Counting Sort

non comparision sort

→ Sorting according to keys.

→ Counting the elements having distinct key values

n=17
k=9

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
a	2	1	1	0	2	5	4	0	2	8	7	7	9	2	0	1	9	9

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

Initialize count with 0

```
for(i=0; i<n; i++)
{
  ++count[a[i]];
}
```

	0	1	2	3	4	5	6	7	8	9
count	3	6	10	10	11	12	12	14	15	17
Actual position	2	5	9							16

```
for(i=1; i<=k; i++)
{
  count[i]=count[i]+count[i-1];
}
for(i=n-1; i>=0; i--)
{
  b[--count[a[i]]]=a[i];
}
```

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

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

→ Scan the original array from right to left

→ start from the right side to maintain the stability of sort

```
for(i=0; i<n; i++)
{
  a[i]=b[i];
}
```

upper bound on k should be O(n)

$2n$
$3n$
but not n^2

Time complexity will be O(n+k) (Linear time)

→ k value should be feasible

e.g. if Array has 100 element and k=10000 the size of count will be 0 to 10000 with 100 element

will not work with -ve values and floating values

Counting Sort →

Counting Sort is a sorting Technique based on keys between a specific range. It works by counting the number of objects having distinct key values. Then doing some arithmatic to calculate the position of each object in the output sequence

e.g.

1 0 2 1 0 1 1 5 6 7 5 4 2 2 0 0

n=17

Range = 0-7

K=7

K different elements

① $0 \leq a[j] \leq k$

② $a[j] \in I$

Radix Sort

15 1 321 10 802 2 123 90 109 11

Pass 1

0	010, 090
1	001, 321, 011
2	802, 002
3	123
4	
5	015
6	
7	
8	
9	109

Pass 2

0	001 802 002 109
1	010 011 015
2	321 123
3	
4	
5	
6	
7	
8	
9	090

Pass 3

0	001 002 010 011 015 090
1	109 123
2	
3	321
4	
5	
6	
7	
8	802
9	

010, 090, 001, 321, 011, 802, 002, 123, 015, 109

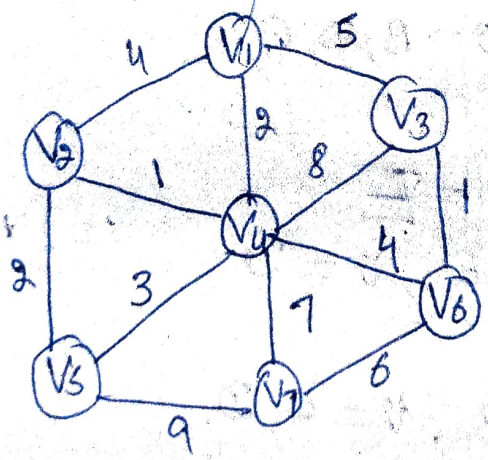
Pass 1 010, 090, 001, 321, 011, 802, 002, 123, 015, 109

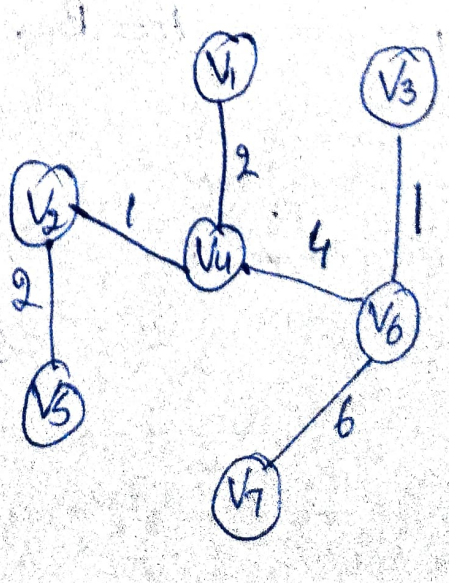
Pass 2 001 802 002 109 010 011 015 321 123, 090

Prim's Algo

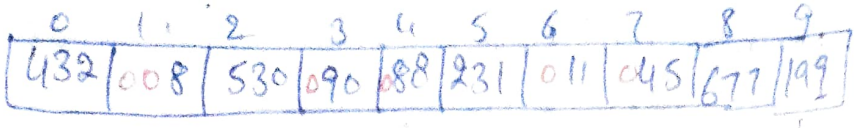
```
MST (G, w, r)
1  for each u ∈ V[G]
2        do key[u] = ∞
3           π[u] = NIL
4  key[r] = 0
5  Q = V[G]
6  while Q ≠ ϕ
7     do u ← EXTRACT-MIN(Q)
8           for each v ∈ Adj[u]
9              do if v ∈ Q and w(u,v) < key[v]
                  then π[v] = u
                  key[v] = w(u,v)
```

Prims

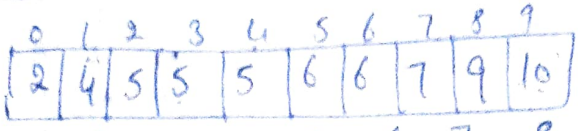




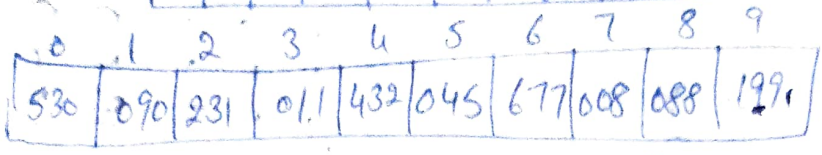
=16 Unit



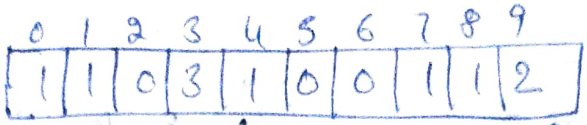
Pass1

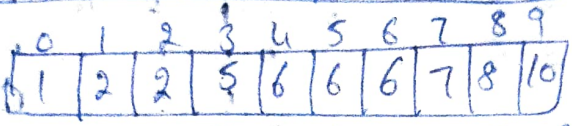


o/p

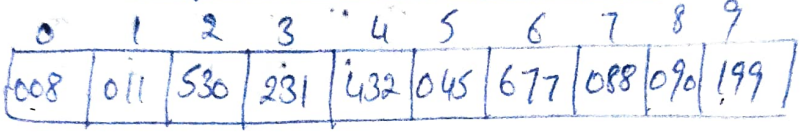


Pass 2

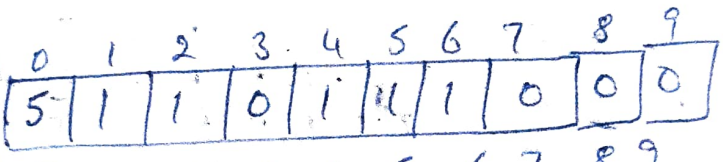


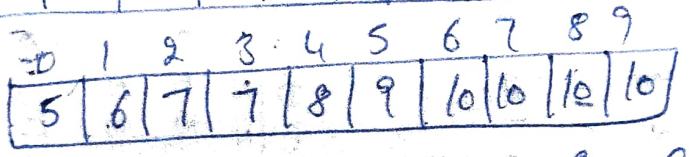


o/p

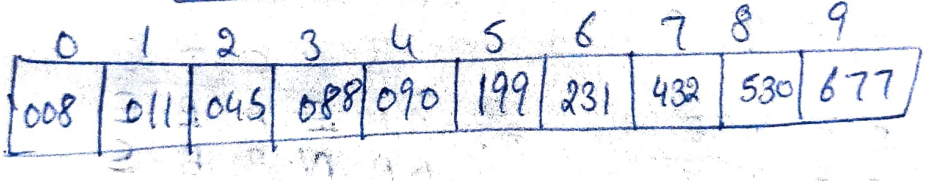


Pass 3





o/p




```
radixsort (a, n)
{
  int max = getmax(A, n)
  for(pass=1; max/pass > 0; pass*10)
  [
    {
      countsort (A, n, pass)
    }
  ]
}
```

O(n+k)

O(d*(n+k))

```
countsort (int A[], int n, int pass)
{
  int count[10] = {0},
  for(i=0; i<n; i++)
    ~~count[a[i]]~~,
    ~~count[a[i]]++~~
  ++count[(a[i]/pass)%10],
```

```
for(i=1; i<=k; i++)
{
  count[i] = count[i] + count[i-1],
}
for(i=n-1; i>=0; i--)
{
  b[--count[(a[i]/pass)%10]] = a[i]
}
}
```

Shell Sort →

23 29 15 19 31 7 9 5 2

15 19 23 29 31 7 9 5 2

Efficiency of shell sort depends on gap sequence. The better the gap, the better shell sort will perform.

gap = n/2

0	1	2	3	4	5	6	7	8
23	29	15	19	31	7	9	5	2

$gap = \lfloor \frac{9}{2} \rfloor = 4$

Pass 1:

23 7 15 19 31 29 9 5 2

23 7 9 19 31 29 15 5 2

23 7 9 5 31 29 15 19 2

23 7 9 5 2 29 15 19 31 — Now compare 2 with previous element with gap of 4

2 7 9 5 23 29 15 19 31

Pass 2: $gap = \frac{4}{2}$

2 7 9 5 23 29 15 19 31

2 5 9 7 23 29 15 19 31

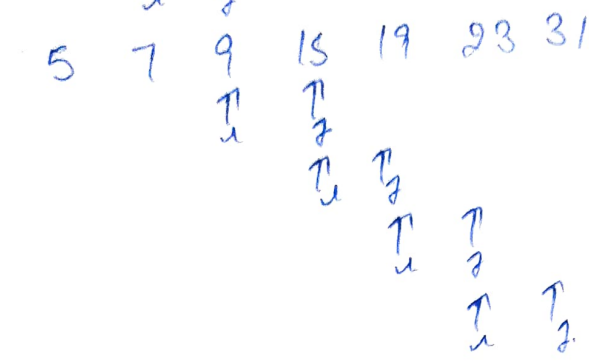
2 5 9 7 15 29 23 19 31

2 5 9 7 15 19 23 29 31

Pass 3 — gap = 1

2 5 9 7 15 19 23 31

Check Pervious → 2 5 7 9 15 19 23 31



```
for (gap = n/2; gap >= 1; gap = gap/2)
{
    for (j = gap; j < n; j++)
    {
        for (i = j - gap; i >= 0; i = i - gap)
        {
            if (a[i+gap] > a[i])
            {
                break;
            }
            else
            {
                swap(a[i+gap], a[i]);
            }
        }
    }
}
```

Q Consider the situation where swap operation is very costly which of the following algorithms should be preferred so that the number of swapping are minimized in general.

a) Heap
b) Selection
c) Insertion
d) Merge

Ans Selection (O(n) swapping)

3 2_a 2_b 1

4 3 2_a 2_b 1

```
Void SelectionSort(int a[], int n)
{
    int i, j, min_idx;
    for(i=0; i<n-1; i++)
    {
        min_idx = i;
        for(j=i+1; j<n; j++)
        {
            if (a[j] < a[min_idx])
            {
                min_idx = j;
            }
        }
        swap(&a[i], &a[min_idx])
    }
}
```