Assignment - 5

Name : R. Nikhila

Registu Number : 192372186

Course Name : Data Structure

Course Code : CSA0389

Submission data : 21-08-2024

Department : CSE(AI)

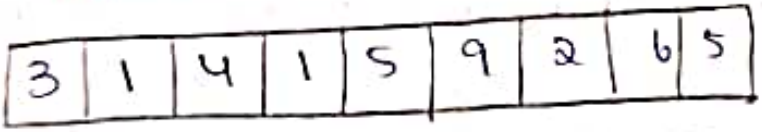
i) Write the algorithm for insertion sort and sort the following sequence : 3, 1, 4, 1, 5, 9, 2, 6, 5

Algorithm

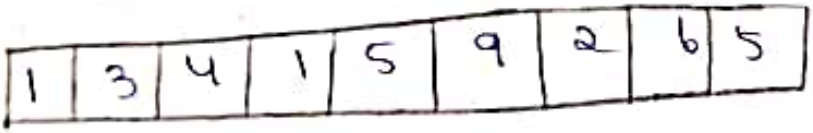
1) Start with the second element of the array
→ the first element is considered sorted
2) For each element from the second to the last
key - the element of key in the sorted position
compare the key with elements in the sorted position from left to right
shift - elements of sorted portion to right if they are greater than the key
Insert - the key into its correct position until the entire array is sorted

3) Repeated the above step for each element until sorted

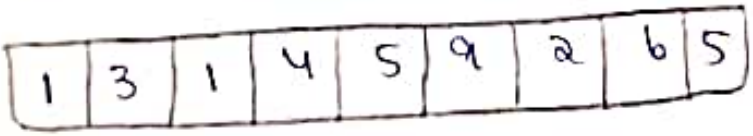
sequence :- 3, 1, 4, 1, 5, 9, 2, 6, 5



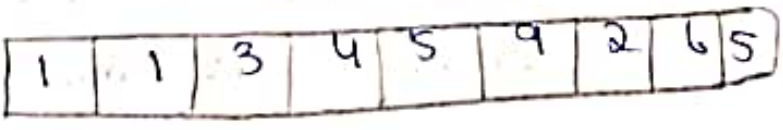
→ 1st two elements are comp



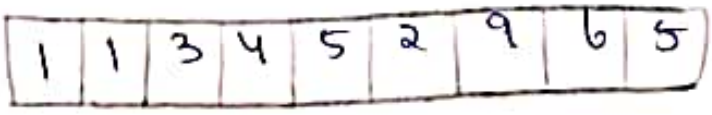
→ compare next 2 elements



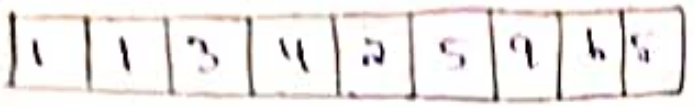
→ compare next 2 elem



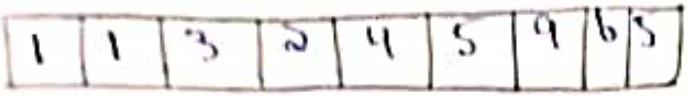
→ 6 > 2, swap 9, 2



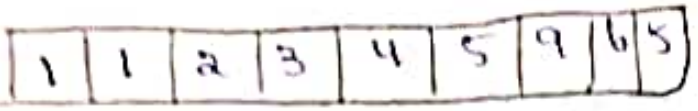
→ 5 > 2, swap



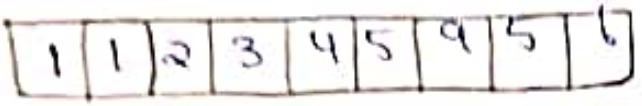
→ 4>2, swap



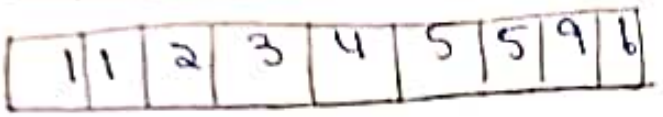
→ 3>2, swap



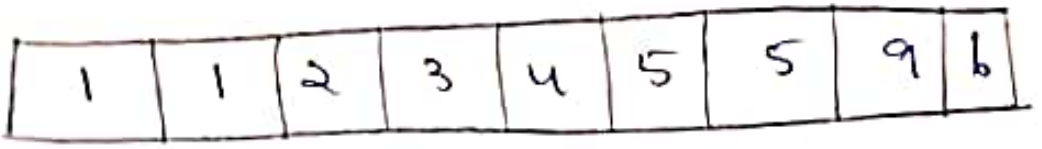
→ 6>5, swap



→ 9>5, swap



→ 9>6, swap



Sorted array.

ii) Explain the procedure for merge sort and perform the merge sort for following input. Also show input for each step 64, 8, 216, 512, 27, 729, 0, 1, 343, 125

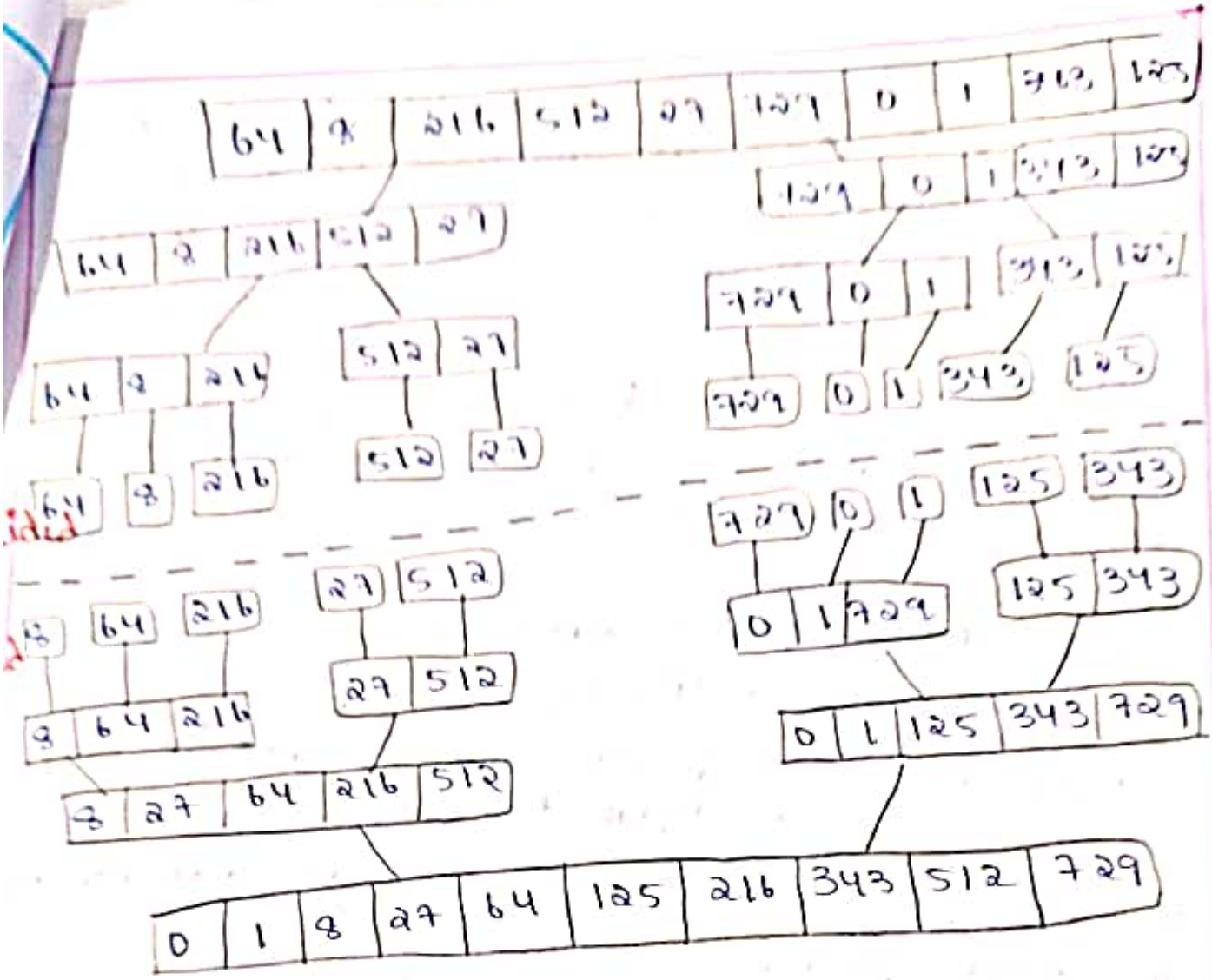
Algorithm:

→ If the array has more than one element, split into two halves

→ continue recursively splitting each half until you have subarrays that are trivially sorted

Conquer:

→ Recursively sort each other smaller subarray. Since arrays with one element are already sorted, this focus on the merging process.



Sorted array

2) Draw the concept map of partitioning in quick sort, try to write an algorithm for it, write it as follows and develop a program for it.

A key step in the quick sort algorithm is 'partitioning' the array

i) choose any number 'p' in the array to use it

ii) Initialize pointer : left pointer : start at beginning
right pointer : start just before pivot

iii) partition process :

→ move the left pointer rightwards until it finds an element greater (or) equal to P

→ move the right pointer leftwards until it finds an element less (or) equal to pivot.

iv) swap pivot : swap the pivot until elements are in correct position

v) Apply the same process to left & right sub array

Algorithm :

1) choose element at index 'high' as pivot

a) Initialize - left = low
- right = high - 1

) while left <= right

) Increment left while A (left) < pivot

) Decrement right while A (right) > pivot

4. swap pivot
5. return left

program:

```
#include <stdio.h>
void swap (int * a, int * b) {
   int temp = * a;
     * a = * b;
     * b = temp;

}
int partition (int arr[], int low, int hight ) {
 int pivot = arr[high];
  int left = low;
  int right = high - 1;
 while (left <= right) {
while( left <= right && arr [left] < pivot){
      left ++;

 }
while( left <= right && arr [right] > pivot){
    right --;

 }
  if (left <= right) {
swap ( & arr [left], & arr [right]);
   left ++;
   right --;

 }
}
```

```
swap(&arr[i], &arr[high]);
  return i;
}
void quick_sort(int arr[], int low, int high){
if (low < high) {
int pivot_index = partition(arr, low, high);
 quick_sort(arr, low, pivot_index - 1);
 quick_sort(arr, pivot_index + 1, high);
}
}
void print_array(int arr[], int size){
 for(int i=0; i< size ; i++){
  printf("%d", arr[i]);
}
  printf("\n"); }
  int main() {
 int arr[] = {10,7,8,9,1,5};
 int size = sizeof(arr)/sizeof(arr[0]);
 printf_array(arr, size);
 quick_sort(arr, 0, size-1);
 printf("sorted array");
 printf_array(arr, size);
  return 0;
  }
```