FREEMIND

Date

Page

Lecture-35 (Bubble Sort)

→ What is sorting
Sorting → ordering of elements → ↑ or ↓ order

→ Bubble Sort Algorithm
"Repeatedly swap two adjacent elements if in wrong order"
↳ untill one array is sorted

Jaise Bubbles Niche se water k surface pe aajate hai Vaise hi jo larger/max. elements honge Vo array ki end me aajayenge.

eg:- 20, 50, 40, 10, 30

20, 40, 50, 10, 30
20, 40, 10, 50, 30
I pass 20, 40, 10, 30, 50

20, 10, 40, 30, 50
II pass 20, 10, 30, 40, 50
III pass 20, 10, 30, 40, 50
IV pass 10, 20, 30, 40, 50

n → elements
n-1 passes → n-1 elements at correct position
i.e ⇒ n elements at correct position.

Bubble Sort Code

```
for (int i=0, i<n-1, i++) {
    for (int j=0, j<n-1-i, j++) {
        if (arr[j] > arr[j+1]) {
            swap(arr[j], arr[j+1])
        }
    }
}
```

Code

```
#include <iostream>
#include <vector>
using namespace std;

void bubbleSort(vector<int> &v) {

    int n = v.size();

    for (i=0; i<n-1; i++) { bool flag = false;
        for (j=0; j<n-i-1; j++) {

            if (v[j] > v[j+1]) { flag = true;
                swap(v[j], v[j+1]);
            }
        } if (!flag) break;
    }
    return;
}
int main () {
    int n
    cin>>n;
    Vector<int> v(n)
    for(i=0; i<n; i++) {
        cin>> v[i];
    }
    bubble sort (v)
    for (i=0; i<n, i++) {
        cout << v[i] << " ";
    }
    cout << endl;

}
```

if (flag = false)
    break;

FREEMIND

Date

Page

⇒ Maximum no. of swaps in worst case in Bubble Sort
→ When array is in descending order.

eg:⇒ 50 40 30 20 10 4 swaps n-1

40 30 20 10 50 3 swaps n-2

30 20 10 40 50 2 swaps n-3

20 10 30 40 50 1 swap 1

$n-1 + n-2 + - - - \cdots + 1$

Sum of 1 to n-1

$n(n-1)/2$ Ans

⇒ Bubble Sort Time and Space complexity

Max swaps $= n(n-1)/2$

Time complexity $= O(n^2)$ → Ignoring constants

Space complexity $= O(1)$

→ There is No extra space needed array is altered in its given original space

→ How to optimize the bubble sort in the case of nearly sorted array.

```
i = 0 ⟶ i = n-1
    bool flag = false
    j = 0 ⟶ j = n-i-1
        if (arr.[j] > arr.[j+1]) {
            flag = true
            swap (arr[j], arr[j+1])
        }
    if (!flag) break;
```

→ Stable and Unstable Sort

↳ does not disturb the order of elements with same value.

	40	30	10	20	30*
Stable :-	10	20	30	30*	40
Unstable :-	10	20	30*	30	40

Lecture 36

Selection Sort

⇒ Selection sort Algorithm

"Repeatedly find min. element in unsorted array & place it at beginning"

- Until array is sorted

eg: 5, 8, 4, 9, 2

2 | 8, 4, 9, 5

2, 4 | 8, 9, 5

2, 4, 5 | 9, 8

2, 4, 5, 8 | 9

Sorted

⇒ Selection sort code

```
for (i = 0 ; i < n-1 ; i++) {
    int min_idx = i;
    for (j=i+1; j < n ; j++) {

        if (arr[j] < arr[min_idx]) {
            min_idx = j
        }
    }
    if (min_idx != i) {
        swap (arr[i], arr[min_idx])
    }
```

Code

```
#include <iostream>
#include <vector>
using namespace std;

void selectionSort(vector<int> &V){

    n = V.size();
    for(i=0; i<n-1; i++){
    int min_idx = i                          //Finding min. element in Unsorted array
        for(j=i+1; j<n; j++){
            if(V[j] < V[min_idx]){
                min_idx = j
            }
        }
        if(min_idx != i){
            swap(V[i], V[min_idx])
        }                                    //Placing min. element at beginning.
    }
    return;
}
int main(){
    int n
    cin>>n
    vector<int> V(n);
    for(i=0, i<n, i++){
        cin >> V[i];
    }
    selectionSort(V);
    for(i=0; i<n; i++){
        cout << V[i] << " "
    } cout<<endl;
```

(26)

⇒ Selection Sort Time and Space Complexity

$n-1 + n-2 + \cdots + 2+1$

Sum of 1 to $n-1$

$= n(n-1)/2$

Time complexity = $O(n^2)$

Space complexity = $O(1)$

→ Selection sort is unstable sorting Algorithm

3 4 3* 2

2 4 3* 3

2 3* 4 3

2 3* 3 4

⭐ Applications of Selection Sort

→ Max Swaps O(N)

⭐ in Insertion_Sort (int arr[], int size) { {Tab fill code}

```
int i, j, t;
for (i=1; i < size; i++) {
    j = i;
    while ( j>0 and arr[j] < arr[j-1])
        t = arr[j];
        arr[j] = arr[j-1]
        arr[j-1] = t
    }
}
return arr
}
```

// swaps(arr[j], arr[j-1])

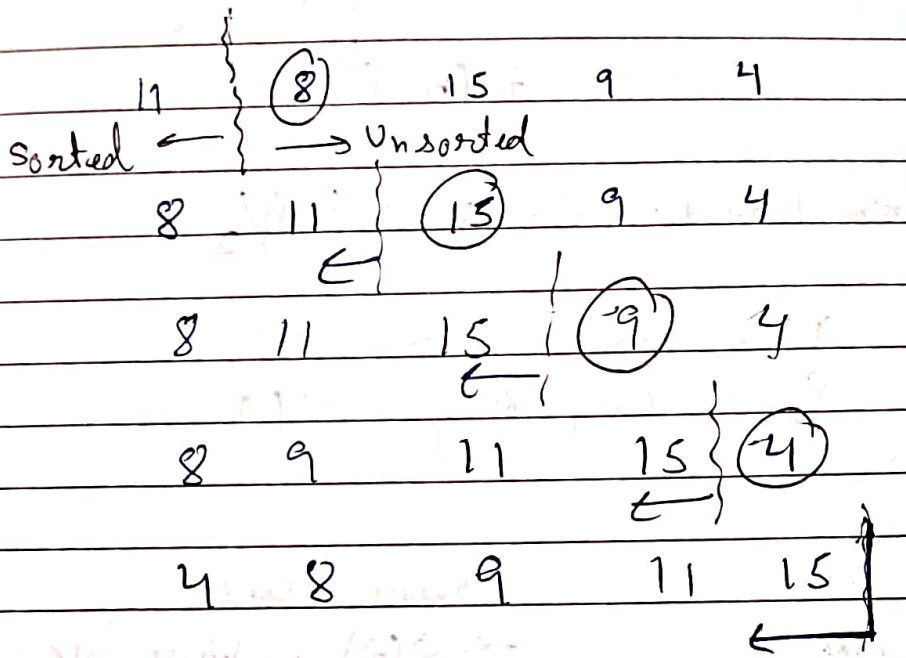
FREEMIND

Date

Page

Lecture - 37

(Insertion Sort)



Repeatedly take elements from Unsorted sub array and insert it in sorted sub-array

correct Position

⇒ Insertion sort code

```
Void & InsertionSort (vector<int> & V) { int n = V.size();
    for (int i = 1; i < n; i++) {
        int current = arr[i];
        int j = i-1;
        while (j>=0 && arr[j] > current ) {   // Find correct Position for over current Element
            arr[j+1] = arr[j];
            j--;
        }
        arr[j+1] = current;   // Insert current Element
    }
    return;
}
int main () {
    { Same as Prev. sorting
}
```

FREEMIND

Date

Page

⇒ Insertion sort time and space complexity.

$1 + 2 + \cdots + (n-1)$

⇒ Sum from 1 to n-1 ⟹ $n(n-1)/2$

Time complexity = $O(n^2)$
Space complexity = $O(1)$

- Selection Sort
  - → $O(n^2)$ in worst case
  - → $\Omega(n^2)$ in best case
- Insertion Sort
  - → $O(n^2)$ in worst case
  - → $\Omega(n)$ in best case

⇒ Applications where
→ Array is almost sorted / few unsorted elements

→ Insertion sort is a stable sorting algo.

3 4 3* 2

3 3* 4 2

No ($3^* > 3$) ~~So, remains same~~

FREEMIND

Date

Page

Lecture - 38

{Problem Solving on Sorting Algo-1}

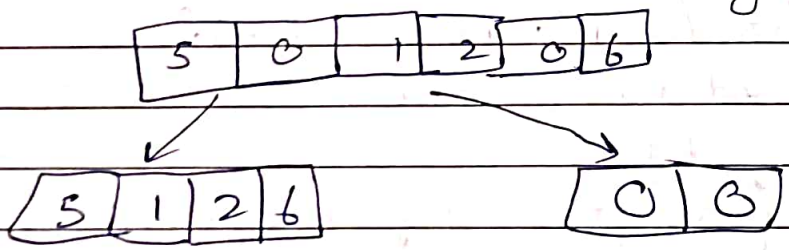
Q:- Given an integer array arr, move all ~~zero~~ 0's to the end of it while maintaining the non-zero elements. (relative order of)

Note that you must do this in place without making a copy of the array.

Input :- 0, 5, 0, 3, 42

Output :- 5, 3, 42, 0, 0

Method 1:- Make two ~~arrays~~ arrays one having non zero elements and one having 0's and combine them



→ But there will be an copy of array So use another method

Method 2:- Bubble Sort :- Jaise max. element ko last me leke jate the waise hi zero to last me leke jayenge

```
void Sort(vector<int>&V)
int i, n = V.size();
for(i = n-1 ; i >= 0 ; i--) {
    bool flag = false ; int j = 0;
    while(arr[j] = 0 && arr[j+1] != 0) {
        swap(arr[j], arr[j+1]);
        flag = true;
        j++;
    } if(!flag) break;
}
```

FREEMIND

Date

Page

Q) Given an array of names of the fruits; you are supposed to sort it in lexicographical order using the selection sort
→ min. Element
→ Place it at begining

input:- ["papaya", "lime", "Watermelon", "apple", "mango", "kiwi"]

output:-

output:- ["apple", "kiwi", "lime", "mango", "papaya", "Watermelon"]

Sol.→

```
#include <iostream>
#include <
using namespace std;

void selectionSort(char fruit[][60], int n){

    for(i=1 ; i<n-1 ; i++){
        int min_idx = i;
        int j;
        for(j= i+1 ; j<n ; j++){

            if(strcmp(fruit[min_idx], fruit[j])>0){
                min_idx = j
            }
        }
        if(i != min_idx){
            swap(fruit[i], fruit[min_idx])
        }
    }
    return;
}
int main(){
    char fruit[][60] = {"papaya", "lime", "Watermelon",
                        "apple", "mango", "kiwi"};
```

```
int n = sizeof (fruit) / sizeof (fruit[0]);
selectionSort (fruit, n);
for (i = 0 ; i < n; i++)
    cout << fruit[i] << " ";
} cout << endl;
return 0;
}
```

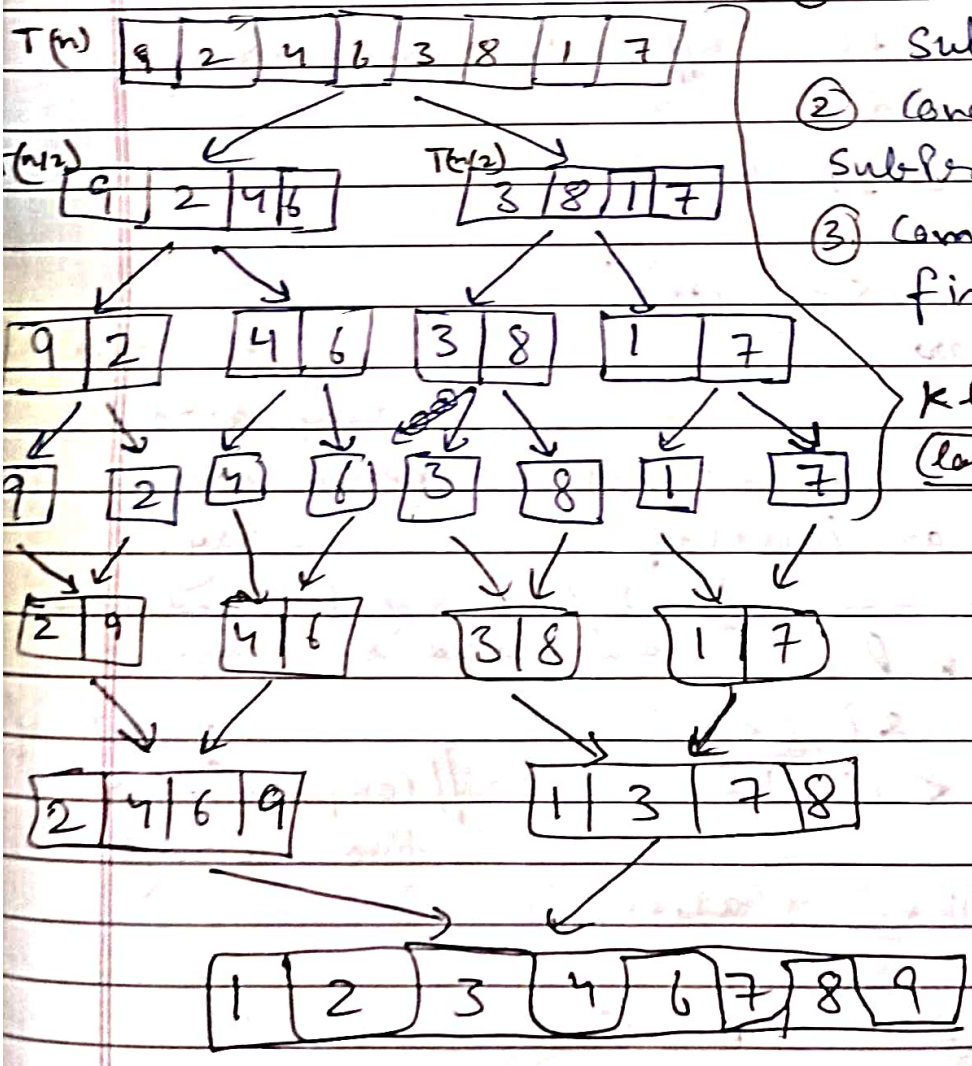
Lecture-39 ★★

(Merge sort)

• Divide & Conquer (DAC)

→ Merge sort algorithm

① Divide problem into sub problems
② Conquer (Solve) the subproblems
③ Combine solutions to get final output.



FREEMIND

Date

Page

⊛ Merge Sort Code

```
mergeSort(arr, l, r) {          // l → 0, r → n-1
    // base case
    if (l >= r) return;

    int mid = (l + r)/2;

    mergeSort(arr, l, mid);     // recursive case
    mergeSort(arr, mid+1, r);
    merge(arr, l, mid, r);
}

merge(arr, l, mid, r){
    int an = mid - l + 1        // create 2 temp. arrays
    int bn = r - mid
    int a[an], b[bn];

    for(int i = l; i < an; i++){
        a[i] = arr[l+i]         // Fill the two temp arrays from original array.
    }
    for(int j = mid+1; j < bn; j++){
        b[i] = arr[mid+1+j]
    }
    int i=0, j=0, k=l;          // initial index of 1st, 2nd array and merged sub array.
    while(i < an && j < bn)     // Compare these two arrays the elements of
        if(a[i] < b[j]){
            arr[k++] = a[i++]
        }else{
            arr[k++] = b[j++]
        }
```

// Jb ek array khtm ho jaegi or dusri bach jaegi Tb dono vala while loop work nahi karega So, single vala use hoga.

```
while (i < an) {
    arr[k++] = a[i++]
}
while (j < bn){
    arr[k++] = b[j++]
}
```

(3)

```
int main () {
    int arr[] = { 9, 2, 4, 6, 3, 8, 1, 7}
    int n = sizeof (arr) / sizeof (arr[0]);

    mergeSort (arr, 0, n-1);

    for (int i=0 ; i<n ; i++) {
        cout << arr[i] << " ";
    } cout << endl;
    return 0;
}
```

(*) Merge sort Time and Space complexity

$T(n) = n + n + n + \text{---} + k \text{ times}$

$T(n) = n \log_2 n$

$\frac{n}{2^k} = 1$

$n = 2^k$

$\log_2 n = k$

Finding Time complexity by checking no. of subarrays and steps included to solve the subarray

Date

Page

Master's Theorem → To Find Time complexity of D A C algos

① $T(n) = a\,T(n/b) + \theta(n^k \log^p n)$

$a \geq 1$, $b > 1$, $k \geq 0$, p = real no.

→ if $\log_b^a > k$, $T(n) = \theta(n \log_b^a)$

if $\log_b^a = k$, if $p > -1$, $\theta(n^k \log^{p+1} n)$

if $p = -1$, $\theta(n^k \log\log n)$

if $p < -1$, $\theta(n^k)$

if $\log_b^a < k$, if $p \geq 0$, $\theta(n^k \log^p n)$

if $p < 0$, $\theta(n^k)$

Example

Merge Sort

$T(n) = 2T(n/2) + n^1$

$a = 2, b = 2, k = 1$

$\log_a^b = 1$, $P = 0$

$\theta(n^1 \log^1 n)$

$\theta(n \log n)$ Ans

→ Space Complexity

Recursive stack → max $\log n$ calls

Merge, temp arrays → N

space complexity → $O(n)$

Time " → $O(n \log n)$

→ Is Merge sort Stable?

Yes

Date

Page

→ Applications

→ Large Data sets

→ Linked List

→ Drawbacks

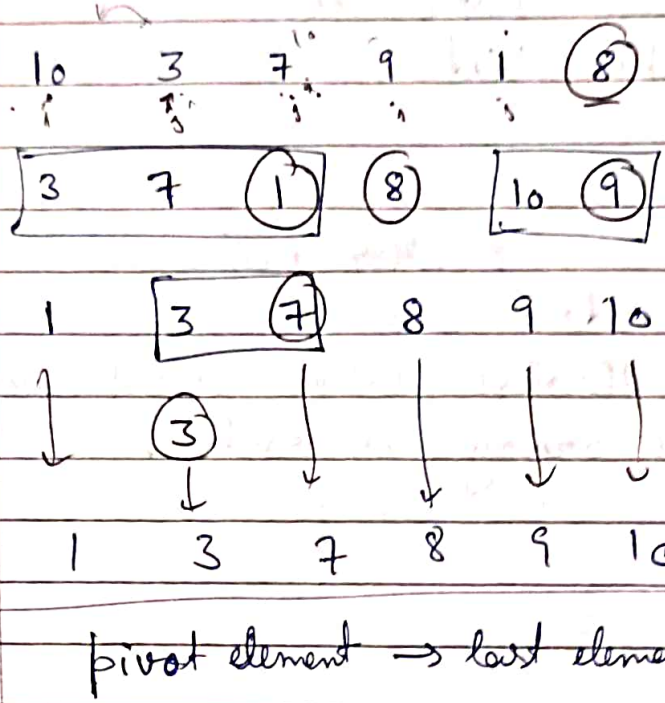
→ slower for smaller tasks

→ O(n) extra space

→ goes through whole process even if array is sorted.

Lecture - 40

(Quick Sort Algorithm)



DAC
Divide & Conquer Rule

1 element to Uski right position par lekar aao fir Uske aas paas k sub arrays me se ek element lo and Usse Uski Sahi position par le jao
→ Repeat Recursively

```
② QuickSort(arr, first, last){
      if(first >= last) return; // base case {Jb array me ek hi element Reh jaye}
      pi = partition(arr, first, last);
      QuickSort(arr, first, pi-1);
      QuickSort(arr, pi+1, last);   // Recursive case
  }
```

Partition Algorithm

```
① partition(arr, first, last){
      pivot = arr[last];
      i = first-1;
      for(j = first; j < last-1; j++){
          if(arr[j] < pivot){
              i++
              swap(arr[i], arr[j])
          }
      }
      //Now i is pointing to the last element < pivot
```

? Problems → Agar i^{th} element hi pivot se chota hua to swap kaise karenge Vo to same ho jaega → ~~swap~~ swap karne pe same rahega change nahi hota

```
// correct position for pivot will be i+1
    swap (arr[i+1], arr[last])
    return i+1;
}
```

③
```
int main () {
    int arr[] = {10, 3, 7, 9, 1, 2}
    int n = sizeof(arr)/sizeof(arr[0]);

    quickSort(arr, n);

    for (int i=0; i<n; i++){
        cout<<arr[i]<<" ";
    } cout<<endl;
    return 0;
}
```

Quick Sort Time and Space Complexity

Best case → $T(n)$

5 3 4 1 2 (n)

1 2 | 3 | 5 4

$T\left(\frac{n-1}{2}\right)$ ↑ $T\left(\frac{n-1}{2}\right)$

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

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

⇒ Acc. to Master's Theorem

Time complexity = $\Theta(n \log n)$ (Best Case)

1 2 3 4 (5) n
1 2 3 (4) n-1
n-2
1

Date
Page

Worst case

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

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

Time complexity = $O\left(\frac{n(n+1)}{2}\right)$ (Worst case)

$\Rightarrow O(n^2)$

- Space complexity = $O(n)$

(*) Need for new ways of partioning and randomised quicksort algorithm

But stable
Median Nikalne k liye bhi To sort krna pdeda so, choose

Median → pivot → $n \log n$

Randomised pivot

- pivot = first + ~~random~~ rand() % (last - first + 1)

⇒ Is Quick Sort Stable?
Not Stable

⇒ Applications.
→ Memory is a concern
→ inbuilt sorting algorithms.

⇒ Merge Sort V/s Quick Sort

↓	↓
→ Large datasets	→ Smaller datasets
→ Stable	→ unstable
→ Linked lists	→ New memory is costly

FREEMIND

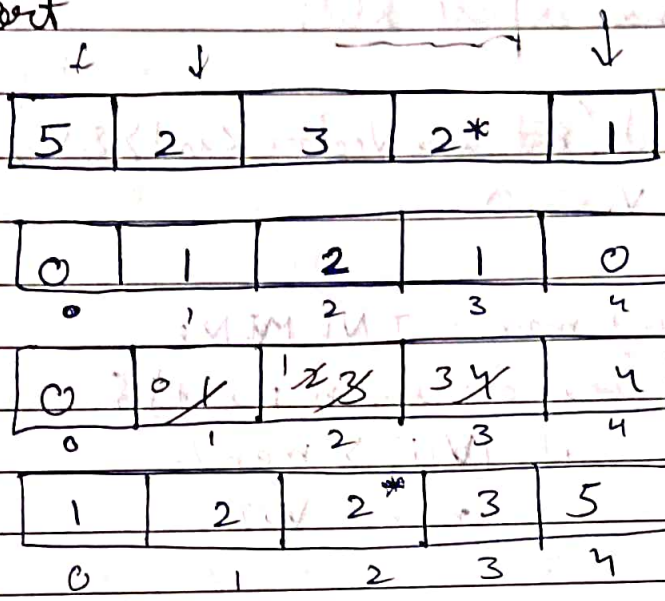
Date

Page

Lecture - 41

(Advanced Sorting algorithms)

(A) Count Sort



① ye Piche se pointer chalega or jo bhi element hoga Apni Commulative freq. k acc. minus hokr apni position par aajayega

(A) Count Sort code :-

```
CountSort(arr, n){
① // finding max element
      max;
② // finding frequency } int freq[max+1];
   for (i=0 to n-1)
      freq[arr[i]]++;
③ //calculate commulative freq.
   for (i=0 to max)
      freq[i] += freq[i-1]
④ // ans array
      int ans[n];
   for [i=n-1 to 0] {
      ans[--freq[arr[i]]] = arr[i]
   }
}
```

FREEMIND

Date

Page

#Code

```
#include <iostream>
#include <vector>
using namespace std;

void CountSort(Vector<int>& V) {
    n = V.size()

    int max = INT_MIN;
// max element  for(int i=0; i<n; i++){
                    if [V[i] > max]{
                        max = V[i]
                    }
                }

// freq.          Vector<int> freq(max+1, 0)
                for (i=0; i<; i++){
                    freq [V[i]] ++;
                }

// commulative freq.  for (i=0; i<max; i++){
                        freq[i] += freq[i-1];
                    }

// ans array          Vector<int> ans(n);
                    for (i=n-1, i>=0, i--){
                        ans[--freq[V[i]]] = V[i];
                    }

                    for (i=0; i<n; i++){
                        V[i] = ans[i];
                    }

}
```

```
int main () {
    int n;
    cin>> n;
    Vector <int> a(n);

    for(int i=0; i<n; i++) {
        cin>> a[i];
    }
    count Sort (a);

    for(int i=0; i<n; i++) {
        cout << a[i] << " ";
    }
    cout << endl;
    return 0;
}
```

Time and Space complexity

Ts O(n) Ss O(n+k)

Applications

→ Max. element is the order of n

Drawbacks

→ Can't use for floating no.s

→ For -ve values we have to normalize the array. (Phle -ve element ko minus karo Phr final sorted array mein vapas no. ko add karo)

→ Can't use when there is a disparity in no.s.

Radix Sort

	170	45	75	90	802	2	Sort by ↓ (Taking ones digit)
1st Pass ⇒	170	90	802	02	45	75	(Tens Digit)
2nd Pass ⇒	802	002	045	170	075	090	(Hundredth Digit)
3rd Pass ⇒	002	045	075	090	170	802	

Radix Sort Code

```
#include <iostream>
include <Vector>
using namespace std;

Void Count Sort (Vector <int> &V, int pos) {
  // Creat freq array
     int n = V.size();
     Vector <int> freq (10,0)
     for (int i=0 ; i<n; i++) {
          freq[(V[i]/pos)%10]++
     }
//Commulative freq.
     for (int i = 0 ; i<10 ; i++) {
          freq[i] += freq[i-1] ;
     }
// ans array
     Vector <int> ans (n);
     for (i = n-1 ; i>=0 ; i--) {
          ans [freq [(V[i]/pos) %10]] = V[i]
     }
```

```
// copy to original Array
for ( i = 0 ; i < n , i++) {
    V[i] = ans[i];
}

Void RadixSort (Vector<int> &V) {
    // max Element
    int max_ele = INT_MIN ;
    for (auto x : V) {
        max_ele = max( x, max_ele );
    }
    // Count Sort
    for (int pos = 1 ; max_ele/pos > 0 ; pos *= 10){
        countSort (V, pos);
    }
}

int main() {   (same as previous)
    n - enter
    Vector -> initialize & enter

    RadixSort (V)
    Vector -> print
    return 0;
}
```

Time complexity and Space complexity.

$O(d*(n))$ — $O(n)$

FREEMIND

Date

Page

Lecture - 41.2

(Bucket Sort)

1) Create buckets of size n
2) Insert elements into Buckets
   = index = arr[i] * n
3) Sort individual buckets
4) Combine all elements

{Scatter & Gather Approach}

{0.13, 0.45, 0.12, 0.89, 0.75, 0.13, 0.85, 0.39}

0	→ 0.12
1	→ 0.13
2	
3	→ 0.45 → 0.39
4	
5	→ 0.63
6	→ 0.75 → 0.85
7	→ 0.89
8	

{index = arr[i] * size}

Time Complexity

Best Case → O(n)
Avg Case → O(n+k)
Worst Case → O(n²)

Space Complexity

O(n+k)

Bucket Sort Code


```
# include <iostream>
# include <vector>
Using namespace std;

Void    BucketSort (int arr[] , int Size) {

//Step1 =>     Vector <Vector< float>> buckets( size , Vector <float>() );

// Step2 =>     for (int i=0; i< size ; i++) {
                   int index = arr[i] x size;
                      bucket [index]. pushback( arr[i])
                }

//Step3 => Sorting Individual Buckets
                for (int i=0; i< size ; i++) {
                   if (! bucket[i].empty()) {
                      sort ( bucket[i]. began() , bucket[i].end());
                   }
               }
//Step4 => Combinig elements from buckets
               int k=0
               for (i=0 ; i< size ; i++ ) {
                   for (int j=0 ; j< buck[i].size ; j++) {
                       arr [k++] = bucket [i] [j] ;
                   }
               }
}
int main() {
       float arr[] = { - - - ____ }
       int size = sizeof (arr) / sizeof(arr[0]);
       bucketSort (arr, size);
       for (i=0; i< size; i++) {
            Cout << arr[i] << " ";
       }
       return 0;
```

- When elements present in the array are greater than 1 the the product arr[i] * size will exceeds the array size. So, For solving this we have a something different approach

```
=> Void BucketSort(int arr[], int size) {

// Step 1 => vector
    Vector<Vector<float>> bucket(size, vector<float>());
    // Finding Range
    float max_ele = INT_MIN arr[0];
    float min_ele = INT_MAX arr[0];
    for (int i=0; i<size; i++) {
        max_ele = max(max_ele, arr[i]);
        min_ele = min(min_ele, arr[i]);
    }
    float range = (max_ele - min_ele)/size;

// Step 2 => Inserting elements into Bucket.

    for (int i=0; i<size; i++) {
      int index = (arr[i] - min_ele)/range;

      // boundary elements (max value)

// Value index range and diff = zero hojaega
      float diff = (arr[i] - min_ele)/range - index;

      if (diff==0 && arr[i] != min ele) {
          bucket[index-1].pushback(arr[i]);
      }
      else {
          bucket[index].pushback(arr[i]);
      }
    }
// Step 3 -> Step 4 -> Inf main same
```

FREEMIND

Date

Page

Lecture - 42

Q: Write a program to find k^{th} smallest element in an array using quicksort.

input :- Enter elements for array
3 5 2 1 4 7 8 6
Enter Value for k = 5

Output :- k^{th} smallest element is 5.

(concept)

Sol: ~~k^th smallest (arr, l, r, k~~

```
int partition (int arr[], int l, int r) {
    int pivot = arr[r];
    int i = l;
    for (int j=l, j<r, j++) {
        if (arr[j] < pivot) {
            swap (arr[i], arr[j]);
            i++;
        }
    }
    swap (arr[i+1], pivot);
    return i;
}
int k^th smallest (int arr[], int l, int r, int k)

    if (k>0 && k<=r-l+1) {
    int pos = partition (arr, l, r);
    if (pos-l == k-1) {
        return arr[pos];
    }
```

Date

Page

```
else if (pos-l > k-1) {
        kth smallest (arr, l, pos-1, k);
}
else {
        kth smallest (arr, pos+1, r, k-pos+l-1)
}
}
return ~~INTmax~~ INT_MAX;

}
int main() {
        arr ———
        cin n → size of (arr)/ size of (arr[0])
        k = 5

        kth smallest (arr, 0, n-1, k);

        print arr

}
```

(#inputs)

⇒ Time Complexity :-

Avg. Case → O(n)

Worst Case → O(n²)

FREEMIND

Date

Page

Q1 Given two sorted arrays, Write a program to merge them in sorted

input:- num1[] = {5, 8, 10}, num2[] = {2, 7, 8}

output:- num3[] = {2, 5, 7, 8, 8, 10}

Sol- void Merge(int arr1[], int n1, int arr2[], int n2, int arr3[]) {

```
int a = 0;
int b = 0;
int c = 0;

while (a < n1 && b < n2){
    if (arr1[a] < arr2[b]){
        arr3[c++] = arr1[a++];
    } else {
        arr3[c++] = arr2[b++];
    }
}while (a < n1) {
    arr3[c++] = arr1[a++];
}
while (b < n2) {
    arr3[c++] = arr2[b++]
```

```
int main() {
    int arr1 ——
        arr2 ——
        n1 -> size of (arr1)/ ——
        n2 ->          (arr2)/ ——
    int n3 = n1 + n2
    int arr3[n3]
    merge (arr1, n1, arr2, n2, arr3);
    print arr3
}
```