CS71: Design and Analysis of Algorithms

Smith Gupta

Week 2

Problem 6.

How many times is the comparison `i >= n` performed in the following program? ***2 points***

```
int i = 300, n = 150;
main(){
  while (i >= n){
    i = i-2;
    n = n+1;
  }
}
```

50

51

52

SOLUTION
i = 300, 258, 256, 254, …, 200, 198
n = 150, 151, 152, 153, …, 200, 201

Problem 7.

Invention of chess

According to a well-known legend, the game of chess was invented many centuries ago in northwestern India by a certain sage. When he took his invention to his king, the king liked the game so much that he offered the inventor any reward he wanted. The inventor asked for some grain to be obtained as follows: just a single grain of wheat was to be placed on the first square of the chessboard, two on the second, four on the third, eight on the fourth, and so on, until all 64 squares had been filled. If it took just 1 second to count each grain, how long would it take to count all the grain due to him?

SOLUTION

Total number of grains —

$2^0 + 2^1 + 2^2 + \ldots + 2^{63}$

Sum of first n exponents of 2 = $2^n - 1$

Total time = $2^{64} - 1 \sim 1.845 \times 10^{19}$ seconds

Problem 8.

Gaussian elimination, the classic algorithm for solving systems of n linear equations in n unknowns, requires about $\frac{1}{3}n^3$ multiplications, which is the algorithm's basic operation.

a. How much longer should you expect Gaussian elimination to work on a system of 1000 equations versus a system of 500 equations?

b. You are considering buying a computer that is 1000 times faster than the one you currently have. By what factor will the faster computer increase the sizes of systems solvable in the same amount of time as on the old computer?

Problem 9.

Consider the following algorithm.

ALGORITHM *Mystery*(n)

```
//Input: A nonnegative integer n
S ← 0
for i ← 1 to n do
    S ← S + i * i
return S
```

a. What does this algorithm compute?

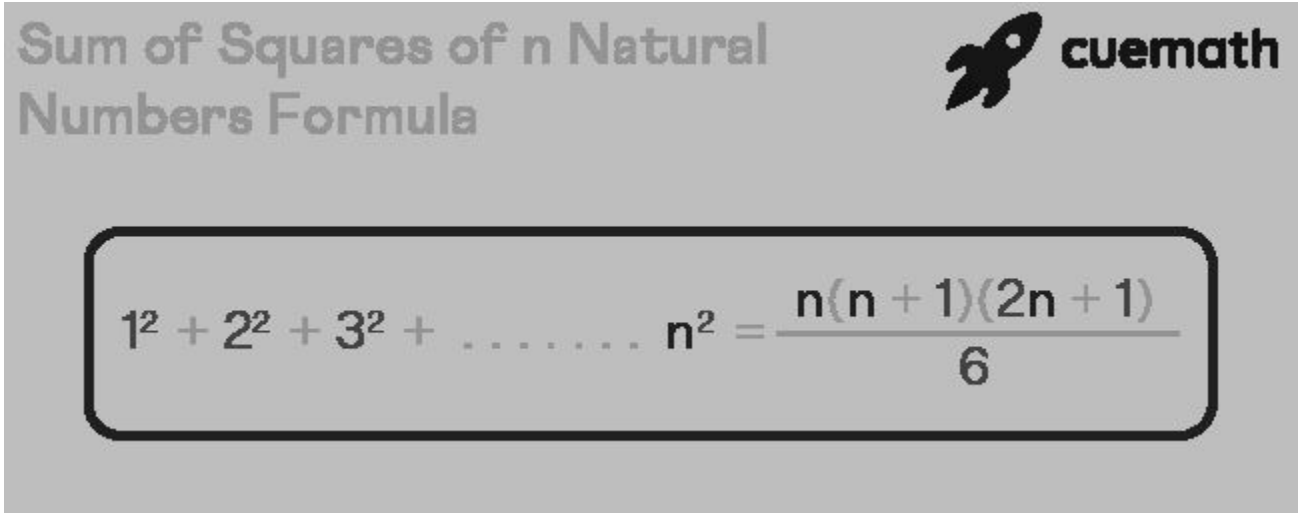
b. What is its basic operation?

c. How many times is the basic operation executed?

d. What is the efficiency class of this algorithm?

e. Suggest an improvement, or a better algorithm altogether, and indicate its efficiency class. If you cannot do it, try to prove that, in fact, it cannot be done.

Problem 9.


Sum of Squares of n Natural Numbers Formula
cuemath
$1^2 + 2^2 + 3^2 + \ldots\ldots\ldots n^2 = \frac{n(n+1)(2n+1)}{6}$

Problem 10.

Consider the following algorithm.

ALGORITHM *Secret*$(A[0..n-1])$

```
//Input: An array A[0..n − 1] of n real numbers
minval ← A[0]; maxval ← A[0]
for i ← 1 to n − 1 do
    if A[i] < minval
        minval ← A[i]
    if A[i] > maxval
        maxval ← A[i]
return maxval − minval
```

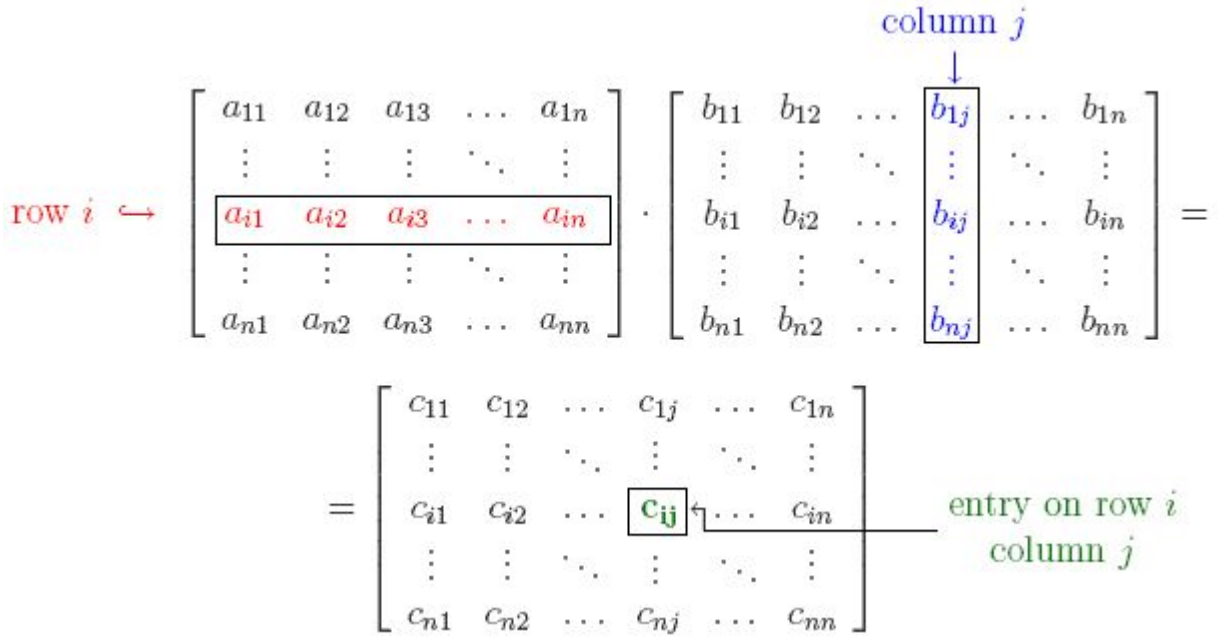
Problem 11.

Consider the following algorithm.

ALGORITHM *Enigma*$(A[0..n-1, 0..n-1])$

```
//Input: A matrix A[0..n − 1, 0..n − 1] of real numbers
for i ← 0 to n − 2 do
    for j ← i + 1 to n − 1 do
        if A[i, j] ≠ A[j, i]
            return false
return true
```

Problem 11.


column j
row i
$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{i1} & a_{i2} & a_{i3} & \dots & a_{in} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1j} & \dots & b_{1n} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ b_{i1} & b_{i2} & \dots & b_{ij} & \dots & b_{in} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nj} & \dots & b_{nn} \end{bmatrix} =$
$= \begin{bmatrix} c_{11} & c_{12} & \dots & c_{1j} & \dots & c_{1n} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ c_{i1} & c_{i2} & \dots & \mathbf{c_{ij}} & \dots & c_{in} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \dots & c_{nj} & \dots & c_{nn} \end{bmatrix}$
entry on row i
column j

Problem 12.

Consider the following recursive algorithm.

ALGORITHM $Q(n)$

//Input: A positive integer n
if $n = 1$ **return** 1
else return $Q(n-1) + 2 * n - 1$

a. Set up a recurrence relation for this function's values and solve it to determine what this algorithm computes.

Problem 12

$$Q(n) = 2*(n-1) + 2*(n-2) + 2*(n-3) + \ldots + 2*3 + 2*2 + 2*1 + 1$$

$$= 2*(1 + 2 + 3 + \ldots + (n-2) + (n-1))$$

$$= 2 * (n) * (n-1) / 2$$

$$= n * (n - 1)$$

Problem 13

Consider the following recursive algorithm.

ALGORITHM *Riddle*$(A[0..n-1])$

//Input: An array $A[0..n-1]$ of real numbers

if $n = 1$ **return** $A[0]$

else $temp \leftarrow$ *Riddle*$(A[0..n-2])$

if $temp \leq A[n-1]$ **return** $temp$

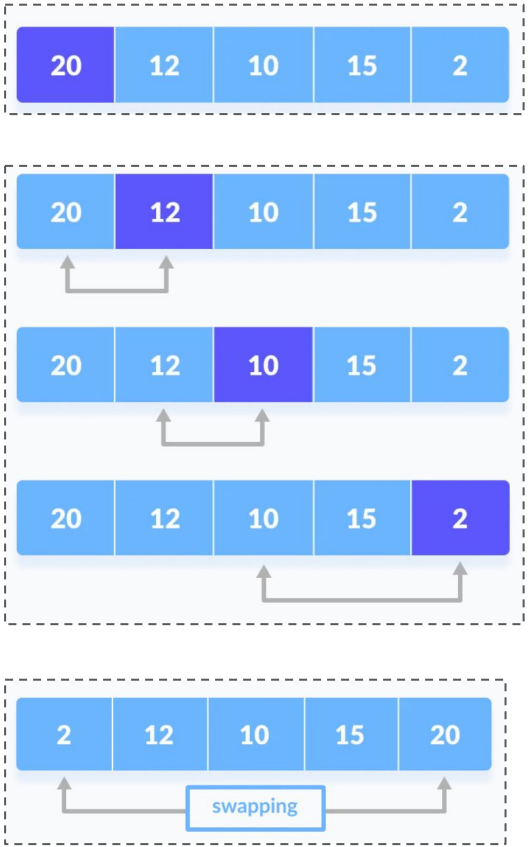
else return $A[n-1]$

Problem 1

Suppose our aim is to sort an array in ascending order. Which of the following statements is true?

- Input in descending order is worst case for both selection sort and insertion sort.
- Input in descending order is worst case for selection sort but not for insertion sort.
- Input in ascending order is worst case for both selection sort and insertion sort.
- Input in ascending order is worst case for insertion sort but not for selection sort.

Selection Sort



Complexity?

$O(n^2)$

Insertion Sort

6 5 3 1 8 7 2 4

Complexity?

$O(n^2)$

Problem 1

Suppose our aim is to sort an array in ascending order. Which of the following statements is true?

Input in descending order is worst case for both selection sort and insertion sort.

Input in descending order is worst case for selection sort but not for insertion sort.

Input in ascending order is worst case for both selection sort and insertion sort.

Input in ascending order is worst case for insertion sort but not for selection sort.

Solution

Input in descending order and ascending order are both worst case for selection sort.

Input in descending order is worst case for insertion sort.

Problem 2

Arrays A and B each contain N integers arranged in a random sequence. We want to check if A and B have any entries in common. Which of the following would be the most efficient algorithm, asymptotically?

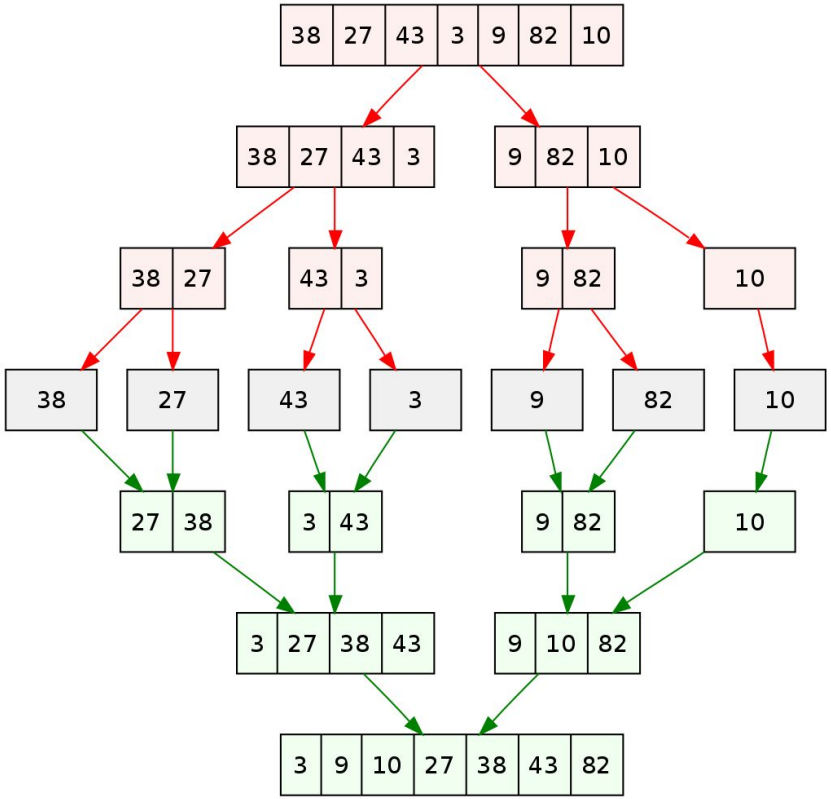
For each pair of positions $0 \leq i,j < N$, check if A[i] is equal to B[j].

Sort A using merge sort. For each element B[i], try to insert B[i] in A and see if you encounter a duplicate.

Sort A using quicksort. For each element B[i], use binary search to check if B[i] appears in A.

Sort A and B using merge sort. Merge A and B to check for duplicates.

Merge Sort



Complexity?

O(nlogn)

Quicksort

6 5 3 1 8 7 2 4

Complexity?

Worst case: $O(n^2)$
Average: O(nlogn)

Problem 2

Arrays A and B each contain N integers arranged in a random sequence. We want to check if A and B have any entries in common. Which of the following would be the most efficient algorithm, asymptotically?

For each pair of positions $0 \leq i,j < N$, check if A[i] is equal to B[j].

Sort A using merge sort. For each element B[i], try to insert B[i] in A and see if you encounter a duplicate.

Sort A using quicksort. For each element B[i], use binary search to check if B[i] appears in A.

Sort A and B using merge sort. Merge A and B to check for duplicates.

Solution

Comparing each A[i] and B[j] takes time $O(n^2)$.

Inserting each B[i] in sorted A takes time $O(n)$, so overall it takes time $O(n^2)$.

Quicksort is $O(n^2)$ in worst case

Mergesort of A and B takes time $O(n \log n)$.

Problem 3

Suppose we want to sort an array in descending order and we implement quicksort so that we always choose the last element in the array as the pivot element. Assume that the input is a permutation of {1, 2, …, n}. Which of the following would ***definitely*** be a worst case permutation of this input for this implementation of quicksort?

{1, 2, …, n} with all odd numbers in ascending order followed by all even numbers in random order

{1, 2, …, n} with all even numbers in descending order followed by all odd numbers in ascending order.

{1, 2, …, n} in descending order.

{1, 2, …, n} in some random order.

Solution

Choosing the last element as pivot for an already sorted array will split the array into size (n-1) and 1 each time.

{1, 2, …, n} in descending order.

Problem 4

Consider a plate stacked with several disks, each of a different diameter (they could all be, for instance, *dosas* or *chapatis* of different sizes). We want to sort these disks in decreasing order according to their diameter so that the widest disk is at the bottom of the pile. The only operation available for manipulating the disks is to pick up a stack of them from the top of the pile and invert that stack. (This corresponds to lifting up a stack *dosas* or *chapatis* between two big spoons and flipping the stack.)

(a) Give an algorithm for sorting the disks using this operation.

(b) How many steps will your algorithm take in the worst case?

Solution

Use a variation of selection sort. Find the widest disk and invert the stack from that point up so that it moves to the top. Then invert the entire stack to move the widest disk to the bottom. Leaving the bottom disk untouched, repeat this with the $n-1$ *disks above, to move the second widest disk to the second position from the bottom. And so on . . .*