What does $O(< expr >)$ mean?

1

What does $\Theta(< expr >)$ mean?

2

What does $\Omega(< expr >)$ mean?

3

What are the best, average and worst case complexities of ***Bubble Sort****?*

4

What are the best, average and worst case complexities of ***Merge Sort****?*

5

Give pseudo code for merging 2 sorted lists, as part of merge sort.

6

Give pseudo code for MergeSort(L).

7

What are the best, average and worst case complexities of ***Quick Sort****?*

8

The complexity (i.e. running time/space) is bounded by the $< expr >$.

1

The complexity (i.e. space/running time) has the complexity proportional to $< expr >$.

2

The complexity (i.e. running time/space) is at least by the $< expr >$.

3

Best: $O(n)$,
Average: $O(n^2)$,
Worst: $O(n^2)$

4

Best: $O(nlog \log_2 n)$,
Average: $O(nlog \log_2 n)$,
Worst: $O(nlog \log_2 n)$

5

Merge(L_1, L_2)
  if $L_1 = []$ *return* L_2
  if $L_2 = []$ *return* L_1
  $x_1 = L_1[0]$
  $x_2 = L_2[0]$
  $L'_1 = L_1[1 : |L_1| - 1]$
  $L'_2 = L_2[1 : |L_2| - 1]$
  if $x_1 \leq x_2$
    return $[x_1]$+*Merge*(L'_1, L_2)
  return $[x_2]$+*Merge*(L_1, L'_2)

Merge two sorted lists

6

MergeSort(L)
  if $|L| \leq 1$
    return L
  Split L into roughly equal halves, L_l *and* L_r
  return Merge(MergeSort(L_l*),MergeSort(*L_r*))*

MergeSort(L)

7

Best: $O(nlog \log_2 n)$,
Average: $O(nlog \log_2 n)$,
Worst: $O(n^2)$

8

What would the pseudo code be for Quick Sort?

9

Say that the input represents a positive integer, x, what is the size of n?

10

What does it mean by $O(1)$?

11

What is the minimum time for any sorting algorithm that uses only number comparisons?

12

What would the pseudo code be for Euclid's algorithm?

13

What would the pseudo code be for Fast Modular Exponentiation?

14

Consider the equation $a^x = y mod p$. If a is a primitive root modulo p, then for every $y (1 \leq y < p)$, such an $x (1 \leq x < p)$ exists. What is x?

15

The is the inverse of exponentiation.

16

$\lfloor \log_b x \rfloor + 1$ *Where b is the number representation, usually binary (so 2).*

10

quicksort(L)
    if length of $L \leq 1$
        return L
    remove the first element, x, from L
    $L_{\leq}$:= *elements of L less than or equal to x*
    $L_{>}$:= *elements of L greater than x*
    L_l := *quicksort*($L_{\leq}$)
    L_r := *quicksort*($L_{>}$)
    return L_l + [x] + L_r

Quick Sort

9

$n \log_2 n$

12

It takes a constant time, no matter the amount of data, to perform the operation.

11

fme(a,b,k)
    $d = a$
    $e = b$
    $s = 1$
    While $e > 0$
        if e is odd
            $s = (s.d) mod k$
        $d = d^2 mod k$
        $e = \lfloor e/2 \rfloor$
    return s

Fast Modular Exponentiation

14

// Assume a >= b
hcf(a,b)
    if $b = 0$
        return a
    $r = a mod b$
    return hcf(b,r)

Euclid's algorithm

13

The discrete logarithm is the inverse of exponentiation.

16

X is the ***discrete logarithm*** *of y with base a, modulo p.*

15

Why can the private key not, in practice, be recovered from the public key when p is large?

17

What is one way you can argue correctness of Euclid's algorithm?

18

What would half the correctness proof be for Euclid's algorithm?

19

$(a.b) mod k =$

20

Let p be a prime number. What is meant by a primitive root modulo p?

21

What does saying that algorithm A runs in time g mean?

22

Let $r = amodb$. $hcf(a,b) = hcf(b,r)$ because all factors of a and b are also factors of b and r and vice versa. If they have the same factors, they have the same highest common factor.

18

To calculate a public key, y, a private key, x is needed. The equation for modular exponentiation can be used: $y = g^x modp$ It is considered a one-way function - easy to compute, hard to invert. For a large p, the only way to figure out the private key would be to use brute force, which would take a large amount of time.

17

$(a.b)modk = (amodk.bmodk)modk$

20

As $r = amodb$, $\exists q$ such that $a = bq + r$, $\therefore r = a - bq$.
Suppose x is a factor of a and b, then $\exists yandz$ such that $a = xy$, $b = xz$.
Hence: $r = xy - xzq$, $r = x(y - zq)$.
$\therefore$ x is a factor of r (and also of b and r).

19

Given an input of size n, the number of operations executed by A is bounded above by g(n).

22

The numbers r_x between 1 and $p-1$ that, when raised by the numbers between 1 and $p-1$ compute all the numbers between 1 and $p-1$ in some order with no repetitions.

21