Design and Analysis of Algorithms

Mirza Ghalib Baig

February 5, 2022

Disclaimer: I stake no claim on the originality of the problems.

1. You are given a list of n closed intervals $[P_1, Q_1], [P_2, Q_2], ..., [P_n, Q_n]$ on a real line as an input. Design an algorithm which return maximum number of intervals that have a common points. For example if input is $[0, 3], [-1, 2], [-2.5, 1], [2.3, 4], [5, 6]$ output should be 3.

2. Assume that the multiplication of two real numbers a and b is a unit-time operation on a computer, but finding the exponent of the form is not. Show that there exists a $\Theta(\log n)$ algorithm of finding the value c^n for any given real number c and integer n.

3. Consider a satellite moving around the earth along the equator and recording some data (air-pressure, temperature, height etc.) for scientific investigation. The data thus obtained is stored in a database where the key to each data entry is the longitude of the point. We require this data to be sorted, but each of the n data entries might have been misplaced by no more than k positions from the correct sorted order (k is a predefined constant, depending on the speed of the satellite). Design a worst-case time and space efficient algorithm for sorting such a dataset.

4. Let us consider a scenario where government of Tamil Nadu has started a unique identification project, which intends to provide a unique id number to all the people spending less than Rs 50 a day . This unique id will be used to create a database which contain other information like age, income etc. of each person. If we want to sort this data using the age of each person as a key, design a sorting algorithm which requires Θ(n) time even in the worst case. Assume that n is the population of Tamil Nadu, and that each person is at most 132 years (!) old. Furthermore, attempt the following problems:

   - Identify the range $[0, n^i]$such that any n integers in this range can be sorted in (n) time.
   - Suppose you are given n integers in the range 0 to k. Design an algorithm which returns number of integers in the range $[a, b]$ in $\Theta(n + k)$ time.
   - We have to sort a list consisting of a sorted list of n $-$ 1 integers followed by a "random" integer. Design a best sorting algorithm for such data set.

5. Show how to multiply the complex numbers $a + bi$ and $c + di$ using only three multiplications of real numbers. Assume that a, b, c, d are all real numbers.

6. For any given integer $k \geq 2$, describe a $(n^{k-1} \log n)$ worst-case time algorithm that, given a set S of n integers and another integer x, determines whether or not there exist k elements in S whose sum is exactly x. Does there exist an algorithm with an asymptotically better worst-case time complexity?

7. A sorting algorithm is called stable if it maintains the relative positions of two records with equal keys. For example, if a record X appears before a record Y (with equal keys), a sorting algorithm is stable if X remains before Y after sorting. Identify the stable sorting algorithms from the following list: insertion-sort, heap-sort, merge-sort, quick-sort (deterministic). If some of these algorithms are not stable, explain why with small examples.

8. Explain how the selection algorithm can be used in the pivot selection of the deterministic quicksort algorithm such that that running time of quicksort becomes $O(n \log n)$ even in the worst case.

9. For n elements, give a simple scheme to convert any unstable sorting algorithm to a stable sorting algorithm. How much additional running time and additional space do you require for making this conversion?

10. Modify the standard merge-sort algorithm to devise an in-place merge-sort algorithm that requires $O(n(\log n)^2)$ time in the worst-case to sort n integers.

11. Consider the following recursive function and answer the questions given below:

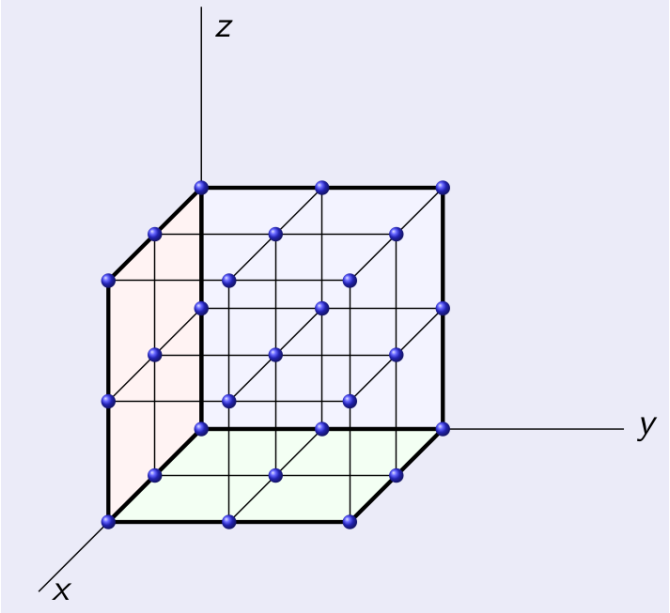
```
int Foo(int x, int y) {
  if(y==0)
          return 1;
  else if (y==1)
          return x;
  else if (y%2==0)
           return Foo(x*x,y/2);
  else return Foo(x*x,y/2)*x;
        }
```

- What does the above function compute?
- Write down the recurrence relation for the above function.
- Solve the recurrence relation obtained above to find the running time of the function.

12. Consider a building made in a shape of three dimensional cube having edge length $m^{1/3}$. In this building, we will have $m^{1/3}$ floors, where each floor has $m^{2/3}$ flats. So we have total m flats in the building numbered from 1 to m. In other words, there are m integral points in the cube where each point represents a flat of the building.

    Each floor of the building has numbering of the flats in sorted order, and the numbering of the flats across the floor is also in sorted order.

    Design an algorithm for unmanned aerial vehicles or delivery drones, which will help the drones to find out the door number mentioned on the packet in minimum possible time. Furthermore, mention the time and space complexity of your algorithms.



    [Note: For the sake of simplicity you can assume that a drone can move from any flat of the building to any other flat in one unit time. Further, once a drone reaches on a door of a flat it can compare the door number written on the package with flat number in one unit time.]

13. Design an algorithm to evaluate the following degree n polynomial

$$(x-1)^n + (x-2)^{n/2} + (x-4)^{n/4} + \ldots + (x-2^i)^{n/2^i} + \ldots + (x-n)^1$$

    in time for a given real number x and integer n.

    [Note: In RAM model of computation, the multiplication or division of two real numbers requires $\Theta(1)$time. The subtraction or addition of two real numbers can also be done in $\Theta(1)$ time. Assume that n is "nice", i.e., of the form for some integer $k \geq 1$.]

14. How would you modify Strassen's algorithm to multiply $n \times n$ matrices in which n is not an exact power of 2? Show that the resulting algorithm runs in time $\Theta(n^{\lg 7})$.

15. You are given an input array (a) containing the numbers $1, 2, 3, ..., n$ in some arbitrary orders. Two elements a[i] and a[j] form an inversion if a[i] > a[j] and i < j. Design a divide and conquer algorithms to find number of inversions. Furthermore, write down the recurrence relation and analyse the time complexity of your algorithms.

16. Your friends have been studying the closing prices of tech stocks, looking for interesting patterns. They have defined something called a rising trend, as follows. They have the closing price for a given stock recorded for n days in succession; let these prices be denoted $P[1], P[2], ..., P[n]$. A *rising trend* in these prices is a subsequence of the prices $P[i_1], P[i_2], ..., P[i_k]$, for days $i_1 < i_2 < ... < i_k$, so that

    - $i = 1$, and
    - $P[i_j] < P[i_{j+1}]$ for each $j = 1, 2, ..., k-1$.

    Thus a rising trend is a subsequence of the days—beginning on the first day and not necessarily contiguous—so that the price strictly increases over the days in this subsequence.

    They are interested in finding the longest rising trend in a given sequence of prices.

    Example: Suppose $n = 7$, and the sequence of prices is

    $$10, 1, 2, 11, 3, 4, 12.$$

    Then the longest rising trend is given by the prices on days 1, 4, and 7. Note that days 2, 3, 5, and 6 consist of increasing prices; but because this subsequence does not begin on day 1, it does not fit the definition of a rising trend.

    - Show that the following algorithm does not correctly return the length of the longest rising trend, by giving an instance on which it fails to return the correct answer.

```
Define i = 1
       L = 1
For j = 2 to n
   If P[j] > P[i] then
      Set i = j.
      Add 1 to L
   Endif
Endfor
```

    - Give an efficient algorithm that takes a sequence of prices $P[1], P[2], ..., P[n]$ and returns the length of the longest rising trend.

    [Hint: Use Dynamic Programming]

17. Write a dynamic programming algorithm for change making problems. wikifordef

18. You have a set of n integers each in the range $0, ..., K$. In time $O(n^2K)$, partition these integers into two subsets such that you minimize $|S_1 - S_2|$, where S_1 and S_2 denote the sums of the elements in each of the two subsets.
    [Hint: Use dynamic programming]