Tutorial-3

Q1 to 6 (Done in Assignment 1)

(Q7) Find two indexes such that A[i]+A[j]=k in minimum time complexity.

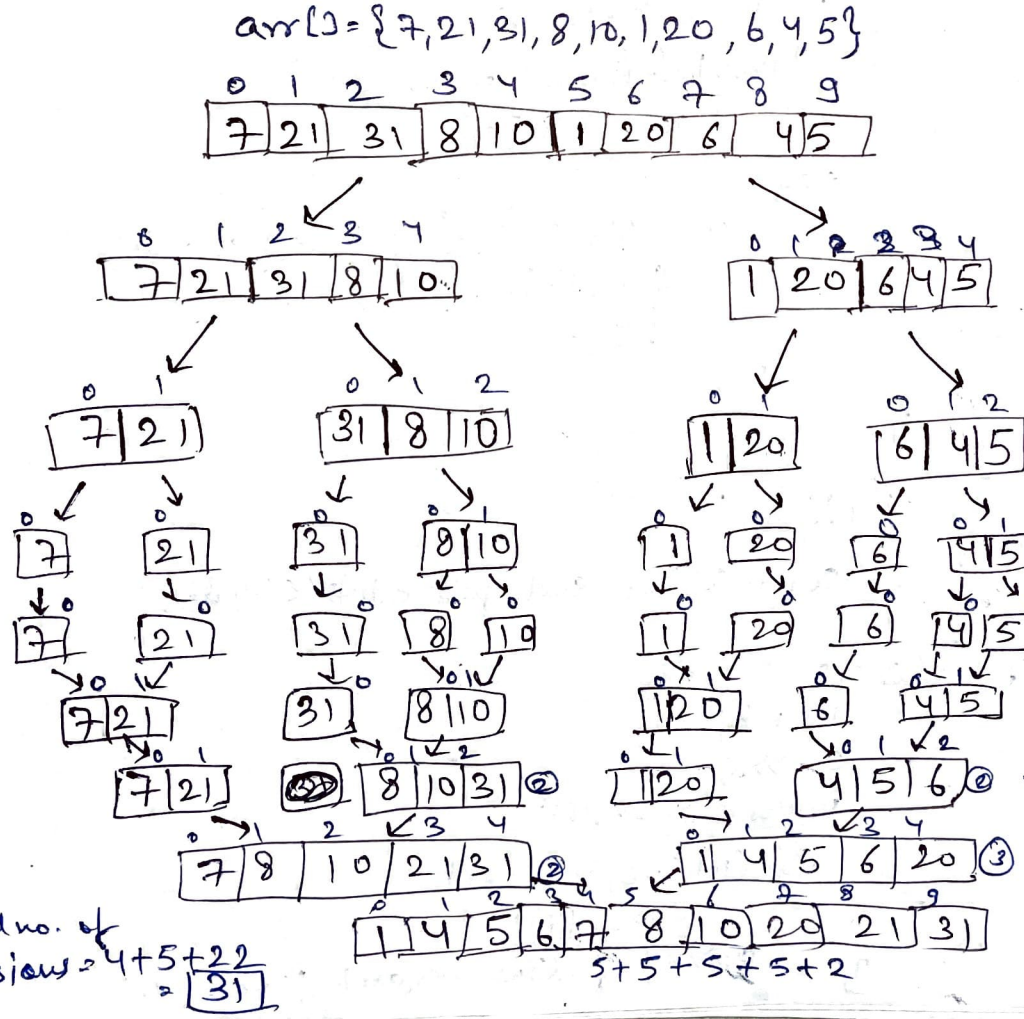
→
```
#include <iostream>
#include <vector>
#include <map>

using namespace std;

void findIndexes (vector<int>&arr, int k) {
    map <int, int> mp;
    bool found = false;
    for (int i=0; i<arr.size(); ++i) {
        int num2 = k-arr[i];
        if (mp.find (num2) == mp.end())
            mp[arr[i]] = i;
        else {
            cout << i << " " << mp[num2] << endl;
            found = true;
            break;
        }
    }
    if (!found)
        cout << "No such pair exists" << endl;
}
int main() {
    int n, k;
    cin >> n;
    vector<int> arr(n);
    for (int i=0; i<n; ++i)
        cin >> arr[i];
    cin >> k; findIndexes(arr,k); return 0;
}
```

Q8) Which sorting is best for practical uses? Explain.

→ **QuickSort :-** It is the fastest general-purpose sort. In most practical situations, Quicksort is the method of choice. If stability is important & space is available, **MergeSort** might be best.

Q8) What do you mean by the number of inversions in an array? Count the no. of inversions in array arr={7,21,31,8,10,1,20,6,4,5} using Mergesort.

→ Inversion count for an array indicates how far (or close) the array is from being sorted. If the array is already sorted, then the inversion count is zero, but if the array is sorted in the reverse order, the inversion count is maximum.



Q10) In which case, Quick sort will give the best & the ~~worst~~ worst case time complexity?

→ The best case for Quick Sort will be when the middle element is picked as a pivot.
The worst case for Quick Sort is when array is sorted in either increasing or decreasing order.

Q11) Write recurrence relation of Merge & Quick sort in best & worst case. What are the similarities & differences between complexities of two algorithms & why?

→ # Recurrence Relations

	Merge Sort	Quick Sort
Best Case	$T(n) = 2T(n/2) + n$	$T(n) = 2T(n/2) + n$
Worst Case	$T(n) = 2T(n/2) + n$	$T(n) = T(n-1) + n$

Similarities

1) Both the methods follow Divide & Conquer approach.
2) Both have best case time complexities as $O(n \log n)$.

Differences

1) Merge sort is a stable algorithm while Quick Sort is unstable algorithm.
2) Worst case time complexity of Merge sort is $O(n \log n)$ while that of Quick Sort is $O(n^2)$.
3) Merge sort is external sorting algorithm while Quick Sort is internal sorting algorithm where data is sorted in main memory.

Q12) Selection Sort is not stable by default but can you write a version of stable selection sort?

→
```
void selectionsort(int *arr, int n){
    for(int i=0; i<n-1; ++i){
        int min=i;
        for(int j=i+1; j<n; ++j){
            if(arr[min]>arr[j])
                min=j;
        }
        int key = arr[min];
        while(min>i){
            arr[min]=arr[min-1];
            --min;
        }
        arr[i]=key;
    }
}
```

Q13) Bubble Sort scans while whole array even when array is sorted. Can you modify the Bubble Sort so that it doesn't scan the whole array, once it is sorted?

→ # Modified Bubble Sort

```
void bubblesort(int *arr, int n){
    int i,j;
    bool swapped;
    for(i=0; i<n-1; ++i){
        swapped=false;
        for(j=0; j<n-1-i; ++j){
            if(arr[j]>arr[j+1])
            {
```

```
        swap(&arr[j], &arr[j+1]);
        swapped = true;
      }
    }
    if (!swapped)
      break;
  }
}
```

Q14) Your computer has a RAM of 2GB & you are given an array of 4GB for sorting. Which algorithm you are going to use for this purpose & why? Also, explain the concept of external & internal sorting.

→ for this purpose, external sorting technique, i.e., Merge Sort should be used.

- In internal sorting, all the data is sorted in main memory all the time while sorting.
- In external sorting, data is stored in the slower external memory (usually a Hard drive). In the sorting phase, chunks of data small enough to fit in main memory are read, sorted & written out to a temporary file.