

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

1

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

2

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

3

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

4

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

5

*MergeSort and QuickSort of examples of what
algorithmic technique?*

6

*What are the three steps of **Divide-and-conquer**?*

7

*What is the **Greedy** algorithm? Give an example in
term of the knapsack problem.*

8

The complexity (i.e. space/running time) has the complexity proportional to $\langle \text{expr} \rangle$.

The complexity (i.e. running time/space) is bounded by the $\langle \text{expr} \rangle$.

2

1

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

The complexity (i.e. running time/space) is at least by the $\langle \text{expr} \rangle$.

4

3

Divide-and-conquer

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

6

5

A greedy algorithm is an algorithmic paradigm that follows the problem solving heuristic of making the locally optimal choice at each stage with the hope of finding a global optimum.

With the knapsack problem, organise all possible items by their value to weight ratio. Then iterate through this list, adding them to the knapsack where possible.

- We divide the input into parts.
- Solve the parts (often recursively).
- Combine the solutions to give the final result.

Three steps of Divide-and-conquer

8

7

*Dynamic Programming is a [redacted] method -
we solve all [redacted] first and then
[redacted] to solve the given problem.*

9

*Give some examples of optimisation problems with
dynamic programming.*

10

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

11

*What would the pseudo code be for Fast Modular
Exponentiation?*

12

*What are some of the advantages of ElGamal
encryption?*

13

*What is the basic procedure for an encryption and
decryption using public key cryptography if Alice
wants to send a message to Bob?*

14

*Describe public key generation in ElGamal
encryption using p as the Prime Modulus and g as
the Primitive root (as described in the COMP26120
lab)*

15

*Describe the encryption procedure used in the
ElGamal cryptosystem given that person B wants to
send message M to person A*

16

- * Some path-finding algorithms use dynamic programming, e.g. Floyd's algorithm.
- * Some text similarity tests, e.g. longest common subsequence.
- * Knapsack problems: The 0/1 knapsack can be solved using dynamic programming.
- * Constructing optimal search trees.
- * Some travelling sales person problems.
- * Genome matching and protein-chain matching.

Optimisation problems with dynamic problems.

10

Dynamic Programming is a bottom-up method - we solve all smaller problems first and then combine them to solve the given problem.

9

```
fme(a,b,k)
  d = a
  e = b
  s = 1
  While e > 0
    if e is odd
      s = (s.d)modk
    d = d2modk
    e = ⌊e/2⌋
  return s
```

Fast Modular Exponentiation

12

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

Euclid's algorithm

11

Alice generates a private random integer a and Bob generates a private random integer b
 Alice generates her public value $g^a \bmod p$
 Bob generates his public value $g^b \bmod p$
 Alice computes $g^{ab} = (g^a)^b \bmod p$
 Bob computes $g^{ba} = (g^b)^a \bmod p$
 Now they have a shared secret k since $k = g^{ab} = g^{ba}$

14

Sender Verification
 Private key remains with owner
 Public key is freely distributable
 No secret channel needed at any point
 No need for pre-shared keys

13

Obtain A 's public key (p, g, g^a)
 Represent the message M as integers in the range $0, \dots, p-1$
 Select a random integer k from $1 \leq k \leq p-2$
 Compute $\gamma = g^k \bmod p$ and $\delta = m \cdot (g^a)^k$
 Send ciphertext $c = (\gamma, \delta)$ to A

16

Generate a large p and a g in $1 \leq g < p$
 Generate a random integer a in $1 \leq a \leq p-2$
 Compute $g^a \bmod p$. The public key is

(p, g, g^a)

The private key is a

15

<p><i>Describe the decryption process used in the ElGamal cryptosystem given that person A has received cyphertext (γ, δ) from person B, encrypted encrypted using the public key (p, g, g^a)</i></p> <p>17</p>	<p><i>Consider the equation $a^x = y \bmod p$. If a is a primitive root of modulo p, then for every $y(1 \leq y < p)$, such an $x(1 \leq x < p)$ exists. What is x?</i></p> <p>18</p>
<p><i>The is the inverse of exponentiation.</i></p> <p>19</p>	<p><i>Why can a private key in the ElGamal cryptosystem not, in practice, be recovered using the public key when p is large?</i></p> <p>20</p>
<p><i>What is one way you can argue correctness of Euclid's algorithm?</i></p> <p>21</p>	<p><i>What would half the correctness proof be for Euclid's algorithm?</i></p> <p>22</p>
<p><i>$(a.b) \bmod k =$ </i></p> <p>23</p>	<p><i>Let p be a prime number. What is meant by a primitive root modulo p?</i></p> <p>24</p>

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

18

Use private key a to compute $(\gamma^{p-1-a}) \bmod p$
NOTE THAT: $(\gamma^{p-1-a}) = \gamma^{-a} = g^{-ak}$
Recover the message M by computing $(\gamma^{-a} \cdot \delta \bmod p)$
Note that this evaluates to $(g^{-ak} \cdot g^{ak} \cdot M \bmod p)$ or
 $1 \cdot M \bmod p$

17

To calculate a public key, y , a private key, x is
needed. The equation for modular exponentiation
can be used to generate the public key: $y = g^x \bmod p$
where g is a primitive root of the modulus p .
It is considered a one-way, or trapdoor function -
easy to compute, hard to invert. For a large p , one
of the few ways to figure out the private key x would
be to calculate $g^x \bmod p$ for every x in $1 \leq x < p$
and find when one of these results matches y

20

The discrete logarithm is the inverse of
exponentiation.

19

As $r = a \bmod b$, $\exists q$ such that $a = bq + r$,
 $\therefore r = a - bq$.
Suppose x is a factor of a and b , then $\exists y$ and z 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).

22

Let $r = a \bmod b$. $hcf(a, b) = hcf(b, r)$ 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.

21

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.

24

$$(a.b) \bmod k = (a \bmod k . b \bmod k) \bmod k$$

23

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

25

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

26

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

27

Give pseudo code for MergeSort(L).

28

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

29

What would the pseudo code be for Quick Sort?

30

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

31

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

32

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

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

26

25

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

MergeSort(L)

Merge(L₁, L₂)
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 $[x_1] + \text{Merge}(L'_1, L_2)$
return $[x_2] + \text{Merge}(L_1, L'_2)$

Merge two sorted lists

28

27

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
 $L_l := \text{quicksort}(L_{\leq})$
 $L_r := \text{quicksort}(L_{>})$
return $L_l + [x] + L_r$

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

Quick Sort

30

29

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

$n \log_2 n$

32

31

What is a permutation of a set?

33

What do we mean by a composition of two permutations?

34

What is the number of possible permutations on an n -element set?

35

In the context of a permutation, what do we mean by a transposition?

36

Convert this pair of simultaneous equations into matrix form

$$a_{1,1}x_1 + a_{1,2}x_2 = b_1$$

$$a_{2,1}x_1 + a_{2,2}x_2 = b_2$$

37

What is the determinant of the matrix:

$$\begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \end{pmatrix}$$

38

What is an upper triangular matrix and how do you calculate its determinant?

39

Which 4 operations have no effect on a matrix's determinant?

40

The composition is the product of two permutations, α and β , on a set n , given by $\alpha \cdot \beta(n)$ or $\beta(\alpha(n))$

34

A 1-to-1 map of the set onto itself. In basic terms, it is a set mapped to another order of itself. i.e $[0, 1, 2, 3, 4] \mapsto [2, 4, 1, 0, 3]$

33

A transposition is a special kind of permutation where only 2 elements in a set are affected (they are swapped). On a set X a transposition $\sigma = (i, j)$ is given by

$$\sigma(k) = \begin{cases} j & \text{if } k = i \\ i & \text{if } k = j \\ k & \text{ow.} \end{cases}$$

36

$n!$

35

$a_1a_4 - a_2a_3$
Often denoted as:

$$\begin{vmatrix} a_1 & a_2 \\ a_3 & a_4 \end{vmatrix}$$

The original system of equations to which the matrix corresponds only has a unique solution if the determinant is non-zero.

38

$$\begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

37

It is a matrix where all of its entries below the diagonal are zero.

$$\begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ 0 & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{n,n} \end{pmatrix}$$

Its determinant is calculated by taking the product of the entries on the diagonal. i.e $a_{1,1} \cdot a_{2,2} \cdot \dots \cdot a_{n,n}$

39

Transposing two rows
Transposing two columns
Adding a multiple of one row to another
Adding a multiple of one column to another
Also note that if all entries in any row or column are 0 then the determinant is 0

40

In a tree, if node u is the **parent (ancestor)** node of v , then v is a () of u . Two children of the same parent are .

41

What is a tree?

42

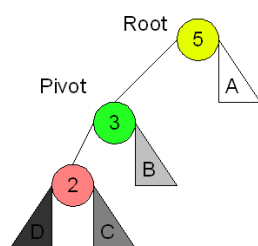
In a tree, an external node is known as . It has no .

43

In a tree, an internal node has one or more .

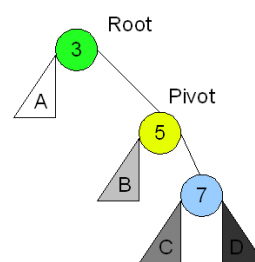
44

What do we do here?
Left Left Case



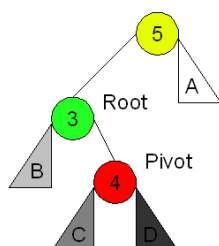
45

What do we do here?
Right Right Case



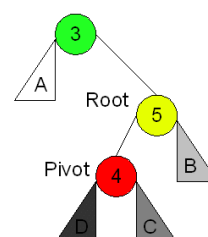
46

What do we do here?
Left Right Case



47

What do we do here?
Right Left Case



48

A tree T is a non-empty set of nodes storing useful information in a parent-child relationship with the following properties:

T has a special node r referred to as the root.
Each node v of T different from r has a parent node u .

42

In a tree, if node u is the **parent (ancestor)** node of v , then v is a child (descendent) of u . Two children of the same parent are siblings.

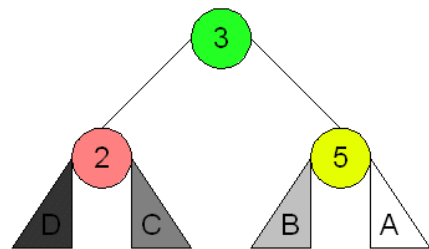
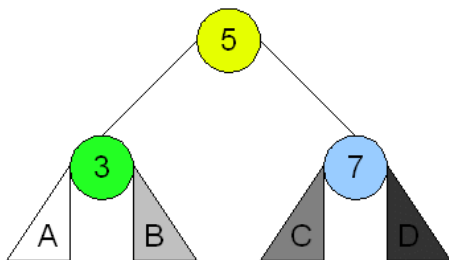
41

In a tree, an internal node has one or more children.

In a tree, an external node is known as a leaf node.
It has no children.

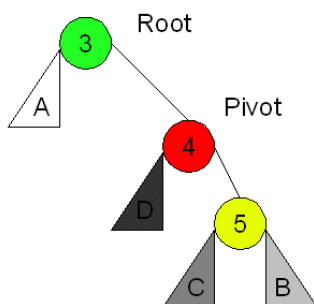
44

43

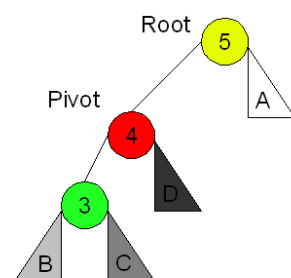


46

45



Then a right-right!



Then a left-left!

48

47

What does a Depth First Search use?

49

What does a Breadth First Search use?

50

What is the running time of Dijkstra's algorithm?

51

What's the running time of a depth first search when the graph is an adjacency matrix?

52

What's the running time of a depth first search when the graph is an adjacency list?

53

How do we insert an element into a heap?

54

How do we remove the smallest element from a (min) heap?

55

What is a priority queue?

56

A queue!

A stack!

50

49

$O(V^2)$ since finding neighbours takes $O(V)$ time.

$O(E + V \log(v))$

52

51

First, you insert it at the next space in the heap (last element of current row, or a new row), then you keep swapping it with its parent if the parent is larger than it.

$O(V + E)$

54

53

*A **priority queue** P is a container of elements with keys associated to them at the time of insertion.*

- insertItem(k, e): Inserts an element e with key k into P . - removeMin(): Returns and removes from P an element with the smallest key.

Two fundamental methods in a priority queue.

We move the last element from the heap to the first element (we can override the first element since we've removed it). Now we 'down heap' by swapping the moved node with its smallest child until it is smaller than both its children or it has no children.

56

55

What is a heap?

57

A heap-based priority queue consists of:

Heap: A complete binary tree with keys that satisfy the heap-order property.

Root: A reference to the last node in T .

Comparator: A comparator that defines the total order relation among keys.

58

*What is an **AVL Tree**?*

59

*Dijkstras algorithm is a **Shortest Path** algorithm for graph structures.*

60

*Give the pseudocode for Dijkstra's algorithm w/
priority queue*

61

A heap-based priority queue consists of:
Heap: A complete binary tree with keys that satisfy the heap-order property.
Last: A reference to the last node in T .
Comp: A comparator that defines the total order relation among keys.

58

*A **heap** is a binary that stores a collection of keys at its internal nodes that satisfies two additional properties:*
A relational property that affects how the keys are stored and a structural property.
It allows insertions and removals to be performed in logarithmic time.

57

Dijkstras algorithm is a shortest path algorithm for graph structures.

60

An AVL tree is another balanced binary search tree. Named after their inventors, Adelson-Velskii and Landis, they were the first dynamically balanced trees to be proposed. Like red-black trees, they are not perfectly balanced, but pairs of sub-trees differ in height by at most 1, maintaining an $O(\log n)$ search time.

59

```
function Dijkstra(Graph, Source):
    dist[source] = 0, create Q
    for each vertex v in Graph:
        if v != source, dist[v] = INFINITY
        prev[v] = UNDEFINED
        Q.addWithPriority(v, dist[v])
    while Q != empty
        u = Q.extractMin()
        for each neighbour v of u:
            alt = dist[u] + length(u, v)
            if alt < dist[v]
                dist[v] = alt, prev[v] = u
                Q.decreasePriority(v, alt)
    return dist, prev
```

Code for dijkstras

61