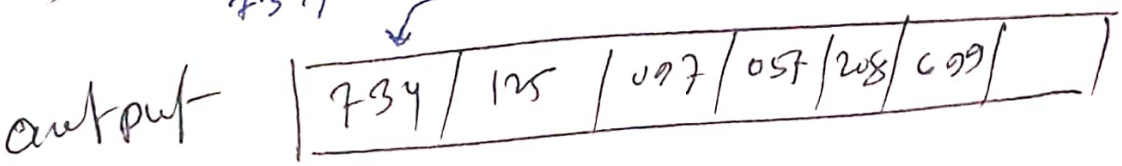
97%10 = 7





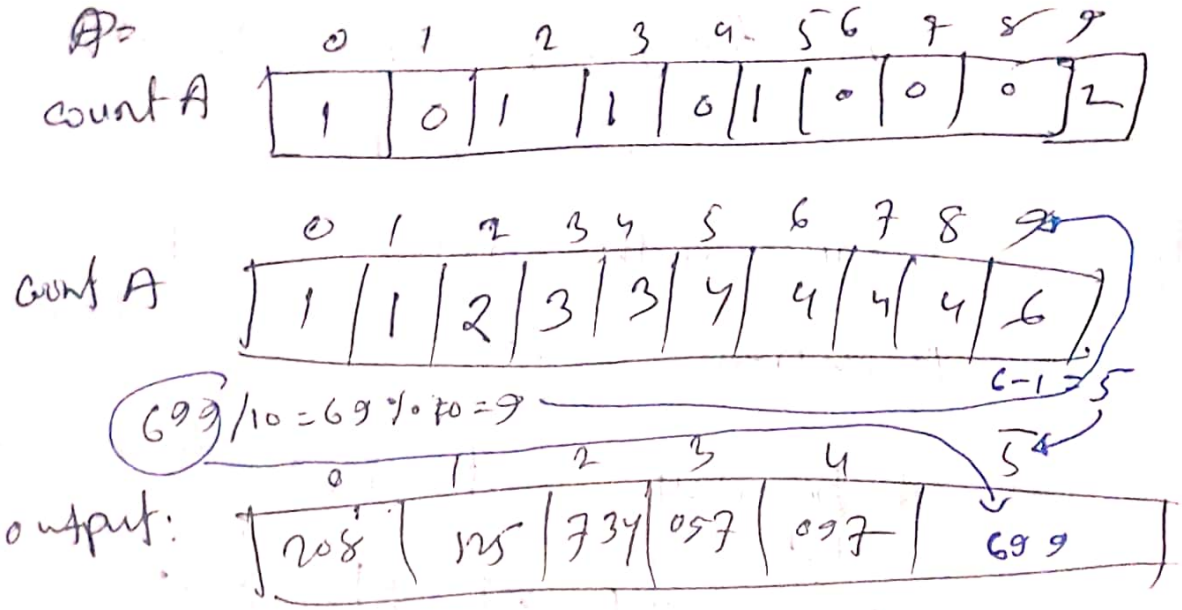
734%10 = 4

1 - 1 = 0

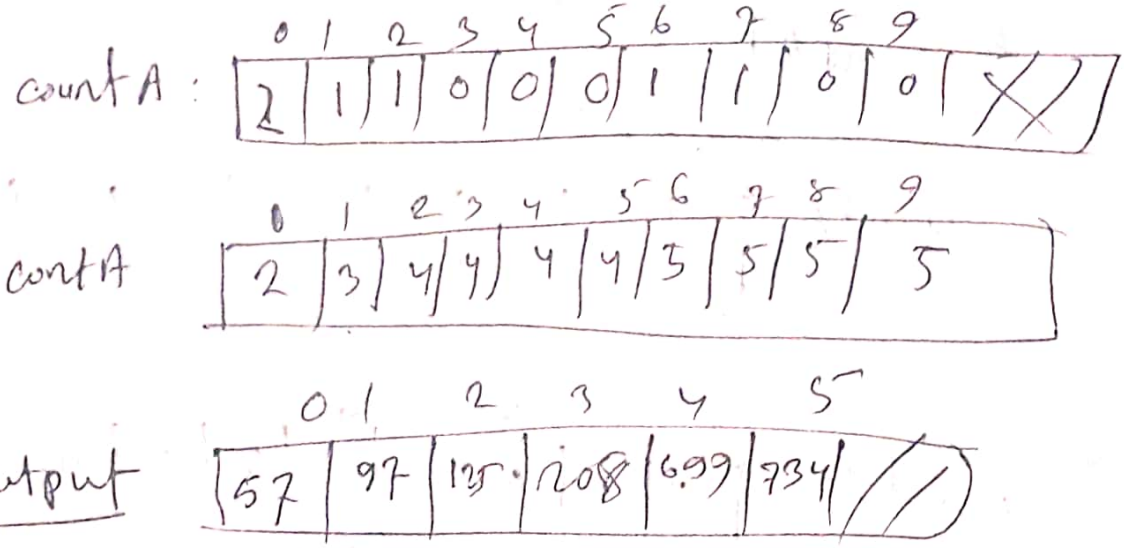


10

ক্রমানুসারে থাকা এলো: 734 125 97 57 208 (699)



b অংশের Same:



```
void radixSort (int n)
{
    int mx = max (arr); // pseudo

    for (int pos = 1; mx/pos > 0; pos *= 10)
        countSort (n, pos);
}

countSort ( int n, int pos)
    int output [n];
    int countA [10];
    memset (countA)
for (int i = 0; i < n; i++)
    countA [(arr[i]/pos) % 10] ++;

for (int i = 1; i <= 10; i++)
    countA[i] += countA[i-1];

for (int i = n-1; i >= 0; i--)
    output [countA[(arr[i]/pos)%10] - 1]
                = arr[i];
    countA [(arr[i]/pos) % 10] --;

}   arr = output);
```

Bucket sort

- Works on floating point numbers between range 0.0 to 1.0.
- inputs should be uniformly and independently distributed to get a running time $O(n)$.

Bucket sort (A)

- let B[0...n-1] be a new array
- n = A.length
- for i = 0 to n-1, B[i] = empty list.
- for i = 1 to n
  - insert A[i] into list B[n A[i]]
- for (i to n-1)
  - sort list B[i] with insertion sort
- concatenate together

17

code:

```
void bucketsort (int n)
{
    vector<float> v[n];
    for (int i = 0; i<n; i++)
    {
        int b = n * arr[i];
        v[b].push_back(arr[i]);
    }
    for (int i = 0; i<n; i++)
        sort(v[i].begin(), v[i].end());
    int indx = 0;
    for (int i = 0; i<n; i++)
    {
        for (int j = 0; j<v[i].size(); j++)
        {
            arr[indx++] = v[i][j];
        }
    }
}
```

Complexity Analysis

Merge Sort

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

$$T(n) = 2T(n/2) + n \quad \text{---} (1)$$

substitute $n/2$ in place in eq (1)

$$T(n/2) = 2T(n/4) + n/2 \quad \text{---} (2)$$

Substitute eq (2) in eq (1)

$$T(n) = 2\{2T(n/4) + n/2\} + n$$

$$T(n) = 2^2 T(n/2^2) + 2n \quad \text{---} (3)$$

substitute $n/4$ in ... (3)

$$T(n/4) = 2T(n/8) + n/4 \quad \text{---} (4)$$

$$T(n) = 2^2\{2T(n/8) + n/4\} + 2n$$

$$= 2^3 T(n/2^3) + 3n \quad \text{---} (5)$$

$$\therefore T(n) = 2^4 T(n/2^4) + 4n \text{----}$$

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

let, $\frac{n}{2^i} = 1$

$\Rightarrow 2^i = n$

$\Rightarrow i = \log_2 n$

$\therefore T(n) = n T(1) + n \log_2 n$

$= n + n \log_2 n$

$= O(n \log_2 n)$

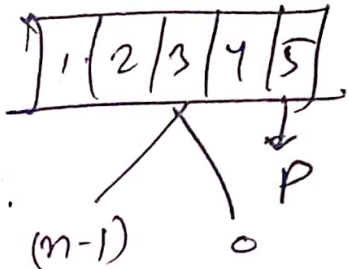
Best, worst, Avg : $O(n \log_2 n)$

Space $= O(n)$

quicksort

partition

Worst case: Already sorted.



$T(n) = T(n-1) + n$ → partition

$= T(n-2) + (n-1) + n.$

$= T(n-3) + (n-2) + (n-1) + n.$

$= T(n-K-1) + (n-K) \dots (n-2) + (n-1) + n.$

n → 1,7

n-1 → 2,7

n-2 → 3,7

n-3 → 4,7

n-k → 5,7

for terminating, $n-K-1 = 1.$

$\Rightarrow K = n-2$

$T(n) = n + (n-1) + (n-2) \dots$

$[n-(n-2)] + T(n-2)$

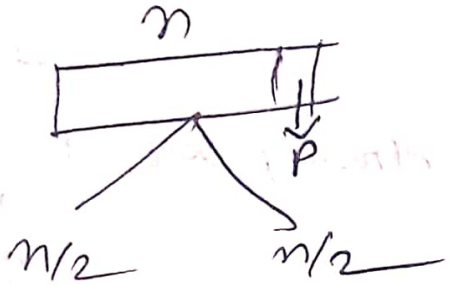
$T(n-(n-2)-1)$

$= n + (n-1) + (n-2) \dots 2+1$

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

$T(n) = O(n^2)$

Best Case: (Array is divided in two equal part)



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

↗ partition

$O(n \log n)$ like as merge sort

Avg case:

1 element pivot 100 elements
100 elements pivot 1 element

$$T(n) = T(i) + T(n-i-1) + c_n$$

$$= c_n + \frac{1}{n}\sum_{i=0}^{n-1} (T(i) + T(n-i-1))$$

$$= c_n + \frac{1}{n}\sum_{i=0}^{n-1} 2\,T(i)$$

$$T(n) = c_n + \frac{2}{n}\left(T(0) + T(1) + \cdots T(n-1)\right)$$

$$n\,T(n) = n^2 c + 2\left(T(0) + T(1) + \cdots T(n-1)\right) \quad \text{---} \ (1)$$

put $n = n-1$

$$(n-1)\,T(n-1) = (n-1)^2 = (n-1)^2 c + 2(T(0) + T(1) + \cdots T(n-2)) \quad \text{---} \ (2)$$

(1) − (2)

$$n\,T(n) - (n-1)\,T(n-1) = cn^2 - c(n-1)^2 + 2T(n-1)$$

$$= cn^2 - c(n^2 - 2n + 1) + 2T(n-1)$$

$$= 2T(n-1) + 2cn - c$$

$$\therefore n\,T(n) = (n-1)c + T(n-1) + 2T(n-1) + 2cn - c$$

$$= T(n-1)\,(n+1) + 2cn - c$$

dividing both side by $n(n+1)$

$$\frac{T(n)}{n+1} = \frac{T(n-1)}{n} + \frac{2c}{n+1}$$ [remove c as it is constant]

$$\Rightarrow \frac{T(n)}{n+1} - \frac{T(n-1)}{n} = \frac{2c}{n+1} \quad \text{---(1)}$$

put $n = n-1$ →

$$\frac{T(n-1)}{n} - \frac{T(n-2)}{n-1} = \frac{2c}{n} \quad \text{---(2)}$$

put $n = n-2$ →

$$\frac{T(n-2)}{n} - \frac{T(n-1)}{n-2} = \frac{2c}{n-1} \quad \text{---(3)}$$

$$\therefore \frac{T(n)}{n+1} - \frac{T(0)}{1} = 2c\left(\frac{1}{n+1} + \frac{1}{n} + \cdots \frac{1}{3} + \frac{1}{2}\right)$$

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

$$T(n) = (n+1)\, 2c \log(n+1)$$

$$\approx n \log(n+1)$$

$$\approx n \log n$$

Merge Sort

left, mid, right

```
void merge (int arr[], int p, int q, int r)
{   int n1 = q-p+1, n2 = r-q-1+1
    int L[n1], M[n2], i, j, k;
    for (i=0; i<n1; i++)  L[i] = arr[p+i];
    for (i=0; i<n2; i++)  M[i] = arr[q+1+j];
    i=0; j=0; k=p
    while (i<n1 && j<n2)
    {   if (L[i] <= M[j])
            { arr[k] = L[i];
              i++;
            }
        else { arr[k] = M[j]
               j++;
             }
        k++;
    }
// Then the sub parts may not be equal, so there may remain some element.
```

```
while (i < n1)
{   arr[k] = L[i];
    i++; k++;
}
while (j < n2)
{   arr[k] = M[j];
    j++;
    k++;
}
}
void mergesort (arr[], l, r)
{ if(l > r) return;
  int mid = (l+r)/2;
  mergesort (arr, l, mid);
  mergeSort (arr, mid+1, r);
  merge (arr, l, mid, r);
}
```

Sub :

Day

Time : Date : / /

Time and Space complexity

Alg	Best case	Avg	Worst	Space (worst)
Bubble	$\Omega(N)$	$\theta(N^2)$	$O(N^2)$	$O(1)$
Selection	$\Omega(N^2)$	$\theta(N^2)$	$O(N^2)$	$O(1)$
Insertion	$\Omega(N)$	$\theta(N^2)$	$O(N^2)$	$O(1)$
Merge	~~$\Omega(N)$~~ $\Omega(N \log N)$	$\theta(N \log N)$	$O(N \log N)$	$O(N)$
Quick	$\Omega(N \log N)$	$\theta(N \log N)$	$O(N^2)$	$O(N \log N)$
Heap	$\Omega(N \log N)$	$\theta(N \log N)$	$O(N \log N)$	$O(1)$

faster