Searching And Sorting

Searching :- It is a technique of searching a desired element in a sequence.

> It is of 2 types - (1) sequential / linear search
(2) Binary search.

1) Linear Search :- In this case desired element is matched with one by one element of the array from the beginning.

'C' function :-

```
int seq_search(int a[], int key, int n)
{
int i;
for(i=0; i<n; i++)
  {
    if(a[i]==key)
      return(i);
  }
return(-1);
}
```

Time complexity :- $T(n) = O(n)$

2) Binary Search :-

> Sequential Search is simple method for searching an element in an array but it takes more time.

> Binary search is a very efficient search technique which works for sorted list.

> We make a comperision between key and the middle element of the array.

> Array should be sorted for this type of search.

> If the search element "key" is less than middle element then searching is done in left half of ~~the~~ ~~middle~~

> If the search element "key" is greater than middle value then searching is done in right half.

'c' fun^c :-

```
int binary_search(int a[], int n, int key)
{ int lb, hb, mid;
  lb=0;
  hb=n-1;
  while(lb <= hb)
  {
    mid=(lb+hb)/2;
    if (key == a[mid])
      return(mid);
    else if (key < a[mid])
      hb=mid-1;
    else
      lb=mid+1;
  }
  return(-1);
}
```

Algo:

```
binary_search(a[], n, key)
1) declar lb, hb, mid
2) set lb=0, hb=n-1
3) Repeat step 4 to 6 till lb<=hb
      set mid=(lb+hb)/2
4)    if key = a[mid]
        return(mid)
5)    else if key < a[mid]
        set hb = mid-1
6)    else
        set lb = mid+1
   [end of if structure]
 [end of while loop]
7) Exit return(-1)
8) Exit.
```

Time complexity = T(n) = log n

Explanation :-

Q) 23, 4, 5, 32, 10, 15, 28, 30, 35

Sort :

4	5	10	15	23	28	28	30	32	35
0	1	2	3	4	5	6	7	8	

1) Search 10 :

mid=(0+8)/2 = 4

a[4] > 10)

So element is present at left half

So set hb = mid-1 = 4-1 = 3

again calculate mid = (0+3)/2 = 1

or

a[1] < 10

So element present at right half

set lb = mid+1 = 1+1 = 2

mid = (2+3)/2 = 2.

a[2] == 10

So element found & present at 2.

(ii) Search 29 :

mid = (0+8)/2 = 4

a[4] < 29

So set lb = mid+1 = 4+1 = 5

calculate mid

mid = (5+8)/2 = 6

a[6] > 29

So set hb = mid-1 = 6-1 = 5

mid = (5+5)/2 = 5

a[5] < 29

Set lb = 5+1 = 6

now lb > hb

So loop will stop & ctrl comes out loop & return -1.

Sorting :- Arranging the elements in ascending or descending order.

Techniques :- 1) Bubble Sort, 2) Insertion Sort, 3) Selection Sort, 4) Quick Sort, 5) Merge Sort, 6) Heapsort, 7) Radix Sort.

0	1	2	3	4	5	6	7
44	55	33	88	77	22	11	66

1) **Bubble Sort** :- Sequence:

Method :-

Pass. 0 :-

i) i=0, compare a[0] and a[1] since 44 < 55 no swapping

ii) i=1, compare a[1] and a[2] since 55 > 33, swapping required, new sequence is

44	33	55	88	77	22	11	66
0	1	2	3	4	5	6	7

iii) i=2, compare a[2] and a[3] since 55 < 88, no swapping

iv) i=3, compare a[3] and a[4] since 88 > 77, swapping required.

44	33	55	77	88	22	11	66
0	1	2	3	4	5	6	7

v) i=4, compare a[4] and a[5] since 88 > 22, swap

44 33 55 77 ~~88~~ 88 11 66

vi) i=5, compare a[5] and a[6] since 88 > 11, swap

44	33	55	77	22	11	88	66

vii) i=6, compare a[6] and a[7] since 88 > 66, swap

44	33	55	77	22	11	66	88

Pass 1 :-

i) i=0, compare a[0] and a[1] since 44 > 33, swap

33	44	55	77	22	11	66	88

ii) i=1, compare a[1] and a[2] since 44 < 55, No swap

iii) i=2, compare a[2] and a[3] since 55 < 77, No swap.

iv) i=3, compare a[3] and a[4] since 77 > 22, swap

33 44 55 22 77 11 66 88

v) i=4, compare a[4] and a[5] since 77 > 11 swap

33 44 55 22 11 77 66 88

vi) i=5, compare a[5] & a[6] since 77 > 66, swap
33 44 55 22 11 66 77 88

pass 2

i) i=0, compare a[0] & a[1], no swap
ii) i=1, compare a[0] & a[2], no swap
iii) i=2, compare a[2] & a[3], swap 33 44 22 55 11 66 77 88
iv) i=3, compare a[3] & a[4], swap, 33 44 22 11 55 66 77 88
v) i=4, compare a[4] & a[5], no swap, 33 44 22 11 55 66 77 88

pass 3

i) i=0, compare a[0] & a[1], no swap
ii) i=1, compare a[1] & a[2], swap 33 22 44 11 55 66 77 88
iii) i=2, compare a[2] & a[3], swap, 33 22 11 44 55 66 77 88
iv) i=3, compare a[3] & a[4], no swap, 33 22 11 44 55 66 77 88

pass 4

i) i=0, compare a[0] & a[1], swap 22 33 11 44 55 66 77 88
ii) i=1, compare a[1] & a[2], swap 22 11 33 44 55 66 77 88
iii) i=2, compare a[2] & a[3], no swap.

pass 5

i) i=0, compare a[0] & a[1], swap 11 22 33 44 55 66 77 88
ii) i=1, compare a[1] & a[2], no swap.

pass 6

i) i=0, compare a[0] & a[1], no swap

Finally

11	22	33	44	55	66	77	88

Cfunc

```
void BubbleSort(int a[], int n)
{ int i, temp, pass;
for( pass=0; pass<n-1; pass++)
 {for( i=0; i<n-pass-1; i++)
    { if( a[i] > a[i+1])
       { temp = a[i];
         a[i] = a[i+1];
         a[i+1] = temp;
       }
    }
 }
```

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

Algo

BubbleSort(a[], n)

1) declar i, temp, pass = 0
2) Repeat step 3 to 9 for pass < n-1
3) set i = 0
   Repeat step 4 to 8 for i < n-pass-1
4) if a[i] > a[i+1]
5) set temp = a[i]
6) set a[i] = a[i+1]
7) set a[i+1] = temp
8) set i = i+1
   end of for loop
9) set pass = pass+1
   end of for loop
10) Exit.

2) Insertion Sort :- It sorts a set of values by inserting values into existing sorted file.

30 20 35 14 90 25 32

Pass 1: a[1] < a[0], interchange the element position of the elements

20	30	35	14	90	25	32

Pass 2: a[2] > a[1], no interchange of the position of the element.

Pass 3: a[3] is less than a[0], a[1] and a[2], so insert a[3] before a[0]

14	20	30	35	90	25	32

Pass 4: a[4] > a[3] no change is performed

14	20	30	35	90	25	32

Pass 5: a[5] is less than a[2], a[3] and a[4], so insert a[5] before a[2].

14	20	25	30	35	90	32

Pass 6: a[6] is less than a[4], a[5] therefore insert a[6] before a[4],

14	20	25	30	32	35	90

C func :-

```
void InsertionSort(int a[], int n)
{
  int i, j, key;
  for(j=1; j<n; j++)
  {
    key = a[j];
    i = j-1;
    while((i>-1) && (a[i] > key))
    {
      a[i+1] = a[i];
      i = i-1;
    }
    a[i+1] = key;
  }
}
```

Algo:

InsertionSort(a[], n)

1) declar i, j, key
2) set j = 1
   Repeat step3 to 7 for j<n
3) set key = a[j]
   set i = j-1,
4) Repeat step 5 & 6 while i>-1 AND a[i] > key
5) set a[i+1] = a[i]
6) set i = i-1
   end of while loop
7) set a[i+1] = key
8) set j = j+1
   end of for loop
9) Exit.

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

3) Selection Sort :-

It require n-1 pass to Sort

Method :

20	35	40	100	3	10	5

Pass 1 : find the smallest element a[4], interchange a[4] with a[0].

3	35	40	100	20	10	5

Pass 2 : Find the smallest element a from a[1] to a[6] ie a[6] interchange a[6] with a[1]

3	5	40	100	20	10	35

Pass 3 : find the smallest element from a[2] to a[6] ie a[5] interchange a[5] with a[2]

3	5	10	100	20	40	35

Pass 4 : find the smallest element from a[3] to a[6] ie a[4] interchange a[3] with a[4]

3	5	10	20	100	40	35

Pass 5 : find the smallest element from a[4] to a[6] ie a[6] interchange a[6] with a[4]

3	5	10	20	35	40	100

Pass 6 : find the smallest element from a[5] to a[6] ie a[5] interchange a[5] with a[5]

3	5	10	20	35	40	100

```
void SelectionSort(int a[], int n)
{
int i, j, pos, min;
for(i=1; i<=n-1; i++)
  {min = a[i-1];
   pos = i-1;
  for(j = i; j<=n-1; j++)
    {if(a[j] < min)
      {min = a[j];
       pos = j;
      }
    }
a[pos] = a[i-1];
a[i-1] = min;
}
}
```

Algo :-

```
SelectionSort(a[], n)
1) declar i, j, pos, min
2) set i = 1
3) Repeat step          for i<=n-1
     set min = a[i-1]
4)   set pos = i-1
5)   set j = i
6)   Repeat step        for j<=n-1
7)     if a[j] < min
8)       set min = a[j]
9)       set pos = j
10)    end of if structure
       set j = j+1
       end of for loop
11)  set a[pos] = a[i-1]
     set a[i-1] = min
     set i = i+1
     end of for loop
12) Exit.
```

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

4) **Quick Sort** :- It is the fastest sort compared to another algorithm. It uses devide & conquire algorithm.

Method:- [9] 3 7 65 12 18 92 11 8 5 32
Pivot i j

[9] 3 7 65 12 18 92 11 8 5 32
Pivot i j

[9] 3 7 65 12 18 92 11 8 5 32
i j

[9] 3 7 65 12 18 92 11 8 5 32
i j

[9] 3 7 65 12 18 92 11 8 5 32
i j

as i < j so swap a[i] with a[j]

[9] 3 7 5 12 18 92 11 8 65 32
i j

As i < j so i

[9] 3 7 5 12 18 92 11 8 65 32
i j

as i < j. so swap a[i] with a[j]

[9] 3 7 5 8 18 92 11 12 65 32
i j

As i < j

[9] 3 7 5 8 18 92 11 12 65 32
i j

[9] 3 7 5 8 18 92 11 12 65 32
i j

[9] 3 7 5 8 18 92 11 12 65 32
i j

[9] 3 7 5 8 18 92 11 12 65 32
i j

[9] 3 7 5 8 18 92 11 12 65 32
i j

[9] 3 7 5 8 18 92 11 12 65 32
j i

as i < j so swap a[j] and a[pivot]

8 3 7 5 [9] 18 92 11 12 65 32 —(1)

consider now left sub part of 9.

[8] 3 7 5
Pivot i j

[8] 3 7 5
i j

8 3 7 5
i j

8 3 7 5
i j

As i < j so swap a[pivot] & a[j]

5 3 7 8 —(ii)

consider left subpart of 8

5 3 7
pivot j

5 3 7
pivot i j

5 3 7
pivot i j

5 3 7
pivot i j

5 3 7 i > j
j e

as i < j so swap a[j] with a[pivot]

5 7 —(iii)

Now consider right subpart of 9

18 9.2 11 12 65 32
pivot j

18 92 11 12 65 32
pivot i j

18 92 11 12 65 32
pivot i j

18 92 11 12 65 32
pivot i j

18 92 11 12 65 32
pivot i j

as i < j so again start

18 12 11 92 65 32
i j

18 12 11 92 65 32
i j

18 12 11 92 65 32
i j

as i > j so swap a[j] & a[pivot]

18 12 11 92 65 32
i j

18 12 11 92 65 32
i j

as i < j so again start

18 12 11 92 65 32
i j

18 12 11 92 65 32
i j

as i > j so swap a[pivot] & a[j]

18 12 11 92 65 32 —(iv)

18 12 18 92 65 32

Consider left subpart of 18

11 12
pivot j

11 12
pivot i j

11 12
j i

as j < i swap a[j] & a[pivot]

11 12 —(v)

Now consider right part of

92 65 32
pivot j

92 65 32
pivot i j

92 65 32
pivot i j

92 65 32
pivot i j

92 65 32
pivot i j

as i < j swap a[pivot] & a[j]

32 65 92 —(vi)

Now consider left subpart of

92

32 65
pivot j

32 65
pivot j
i

32 65
j
i

32 65
j i

swap a[pivot] with a[j]

32 65 —(vii)

Combine all the sorted part (iii), (ii), (i), v, iv, vii, vi

3 5 7 8 9 11 12 18 32 65 92

c) func :-

```
void QuickSort(int x[], int first, int last)
{
int pivot, j, temp, i;
if (first < last)
 {
  pivot = first;
  i = first;
  j = last;
while(i<j)
 {while((x[i]<=x[pivot]
      &&(i<last))
  i++;
while(x[j]>x[pivot])
  j--;
if(i<j)
 {
  temp = x[i];
  x[i] = x[j];
  x[j] = temp;
 }
 }
temp = x[pivot];
x[pivot] = x[j];
x[j] = temp;
QuickSort(x, first, j-1);
QuickSort(x, j+1, last);
 }
}
```

Algorithm :-

QuickSort(x, first, last)

1) declar pivot, j, temp, i

2) if first < last

set pivot = first

set i = first

set j = last

3) Repeat step 4 to 8 while i < j

4) Repeat step 5 while

x[i] <= x[pivot] AND i < last

5) Set i = i+1

end of while loop at step 4

6) Repeat step 7 while loop :-

x[j] > x[pivot]

7) set j = j-1

end of while loop at step 6

8) if i < j

set temp = x[i].

set x[i] = x[j]

set x[j] = temp

end of if structure at step 8

end of while loop at step 3

9) set temp = x[pivot], set x[pivot] = x[j], set

10) QuickSort(x, first, j-1); x[j] =

11) QuickSort(x, j+1, last);

end of if structure at step 2.

12) Exit.

Time complexity : $T(n) = O(n \log n)$

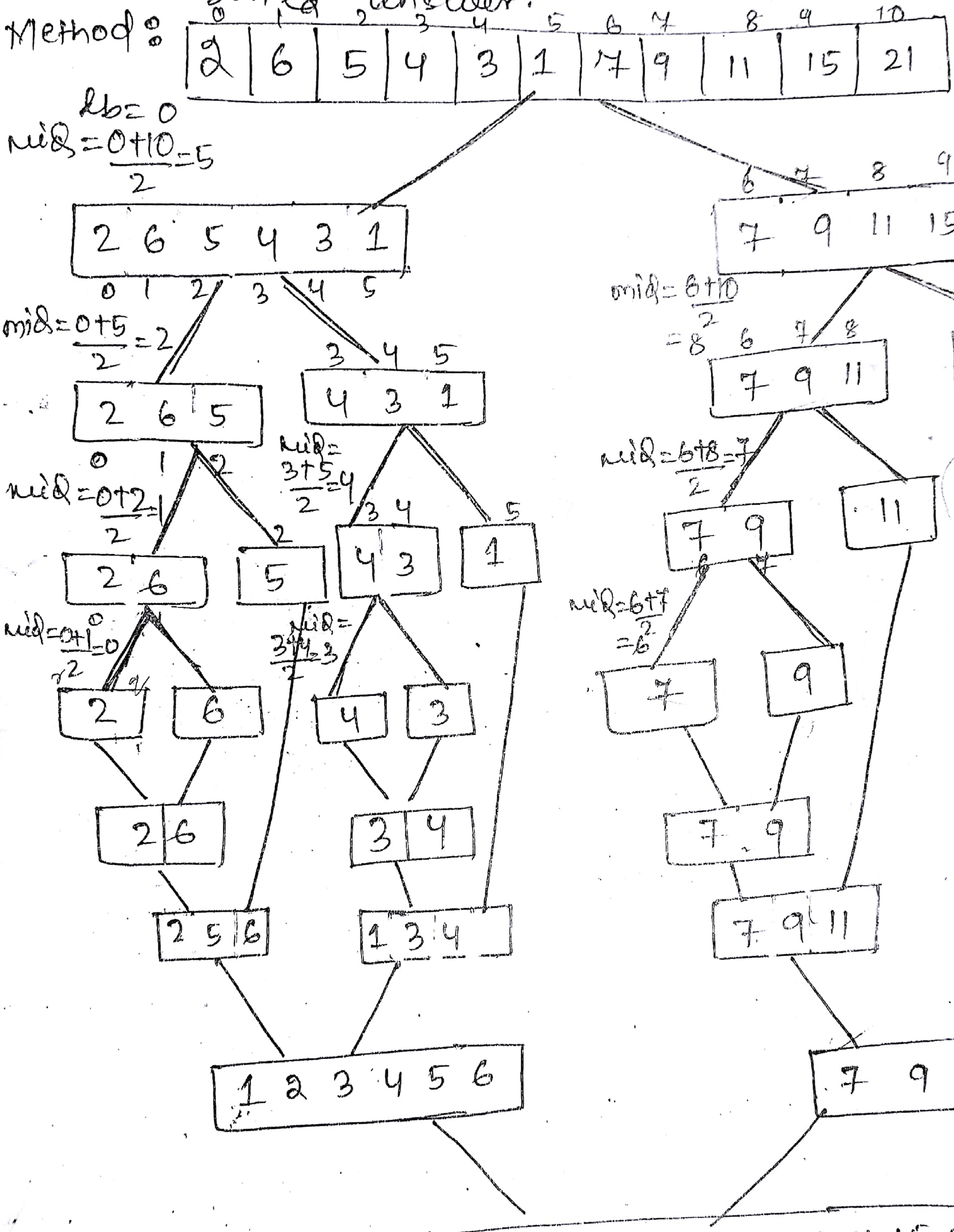
5) Merge Sort :- Merge sort algorithm closely follows the divide & conquer method.

Divide: Divide the n elements sequence to be sorted into two subsequences of n/2 elements each.

conquer: Sort the two subsequence recursively using mergesort

combine: Merge the two sorted subsequence to produce the sorted answer.

Method :-



3 7 11

2 4 5 8

'c' function :-

```
void MergeSort(int a[], int p, int r)
{
int q;
 if (p<r)
  {
  q=(p+r)/2
  MergeSort(a,p,q);
  MergeSort(a,q+1,r);
  Merge(a,p,q,r);
  }
}
void Merge(int a[], int p, int q, int r)
{ int b[20],l1,r1,i;
  l1 =p;
  r1 =q+1;
  i =p;
while((l1<=q)&&(r1<=r))
{if(a[l1]<a[r1])
    {b[i]=a[l1];
      l1=l1+1;
     i=i+1;
    }
else
  {b[i]=a[r1];
    r1=r1+1;
    i=i+1;
  }
}
while(l1<=q)
  {b[i]=a[l1];
   l1=l1+1;
   i=i+1;
  }
while(l1<=r)
  {b[i]=a[r1];
   r1=r1+1;
    i=i+1;
  }
```

Algo :-

MergeSort(a[], p, r)

1) declar q
2) if p < r
   set q = (p+r)/2
   MergeSort(a, p, q)
   MergeSort(a, q+1, r)
   end of if structure
3) Exit

Merge(a, p, q, r)

1) declar b[20], l1, r1, i;
2) set l1=p, r1=q+1, i=p
3) Repeat step 4 & 5
   while l1<=q && r1<=r
4) if a[l1] < a[r1]
   set b[i]=a[l1]
   set l1 = l1+1
   set i = i+1
5) else
   set b[i] = a[r1]
   set r1 = r1+1
   set i = i+1
   end of if structure
   end of while loop
6) Repeat step 7 to 9
   while l1 <= q
7) set b[i] = a[l1]
8) set l1 = l1+1
9) set i = i+1
   end of while loop.
10) Repeat step 11 to 13
    while r1 <= r
11) set b[i] = a[r1]
12) set r1 = r1+1
13) set i = i+1

```
for(i=p; i<=r; i++)
    a[i] = b[i];
}
```

14) Repeat step 15 for i <= r
15) set a[i] = b[i]
    set i = i+1
16) Exit.

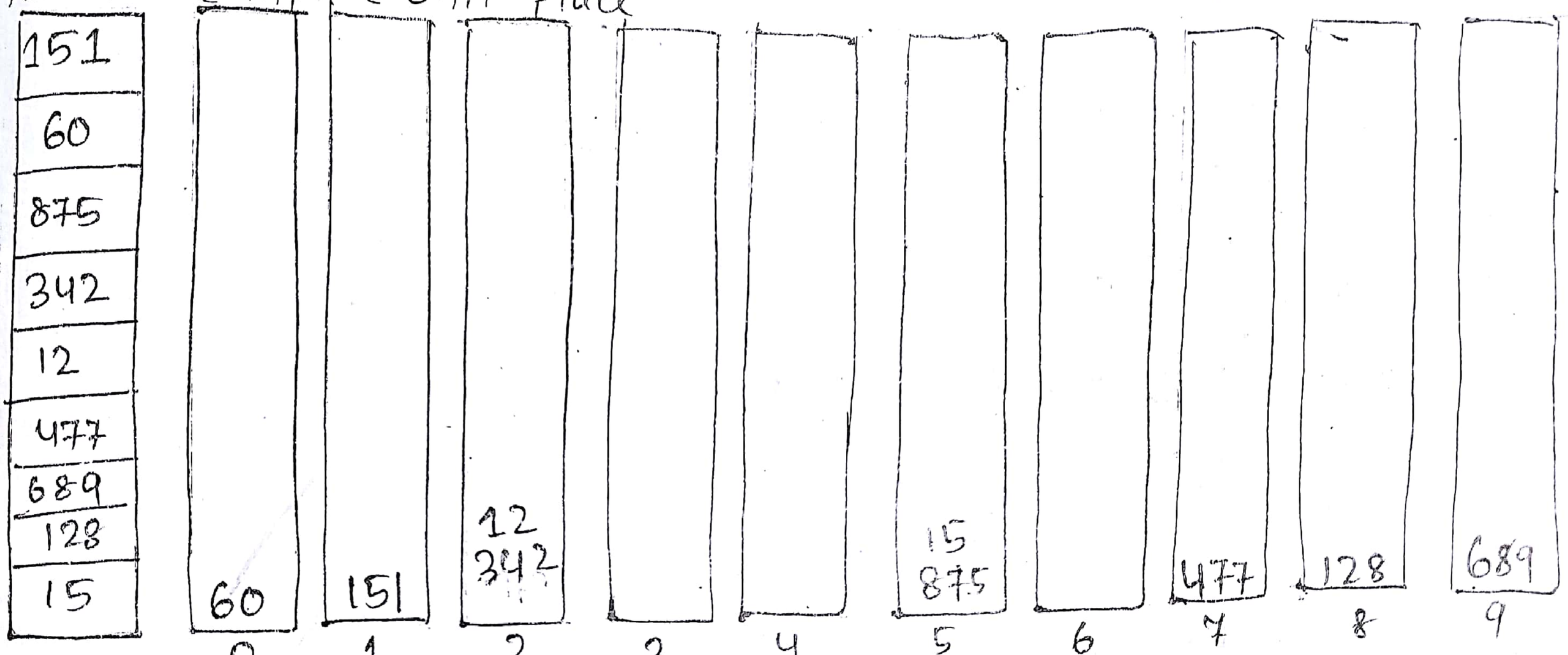
Time Complexity T(n) = O(n log n)

6) Radix Sort : (or Bucket Sort)

This Sorting is based on the values of the actual digits in the positional rep^n of the numbers being sorted.

Method: 151, 60, 875, 342, 12, 477, 689, 128, 15

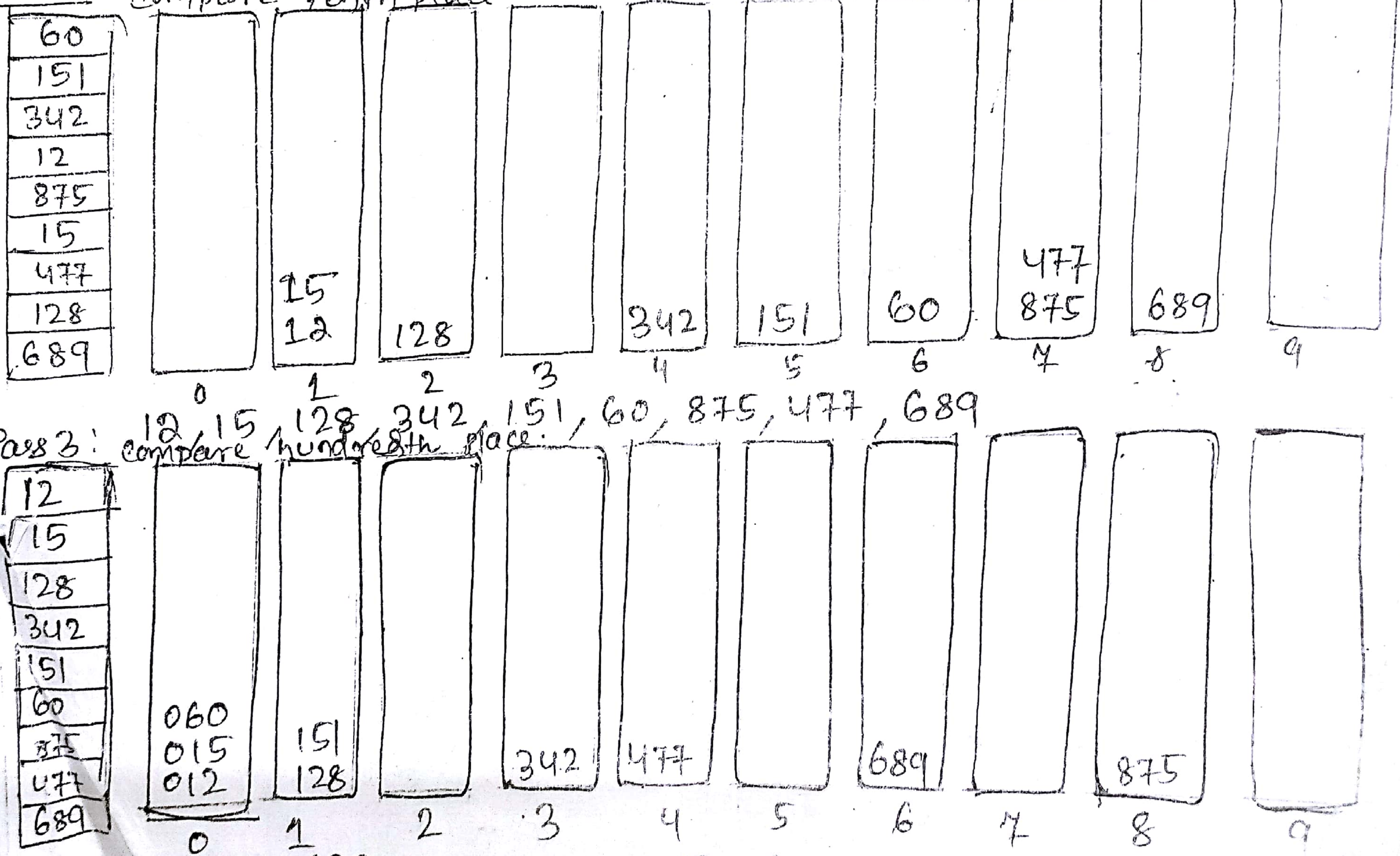
Pass-1 compare unit place



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

60, 151, 342, 12, 875, 15, 477, 128, 689.

Pass 2 compare tenth place.



12, 15, 128, 342, 151, 60, 875, 477, 689

Pass 3: compare hundredth place.

12, 15, 60, 128, 151, 342, 477, 689, 875

7) **Heapsort:** Heap datastructure is an array object that can be viewed as an almost complete binary tree.

> Each nodes of the tree corresponds to an element of the array that stores the value in the node & value of each internal node is greater than or equal to the val of its children.

Maintaining heap property:

*Heapify (a, i)

1) set l = left(i)
2) set r = right(i)
3) if l ≤ heap_size[a] − 1 and a[l] > a[i]
4) then set largest = l
5) else set largest = i
6) if r ≤ heap_size[a] − 1 and a[r] > a[largest]
7) then set largest = r
8) if largest ≠ i
9) then swap a[i], a[largest] and
10) Heapify (a, largest)

*BuildHeap(a)

1) set heapsize[a] = length[a]
2) repeat step 3 for i = (length[a] − 1)/2 down to 0
3) Heapify(a, i)

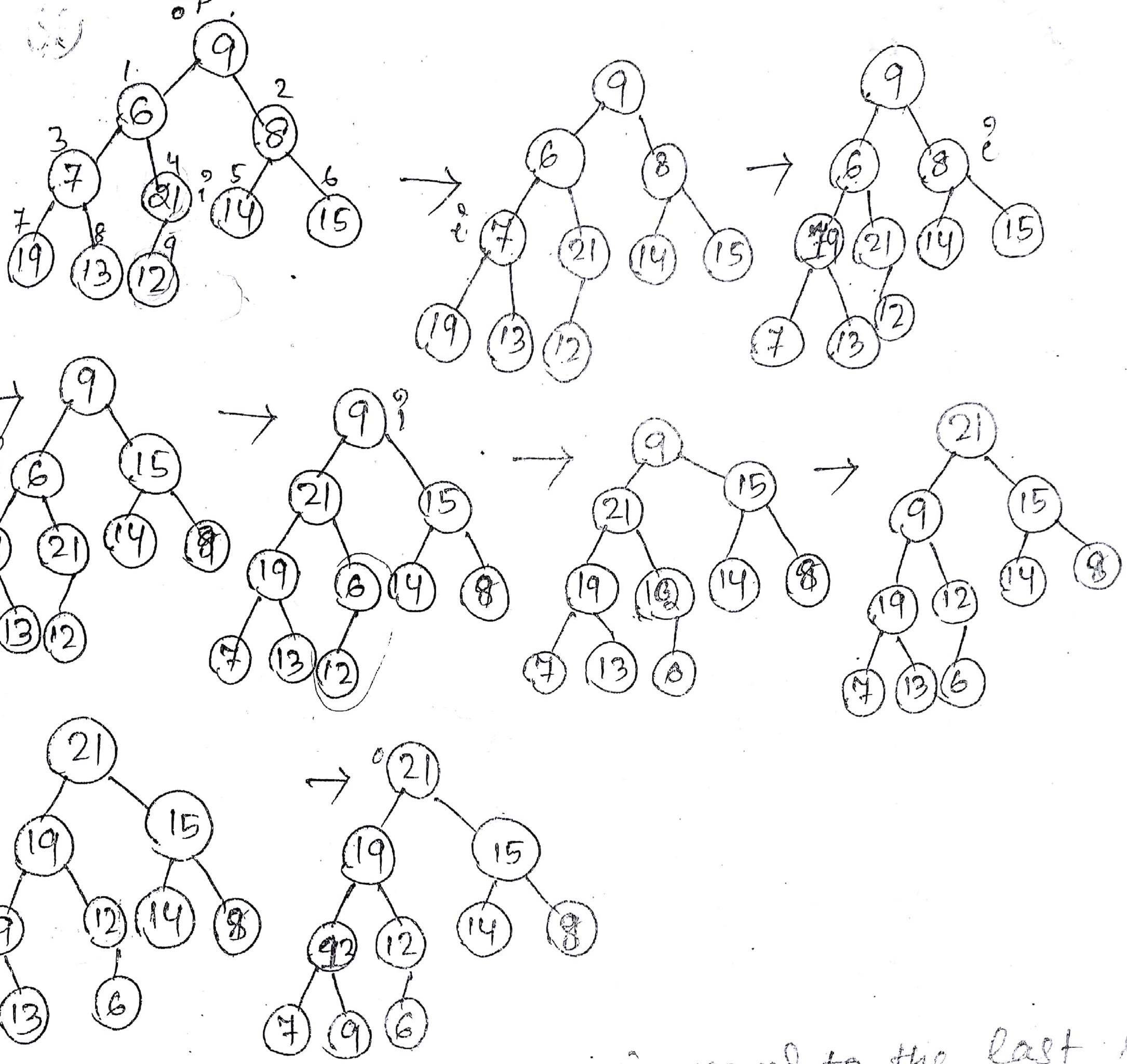
*HeapSort(a)

1) BuildHeap(a)
2) repeat steps 3 to 5 for i = length[a] − 1 down to 1
3) swap (a[0], a[i])
4) set heapsize[a] = heapsize[a] − 1
5) Heapify(a, 0)

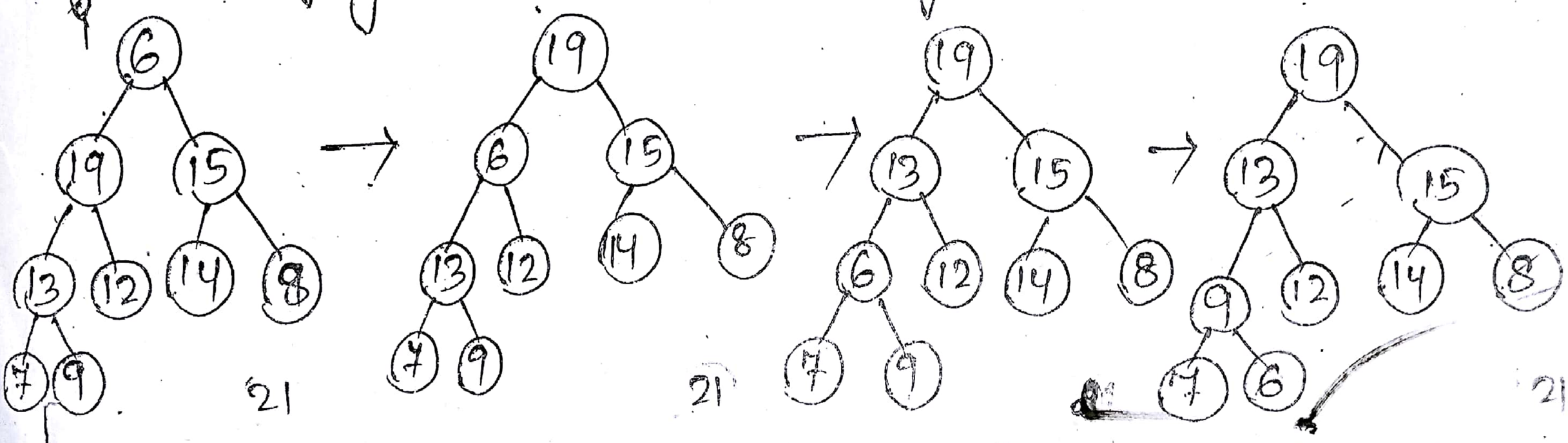
Q) Sort the sequence by using heapsort.

9 6 8 7 21 14 15 19 13 12.

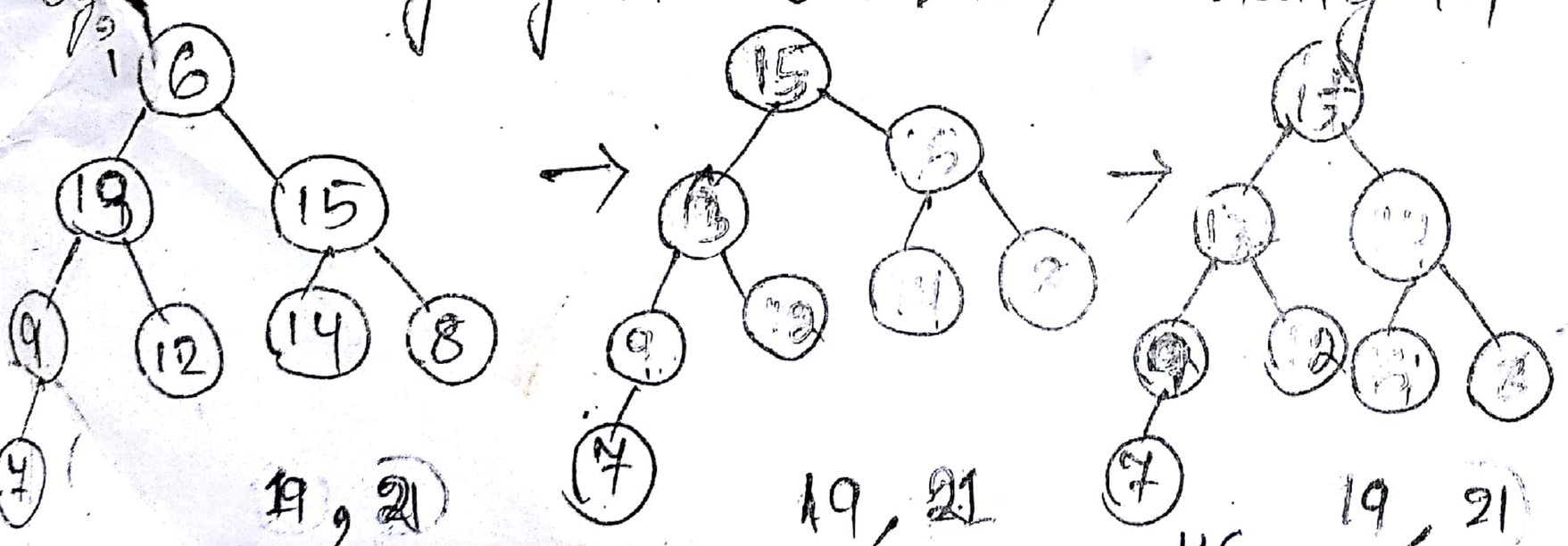
Build Heap:



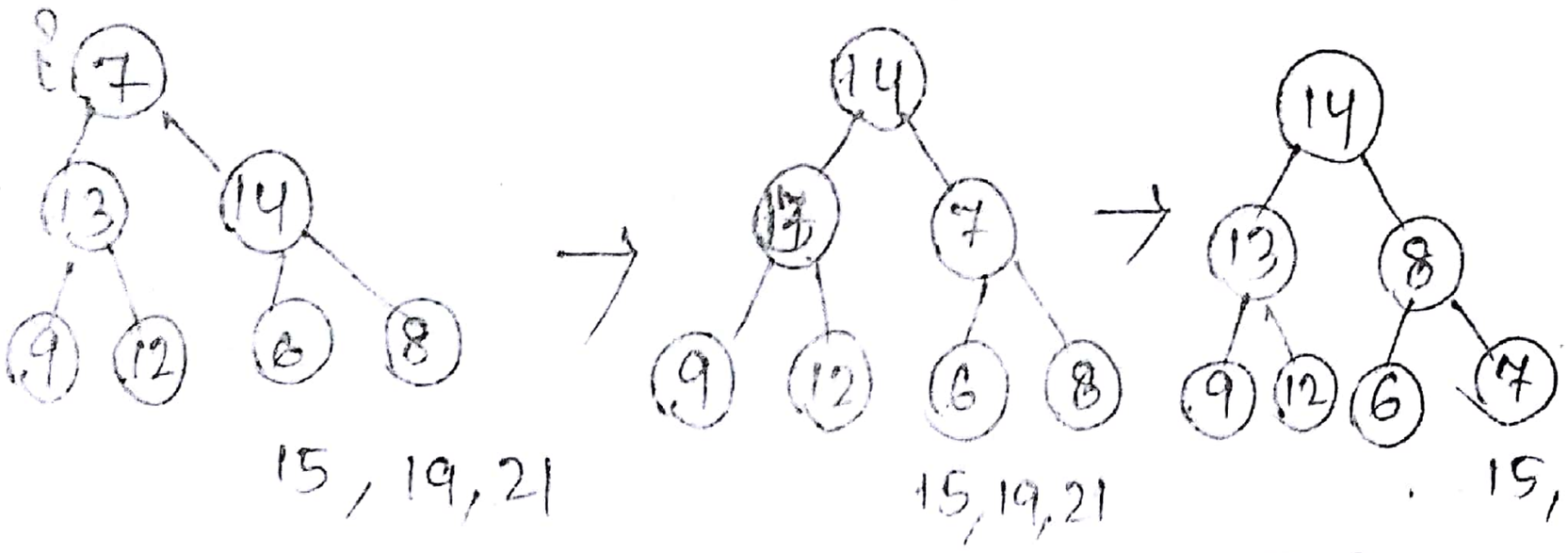
Now the root element 21 is moved to the last location by exchanging it with 6. Finally 21 is eliminated



Now the root element 19 is moved to the last location by exchanging it with 6, finally 19 is eliminated.

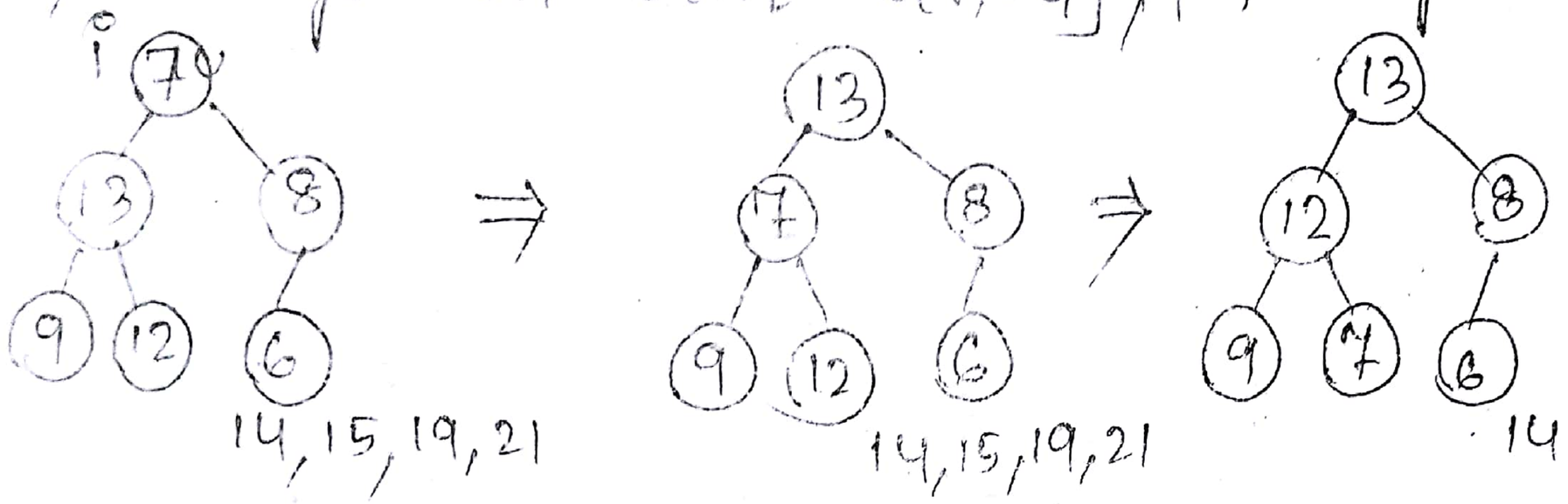


Now the root element 15, is moved to the last location by exchanging it with 7, finally 15 is eliminated.



15, 19, 21 15, 19, 21 15, 19, 21

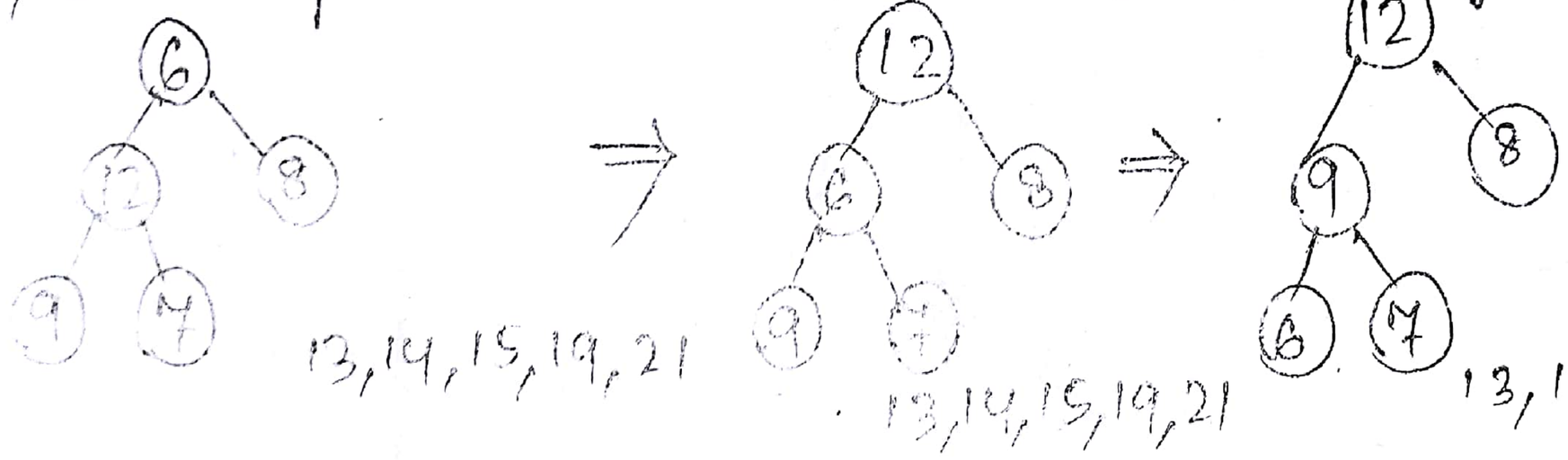
→ Exchange root with a[n-4], & finally 14 is eliminated



14, 15, 19, 21 14, 15, 19, 21 14, 15, 19, 21

2)

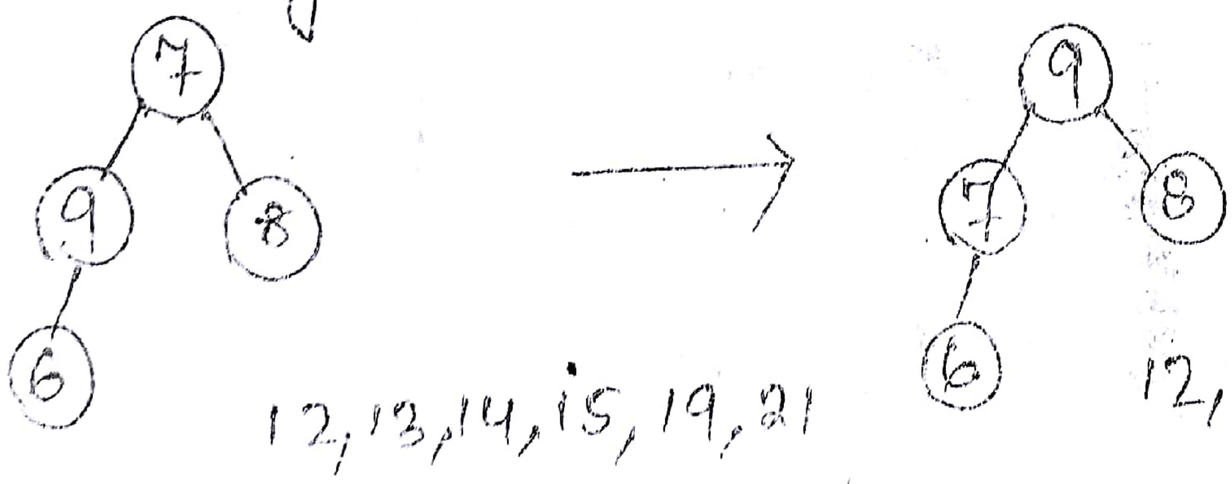
→ Exchange root with a[n-5] & finally 13 is eliminated



13, 14, 15, 19, 21 13, 14, 15, 19, 21 13, 14, 15, 19, 21

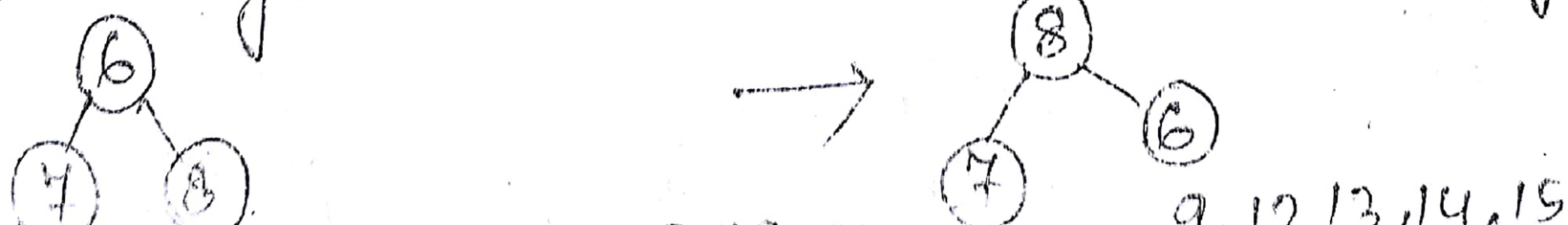
→ Exchange root with a[n-6] & finally 12 is eliminated

3)



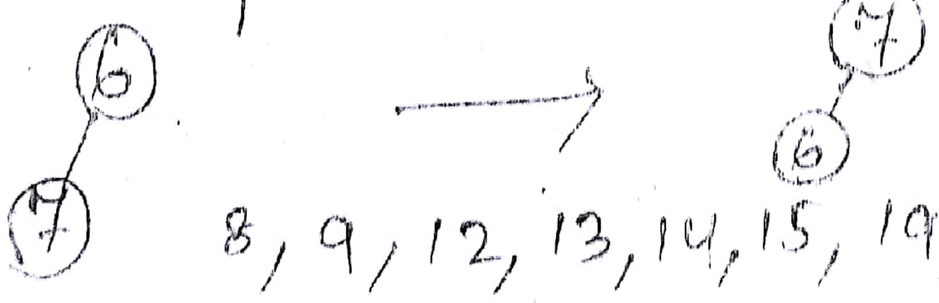
12, 13, 14, 15, 19, 21 12, 13, 14, 15, 19, 21

→ Exchange root with last element; & finally 9 is eliminated



9, 12, 13, 14, 15, 19, 21 9, 12, 13, 14, 15, 19, 21

→ Exchange root with last element & finally 8 is eliminated



8, 9, 12, 13, 14, 15, 19, 21

→ Exchange root with last element & finally 7 is eliminated

7, 8, 9, 12, 13, 14, 15, 19, 21

→ Finally 6, 7, 8, 9, 12, 13, 14, 15, 19, 21