UNIVERSITY PARTNER
UNIVERSITY OF WOLVERHAMPTON

iAcademy
Quality Education Improving Lives

HERALD COLLEGE KATHMANDU

Academic Year	Module	Assessment Number	Assessment Type
S20	Introductory Data Structures and Algorithms (DipIT02)	A1	Assignment Submission

[Assignment Submission]

Student Id : [NP03A190299]
Student Name : [Yogesh Shrestha]
Section : [DC8]
Module Leader : [Mr. Prakash Gautam]

Submitted on : 06-03-2020

Tutorial - 6 & 7

1). 7, 2, 1, 6, 8, 5, 3, 9, 21, 4

Soln:

Here, Quick sort code.

```
Quick_Sort( A, start, End){
 if (start < End ){
 Pindex = Partition( A, start, End)
Quick_Sort (A, start, Pindex-1)
Quick_Sort (A, Pindex+1, End)
 }
}

Partition ( A, start, End){
    Pivot = A[End]
    Pindex = start;
  for (i = start to End -1){
   if (A[i] <= pivot ){
   Swap ( A[i], A[ Pindex]);
   Pindex = Pindex+1;
  }
  Swap(A[ Pindex], A[End]);
   return pindex;
   }
```

P-index = 0

a (i)

7	2	1	6	8	5	3	9	21	4
0	1	2	3	4	5	6	7	8	9

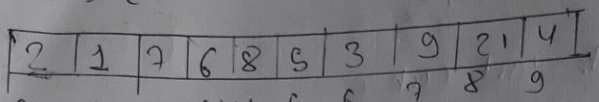
7	2	1	6	8	5	3	9	21	4
0	1 (i)	2	3	4	5	6	7	8	9

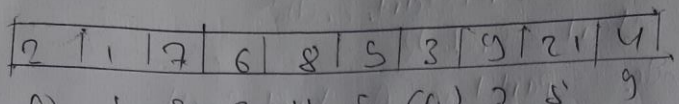
P - index = 1

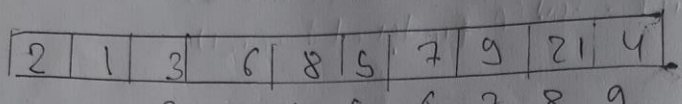
now, Swap (A[i], A[P-index])

2	7	1	6	8	5	3	9	21	4
0	1	2(i)	3	4	5	6	7	8	9

P - index = 2

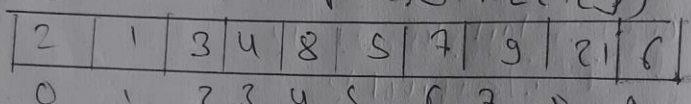




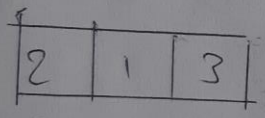


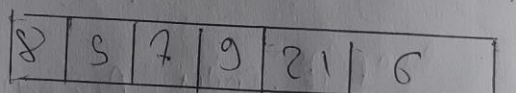
Pindex - 3

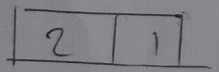
Again swap(A[pindex], A[End])

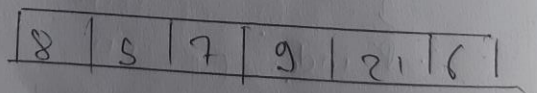


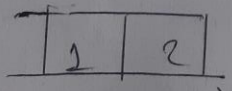
Now the correct position of Pivot element is.

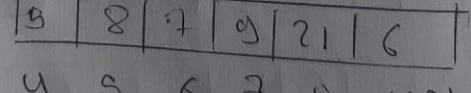


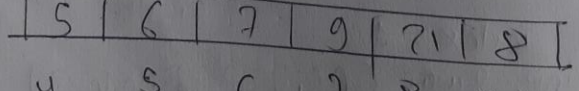


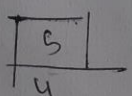


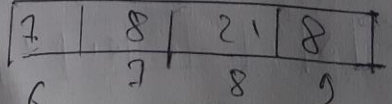


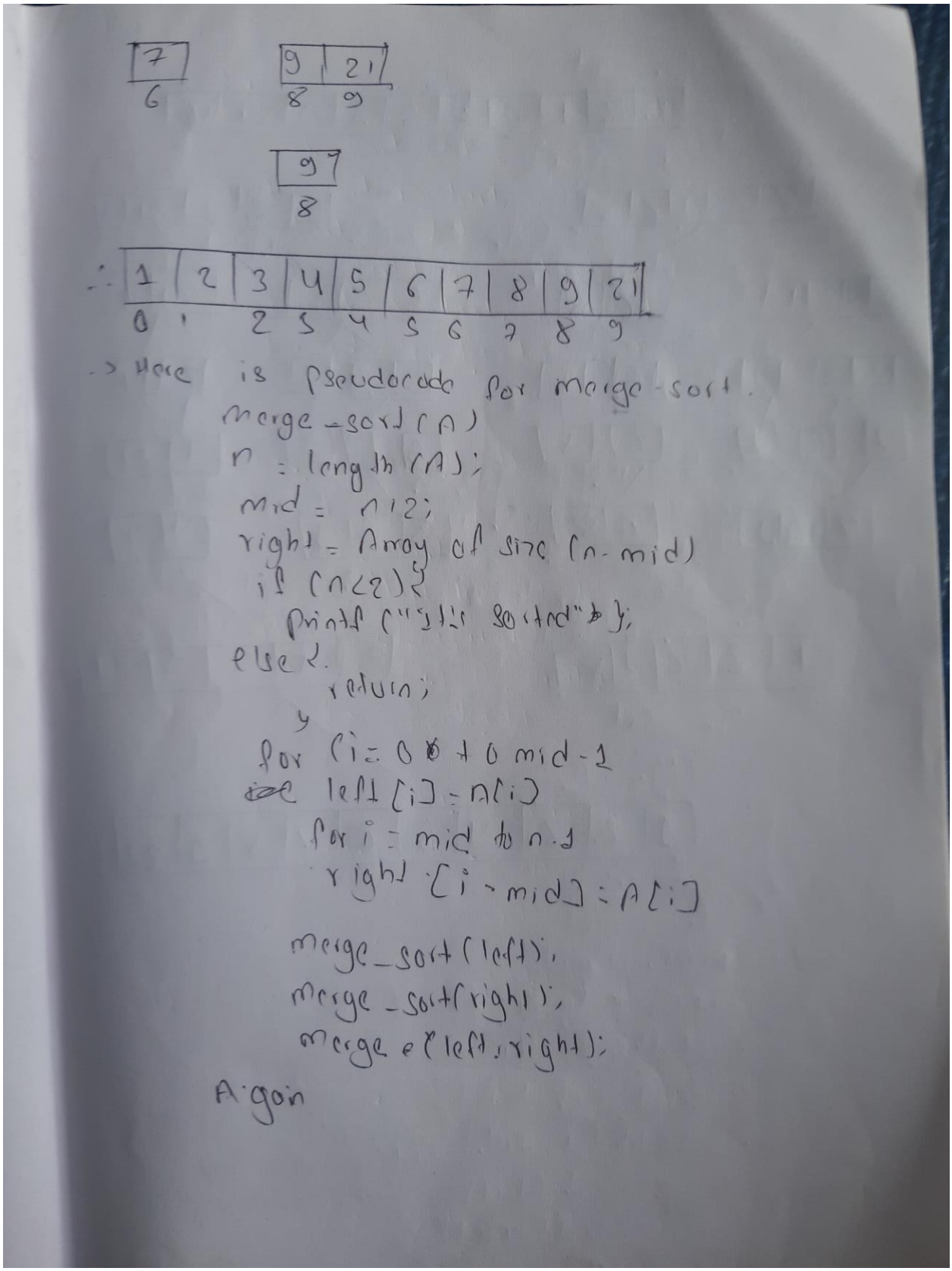


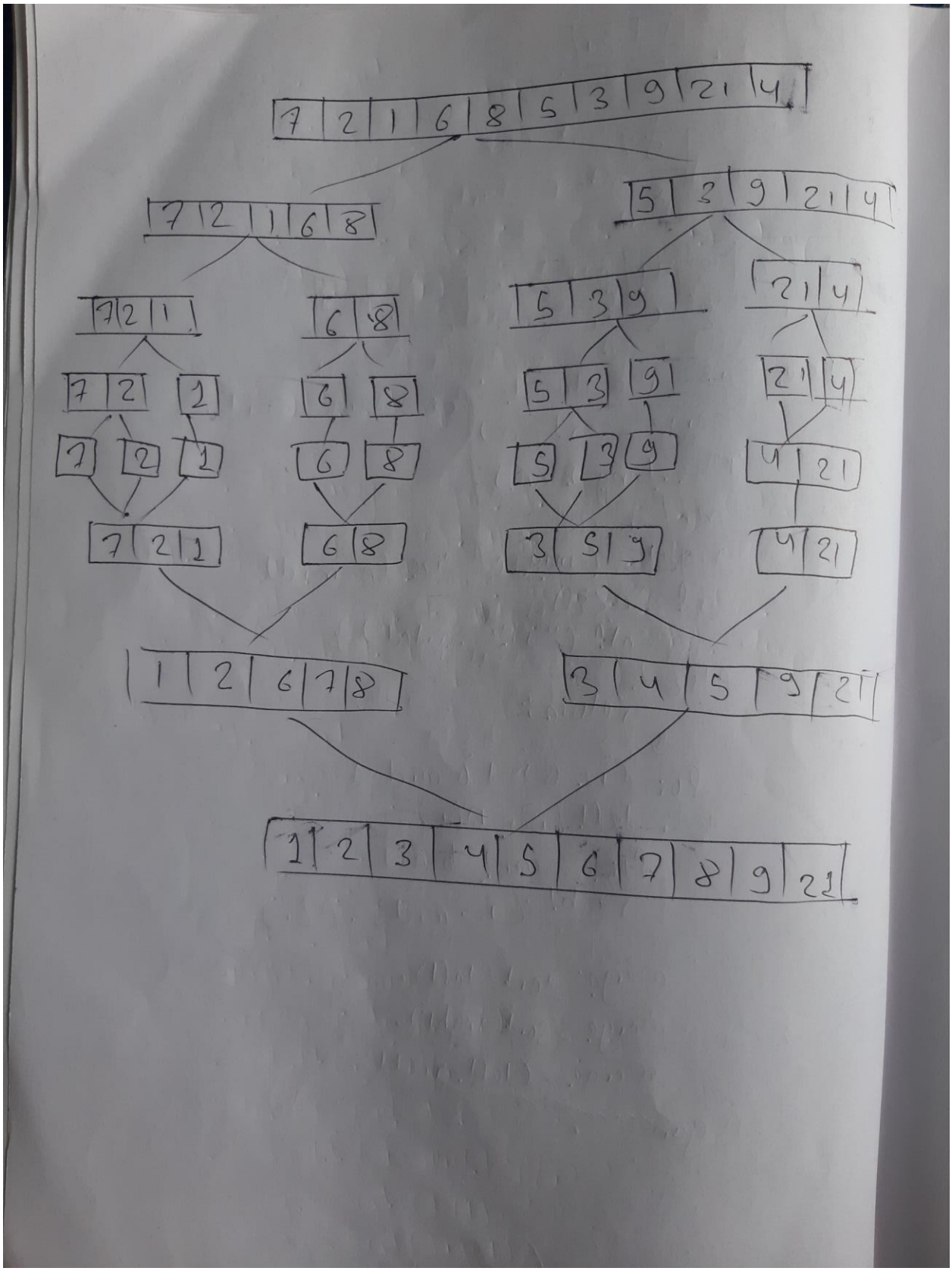












2)

2.1. Bubble sort

```
for (i = 0; i < n-1; i++){
for (j = 0; j < n-i-1; j++){
if (A[j] > A[j+1]){
temp = A[j]
A[j] = A[j+1]
A[j+1] = temp;
}
}
```

-> the bubble sort algorithm is a stable algorithm because the relative order of any two equal elements is preserved in the sorted array.

2.2. Selection sort

```
for (i = 0; i < n-1; i++){
int min = i;
for (j = i+1; j < n; j++){
if (a[j] < a[min]){
min = a;
}
}
if (min != i){
swap (a[i], a[min]);
}
}
```

-> the Selection sort is not a stable because it swaps non-adjacent elements. The time complexity is O(n²) which is worst case.

7.3. Insertion sort

```
for (i: 1; i<n; ++i
 temp = a[i];
    j = i - 1;
while (j) => (j>=0 && a[j] > temp){
 a[j+1] = a[j];
    j--;
  }
 a[j+1] = temp;
}
```

When the array is given in decending order and we are going to sort that array in asending orda that is worst case which is O(n?) because loop is also ca go to i to n and while loop also go to from 'j' to temp. O(n?).

7.4.) merge sort

```
merge (n);
{
 n = length (A);
 mid = n/2;
 left = mid
 right = n-mid;
 for (i: 0 to mid-1)
   left[i] = A[i]
 for (j = mid to n-1)
   right [i-mid] = A[i]
 merge-sort (left);
 merge-sort (right);
 merg (left, right, A)
```

```
merge (A, L, R)
 nL = L;
 nR = R;
while (nL < nR)
{
  if (L[i] <= R[i]);
   A[k] = L[i]; i++;
 else
   A[k] = R[j]; j++;
 k++;
}
while (i < nL)
   A[k] = L[i]; i++; k++;
while (j < nR)
 A[k] = R[j]; j++;
   k++;
```

→ The merge sort is not in place sorting algorithm as it requires O(n) extra space. It is stable as it does not change the order of two or more duplicate values.

2.5) · Quick sort

```
Quick sort (A, lb, ub)
  if (lb < ub)
  {
  int loc = partition (A, lb, ub)
  Quick sort (A, lb, loc - 1);
  Quick sort (A, loc + 1, ub)

Partition (A, lb, ub)
   Pivot = a[lb]
   start = lb;
   end = ub
 while (lb ≤ ub){
 while (a[start] <= pivot)
    start ++
 while [a[end]] > pivot)
    end --
   if (start < end)
   swap [a[start], a[end]
 }
   swap (a[lb], a[end])
    return end;
```

-> The Quick sort is in place sorting algorithm as it takes extra space but only for Recursive function calls. no. Quick sort is not stable as we do swapping of elements according to pivot's position without considering original position.

2.6) Shell sort

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

-> The Shell sort is inplace sorting algorithm as it take O(1) space for exchanging element. It is not stable as it does not examine elements lying between gap.

2.7) Heap sort

```
Heap if g(A, n, i)
  largest = i;
  l = 2xi + 2
  r = 2xi + 2
  if (l <= n and A[l] > A[largest]
    largest = l;
  if (r <= n and A[i] > A[largest]
      largest = r
  if (largest != i)
    swap(A[i], A[largest])
    heapify(A, n, largest)
```

-> The Heap sort is inplace as it only requires extra space to swapping elements. It is unstable

3). Soln

let denote the function $T(n)$

Now,

Partition $f(x^n) \rightarrow$ linear time complexicity $ax+b$;

For two recursive calls of Quick Sort $T(n/2) + T(n/2)$

Then,

$$T(n) = 2T(n/2) + an + b \quad \text{--- (i)}$$

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

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

from (iii) and (ii)

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

from (iv) and (i)

$$T(n) = 2(2[2T(n/8) + (n/4)] + (n/2)) + cn$$

$$= 8T(n/8) + cn + cn + cn$$

$$= 2^3 T(n/3^3) + 3cn$$

Again,

$$T(n) = 2^k T\left(\frac{n}{2^k}\right) + kcn - (a \& b)$$

Best case

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

Taking $\log_2$ on both side

$$\log 2^k = \log_2 n$$

$$\therefore k = \log_2 n \quad \text{--- (v)}$$

Now, from (v) and (vi)

$$T(n) = 2\log_2 n + T(1) + n\log_2 n$$

$$T(n) = n + c \cdot n\log_2 n$$

$\therefore$ Time complexicity $= O(n \log n)$

worst case
for partition fn and b
for two recursive call T(n-1)+1
∴ T(n): T(n-1)+an+b1)
∴ T(n): T(n-1)+an.

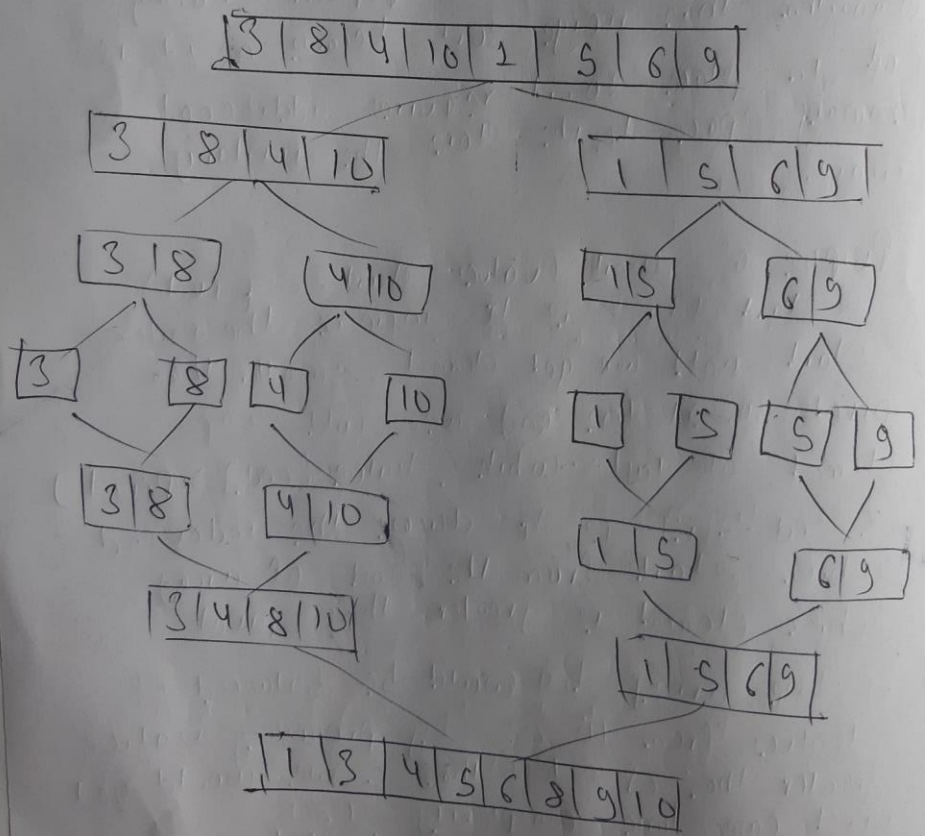
4 -> Ver. Inorder traversal produces sorted key values because it prints left and root and right element.

5 -> The Quick sort is more popular sorting algorithm than merge-sort because it is not in-place so, it requires additional memory space to the store the auxilary arrays.

6 -> Purpose of while (values[start] <= pivot) { start ++; } is to increase the index start until we get element smaller than pivot. So, at the end we could swap element at last. while (values[end] > pivot) { end --; } is to decrease the index end which starts from the end of array when element is greater than pivot.

7 -> The array would be balanced of values from 10-22 by providing values greater than 28 & less than 3. we would get worst case time complexity $O(n^2)$.
Time complexity for Quick sort.
Best case: $O(n \log n)$
worst case: $O(n^2)$
Average case: $O(n \log n)$

8.) The Quick sort is the more popular sorting algorithm as it is inplace because it takes extra space for recursive calls whare merge sort takes O(n) extra space during process. To create Quick sort the recursive function ~~with~~ should divide the array into left side first merged then after dividing toward left it is complete then dividing on right side is performed and merge too.



9) Soln

Bubble Sort
Best case = O(n)
Worst case = O(n²)
Average case = O(n²)

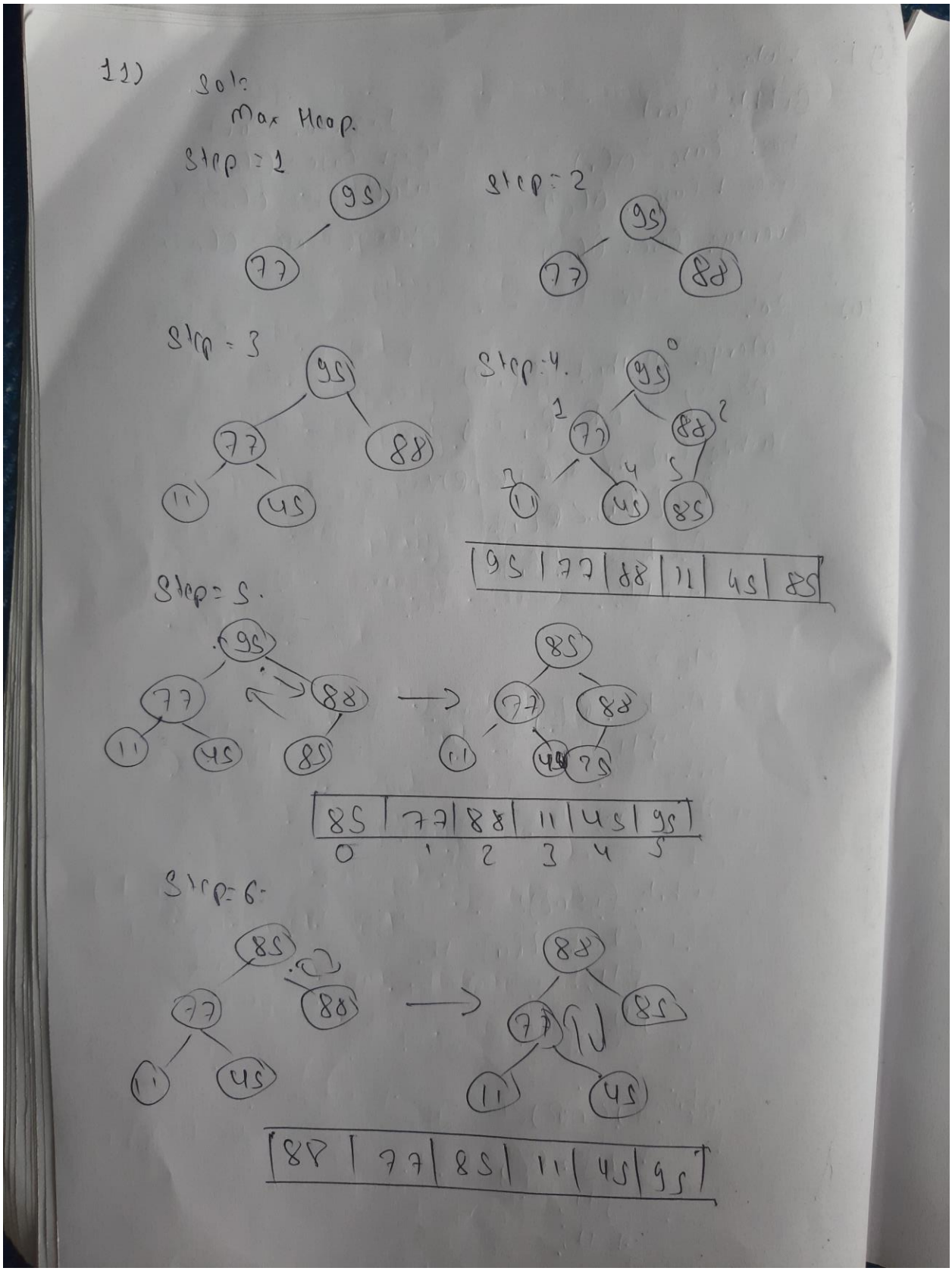
Insertion Sort
Best case = O(n)
Worst case = O(n²)
Average case = O(n²)

10) Sol:

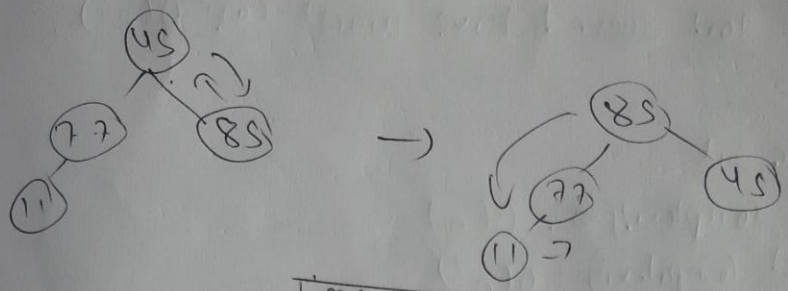
```
Merge Method (A, L, R)
nL = L, nR = R, i = j = k = 0;
while (i < nL && j < nR){
    if (L[i] < R[j]){
        A[k] = L[i];
        i++;
    }
    else{
        A[k] = R[j];
        j++;
    }
    k++;
    }
while (i < nL){
    A[k] = L[i];
    i++;
    j++;
}
while (j < nR){
    A[k] = R[j]
    j++;
    k++;
}
```

11) Sol:

Max Heap.

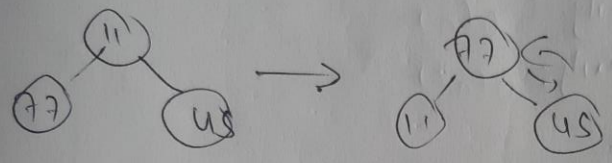


Step = 7

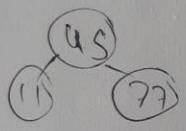


| 85 | 77 | 45 | 11 | 88 | 95 |

Step: 8

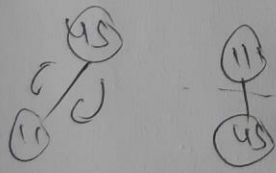


| 77 | 11 | 45 | 85 | 88 | 95 |



| 45 | 11 | 77 | 85 | 88 | 95 |

Step: 9



11	45	77	85	88	95
0	1	2	3	4	5

11	45	77	85	88	95
0	1	2	3	4	5

∴ This is called Sorted Array

12 -> Merge sort doesn't have worst-case $O(n^2)$.

13 =>

a) Insertion Sort
   Best complexity = $O(n)$
   worst complexity = $O(n^2)$

b) Bubble Sort
   Best complexity = $O(n)$
   worst complexity = $O(n^2)$

c) Sequential Search
   Best complexity = $O(1)$
   worst complexity = $O(n)$

d) Merge Sort
   Best complexity = $O(n \log n)$
   worst complexity = $O(n \log n)$

e) Selection Sort
   Best Complexity = $O(n^2)$
   worst complexity = $O(n^2)$

f) Quick Sort
   Best case = $O(n \log n)$
   worst case = $O(n^2)$ #1

14)

15)

16)

14) -> Insertion sort should be faster than Quick sort in an array which already sorted.

15 -> Sorting is known as stable in the input contain some keys and under going algorithm order of input which does not change.

Stable - Bubble Sort
Unstable - Quick Sort

16 -> Quick Sort is a well-known sorting algorithm developed by C.A.R. Hoare and on average, it makes O(n log n) comparisons to sort n items.