Mid term revision

- A revision note
- Self study (implementing some algorithms)

Exam structure

- Answer **Three from FIVE** in paper CS2AO17, at least TWO from four algorithms question
  (70% of module marks)
- Topics covered in Algorithm
  - **Additional data structures***
    Tree, Heap, Graph, tree and graph traversals
  - **Algorithms**
    Divide and Conquer*
    The greedy algorithms (Week 8-9)
    Dynamic programming with Second revision (Week 10-11)
- ***First Revision**

Control abstraction

Define solutions of problems by Control abstraction

- Flow of control is un-ambiguous
- Primary operations are undefined (small combine, solve,….)

Algorithm D&C(P)

if Small(P) return Solve(P)

else{

Divide P into smaller instances P_1, P_2, … P_k

Apply D&C to each of these subproblems

return Combine(D&C(P_1), D&C(P_2), …,D&C(P_k))

}

D&C Max-min

Redundant possibilities are eliminated as quickly as possible

1. Divide the list into small groups.
2. Then find max and min of each group.
3. The max/min of result must be one of maxs and mins of the groups.

e.g. $A = [5,7,1,4,10,6]$

$A_1 = [5,7,1]$, $max(A_1) = 7$, $min(A_1) = 1$
$A_2 = [4,10,6]$, $max(A_2) = 10$, $min(A_2) = 4$
So the min and max of A is $min(1,4)$ *and* $max(7,10)$,
i.e. 1 *and* 10

Trace of recursive calls

[5, 7,1, 4, 10, 6]

[5, 7,1]		[4, 10, 6]		Divide

[5]	[7,1]	[4]	[10, 6]

5,5	1,7	4,4	6,10

1,7	4,10

Combine	1,10

Selection -- Strategy

Choose a value in the list.

Partition the list so that the chosen value is in its final position if we are sorting.

- Call this position j.

(i.e. all values to the left of j are smaller and all the elements to the right are larger).

- Since j is in correct place if $j=k$, we have found the kth smallest (if $j=n-k$, the k^{th} largest).
- If $j > k$ then the k^{th} smallest is in the left sub-list otherwise in the right sub-list.

Selection algorithm example

Find second smallest k=2

- [5,7,4,1,10,6]
- [4,1,5, 7,10,6] after partition –5 is the third smallest j =3
- [4,1,5] consider left sub-list A[1,2,3]
- [1,4,5] after partition –4 is the second smallest j =2 =k

Heap and Graphs

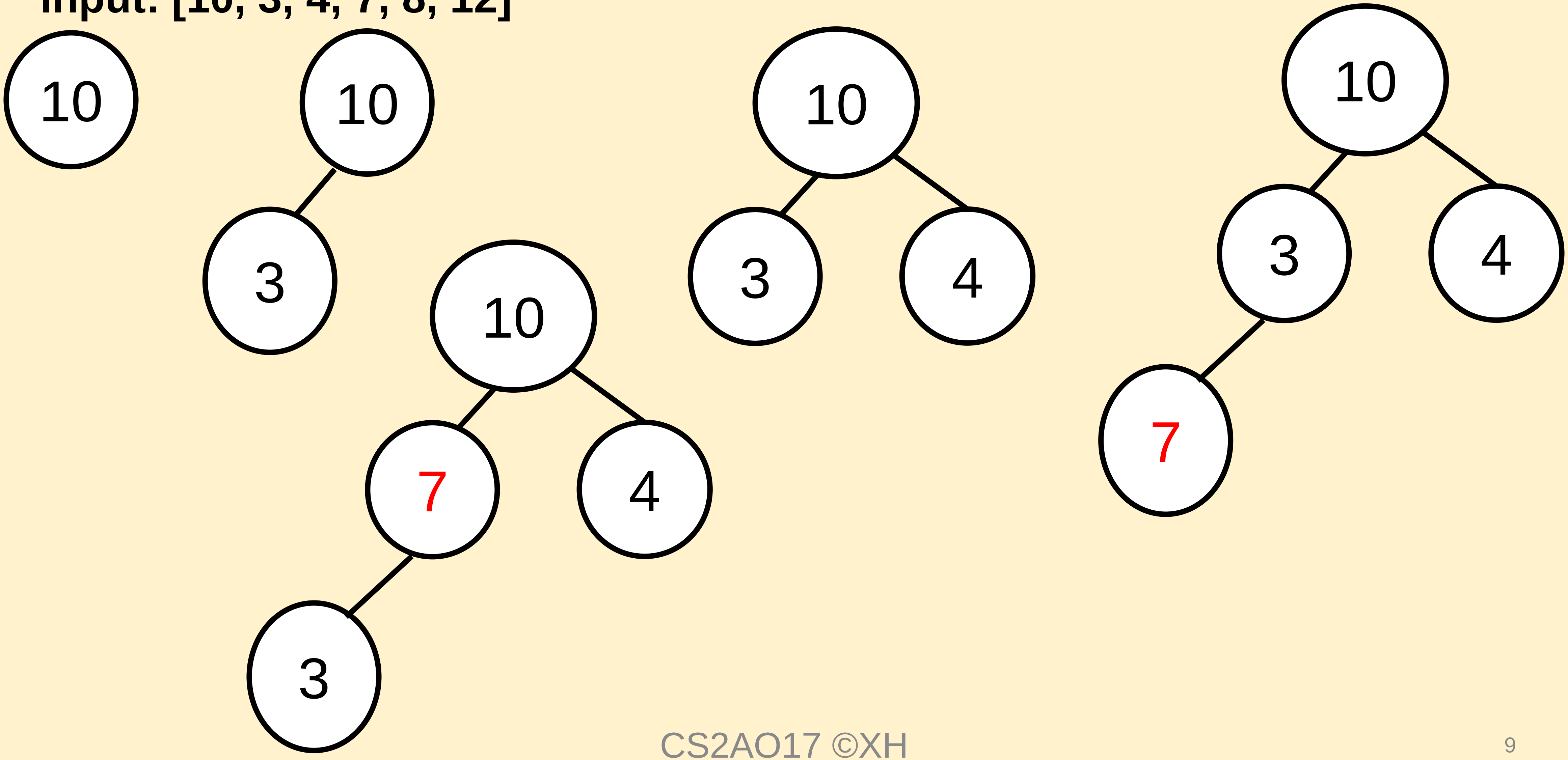
Main topics

- Heaps
  - Construction of Heap
    - Insertion (worst case O(nlogn))
    - bottom up (average O(n))
  - Heapsort

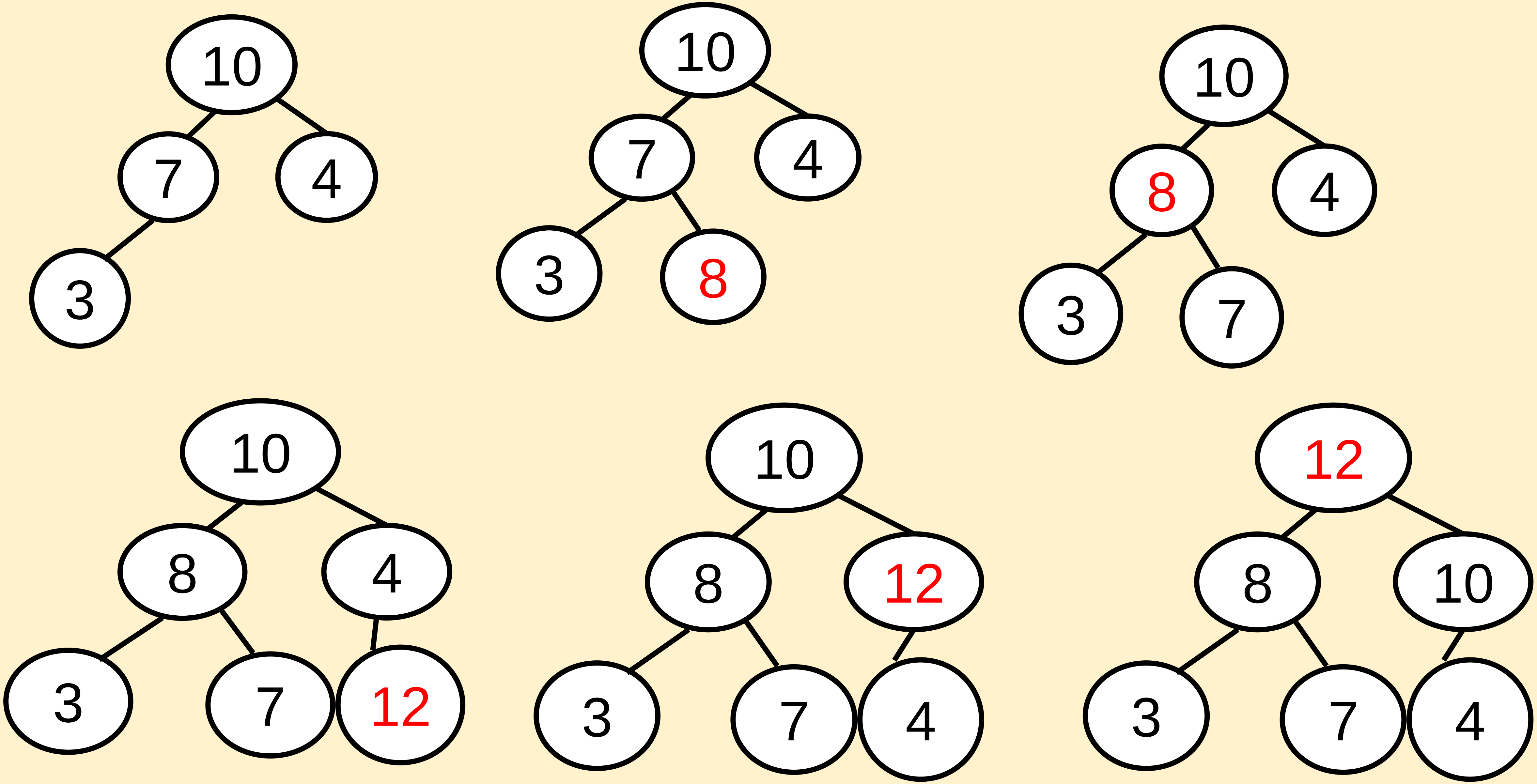
- Graphs
  - Representation
    - Adjacent list
    - Adjacent matrix
  - Traversals
    - BFS
    - DFS

Making a Heap from *n* values

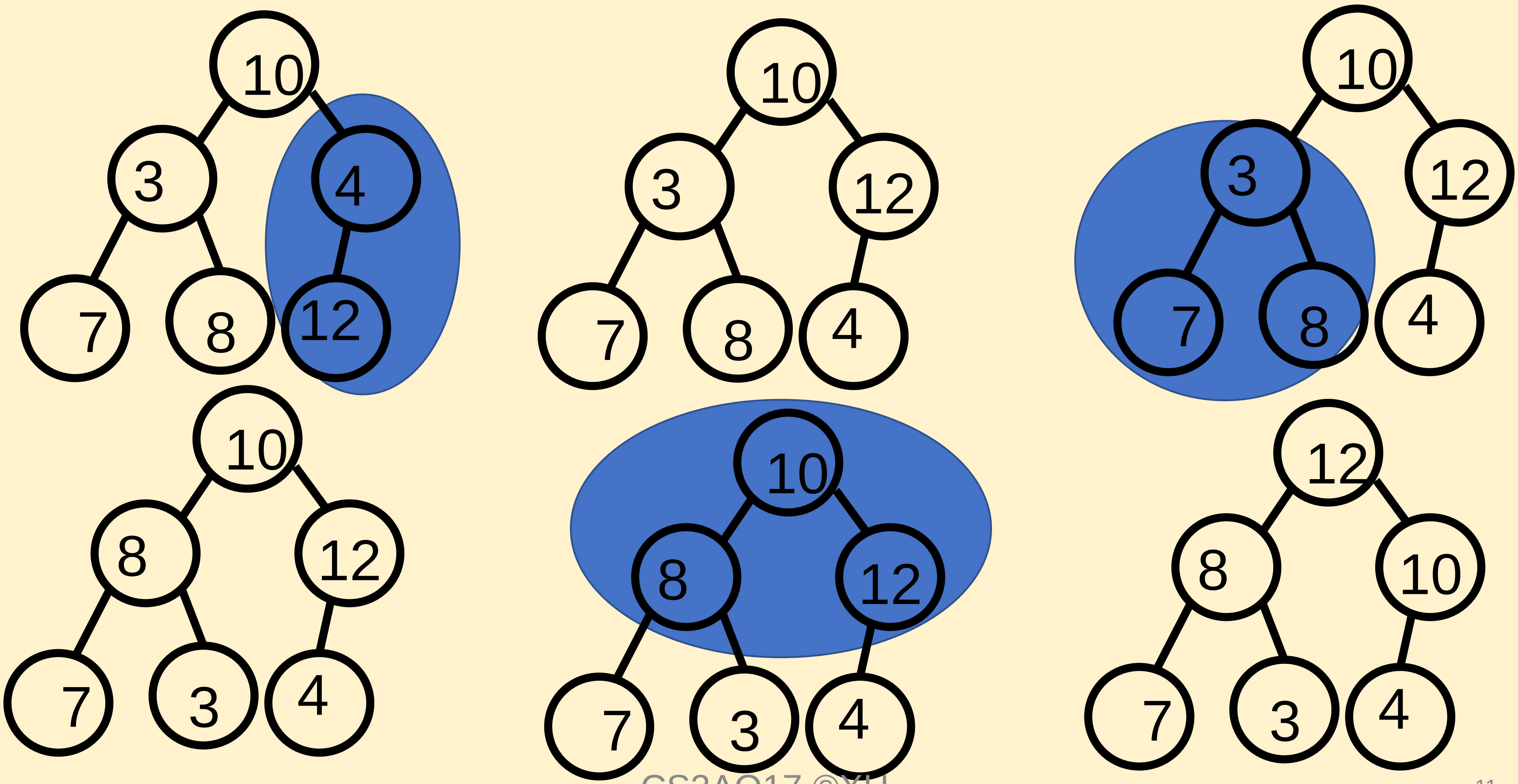
Input: [10, 3, 4, 7, 8, 12]



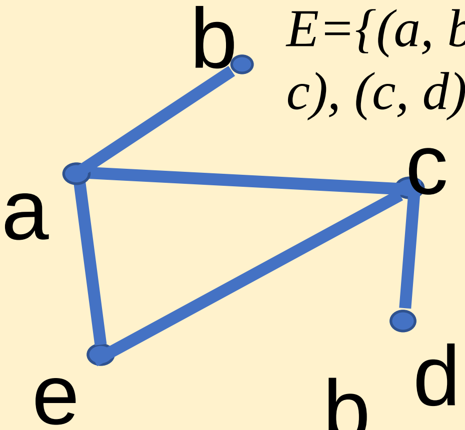
Input: [10, 3, 4, 7, 8, 12]



Example Fast heap (Bottom up)



Definitions (Graph)



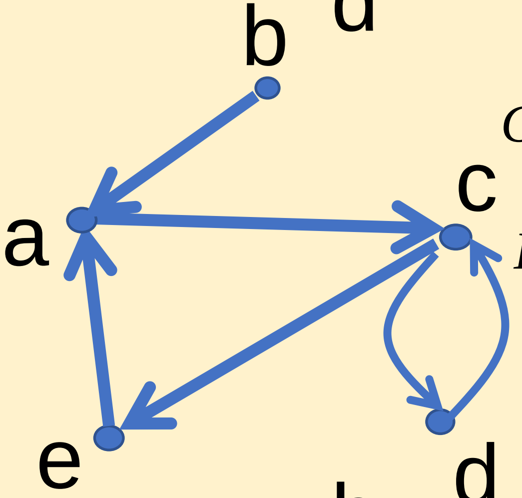
G= (V, E), V= (a, b, c, d, e); E={(a, b), (a, c), (a, e),(e, c), (c, d) }

Graph

- a graph *G(V, E)* consists of a finite set *V* of vertices and a set of *E* of edges where each edge is a pair of vertices *(i, j)*

Directed/undirected Graph

- a graph is undirected if *(i, j) = (j, i)*, i.e., pair of vertices are unordered for all edges, otherwise directed



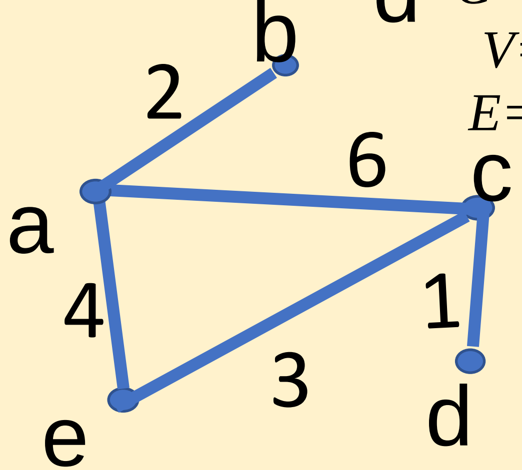
G= (V, E), V= ? E=?

Weighted graph

- a weighted graph G(V; E) has a cost, a real number, attached to an edge.

Degree

- The degree of a vertex is the number of edges attached to it.
  - ✓ In-degree of vertex *j* is the number of edges ending at *j*.
  - ✓ Out-degree of vertex *j* is the number of edges leaving *j*.

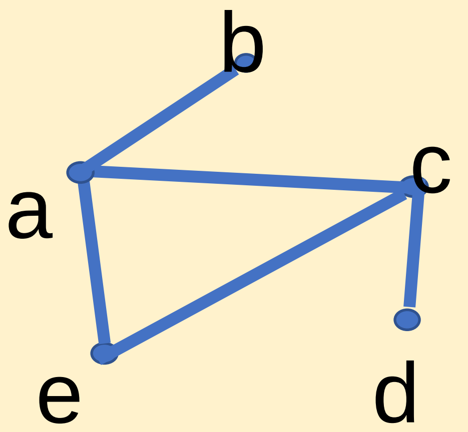


G= (V, E), V= ? E=?

Matrix representation

Undirected graph

$$A[i,j] = \begin{cases} 1 & \text{if edge } (i, j) \text{ is in } E(G) \\ 0 & \text{otherwise} \end{cases}$$



$$\begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{bmatrix}.$$

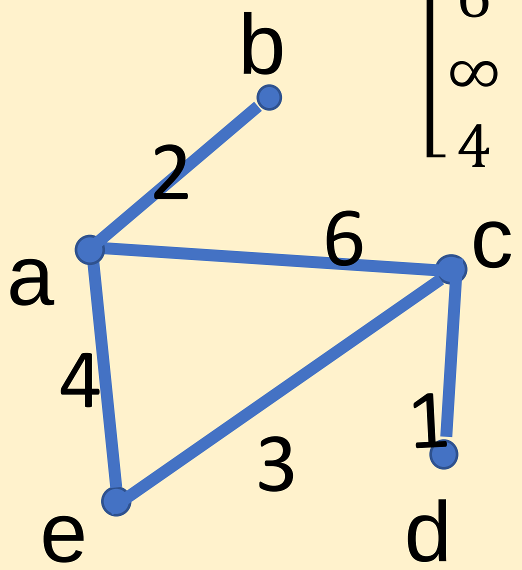
Directed graph

$$A[i,j] = \begin{cases} 1 & \text{if edge } <i, j> \text{ is in } E(G) \\ 0 & otherwise \end{cases}$$

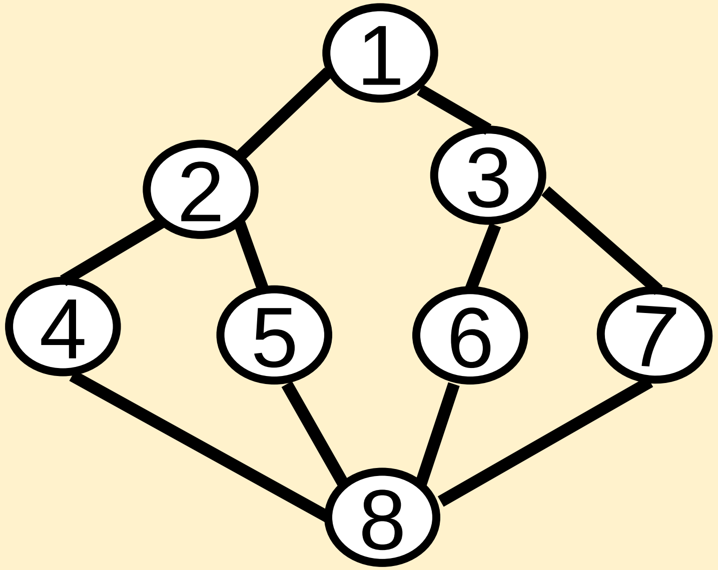
Weighted graph

$$A[i,j] = \begin{cases} c & \text{if edge } <i, j> \text{ is in } E(G) \\ \infty & \text{otherwise} \end{cases}$$

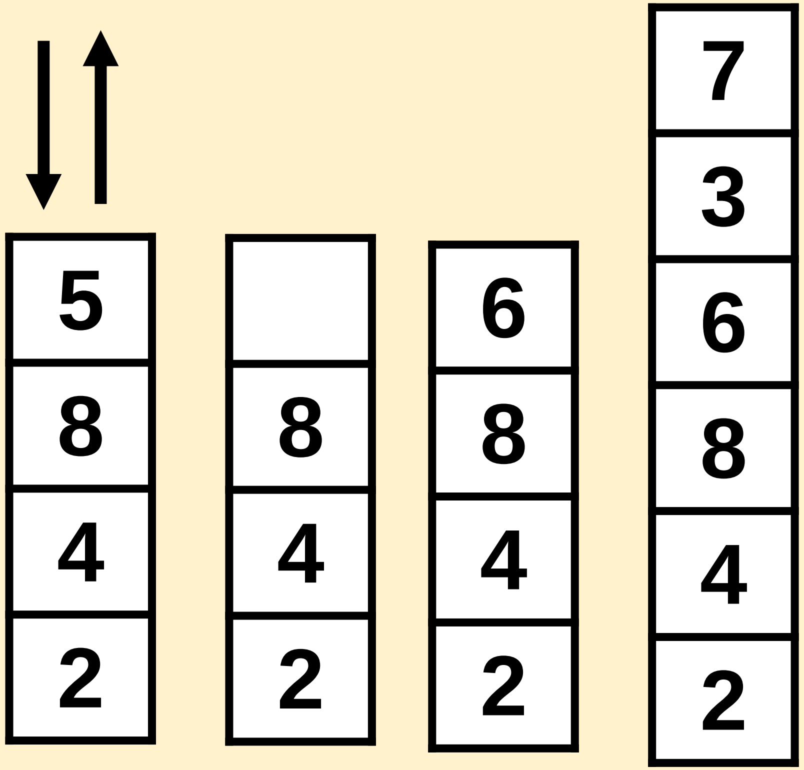
$$\begin{bmatrix} 0 & 2 & 6 & \infty & 4 \\ 2 & 0 & \infty & \infty & \infty \\ 6 & \infty & 0 & 1 & 3 \\ \infty & \infty & 1 & 0 & \infty \\ 4 & \infty & 3 & \infty & 0 \end{bmatrix}$$



Example: Depth First Search from vertex 1



Stack of visited vertices from 1



Vertices are visited vertices order 1,2,4,8,5,6,3, 7

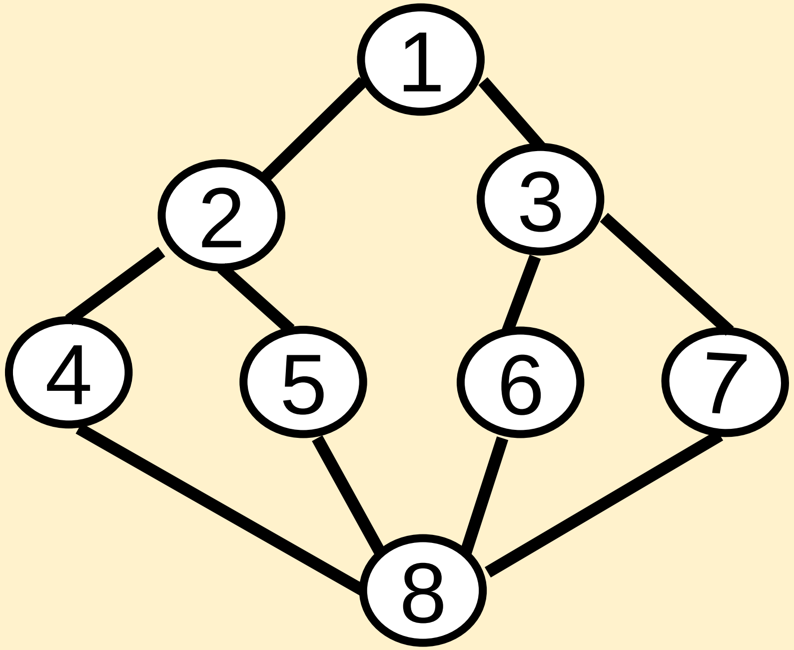
Breath first search

Theorem: Algorithm BFS visits all reachable vertices.

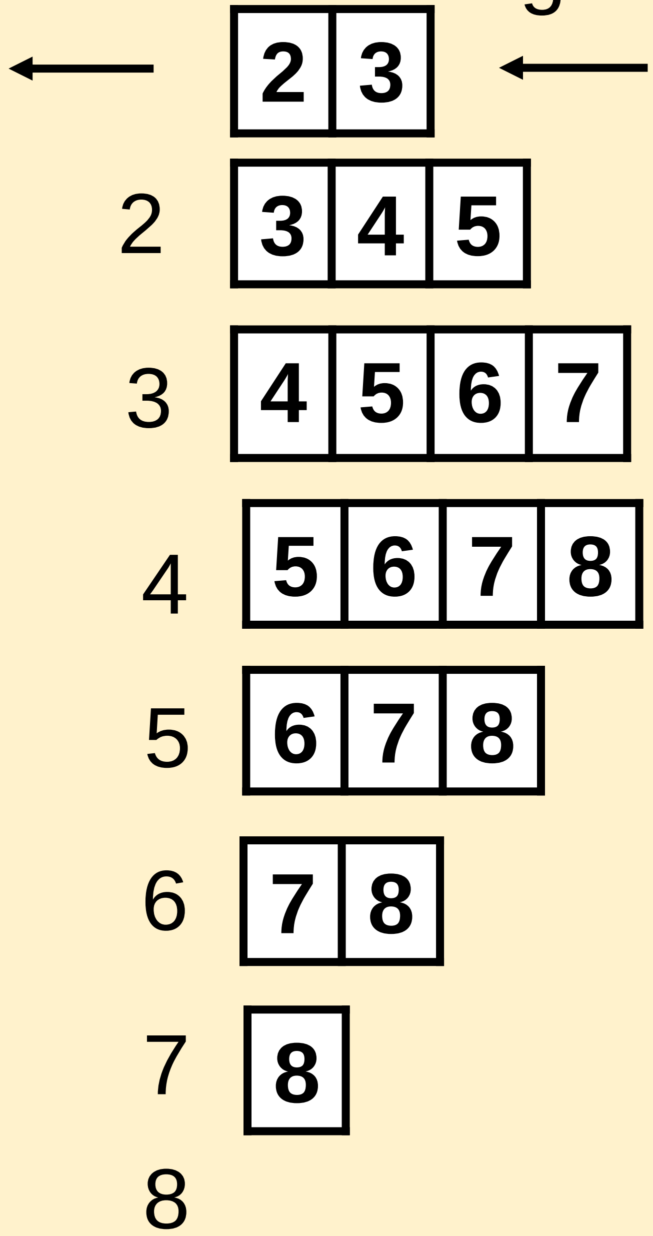
Algorithm:

- Start at a vertex v– mark it as reached
- The vertex v is, as yet, unexplored
- When all the vertices adjacent to it (connected by an edge) have been visited, v has been explored(reached).
- Collect all the unvisited vertices adjacent to v and add them to a list.
- Take a vertex from the list and repeat the process
- When there are no vertices left in the list they have all been explored (reached).
- This yields the set of vertices that are "reachable" from the start vertex v.

Breadth First Search from vertex 1



Queue for tracking visiting order



Vertices are visited vertices order: 1,2,3,4,5,6,7,8

Applications of Graph traversal

Connected Components

- If G is connected undirected graph then all vertices of G are visited on first call of BFS. If G is not connected then we need at least two calls to BFS.
- An extension of BFS algorithm can be designed to find all the connected components.

Spanning Trees

- A graph G has a spanning tree if and only if G is connected.
- A slight modification of BFS can be made to compute a spanning tree.

Suggested problems

- Heap sort
- Convex hull
- D&C max-min

- Work with others together to solve as many as possible.
  e. g. two problems as a group two.
  or three problems as a group of four.
- If you work on your own, do at least one problem.
- For each problem (for your own reference)

  Write a code or find a code from internet.
  Write a A4 page to explain the method.
  Implement the code (any language)
  Report the results, attach to the method page.
- You are invited the share in blackboard forum.
- This work is unassessed to replace Week 7 lecture as self study, as part of revision.

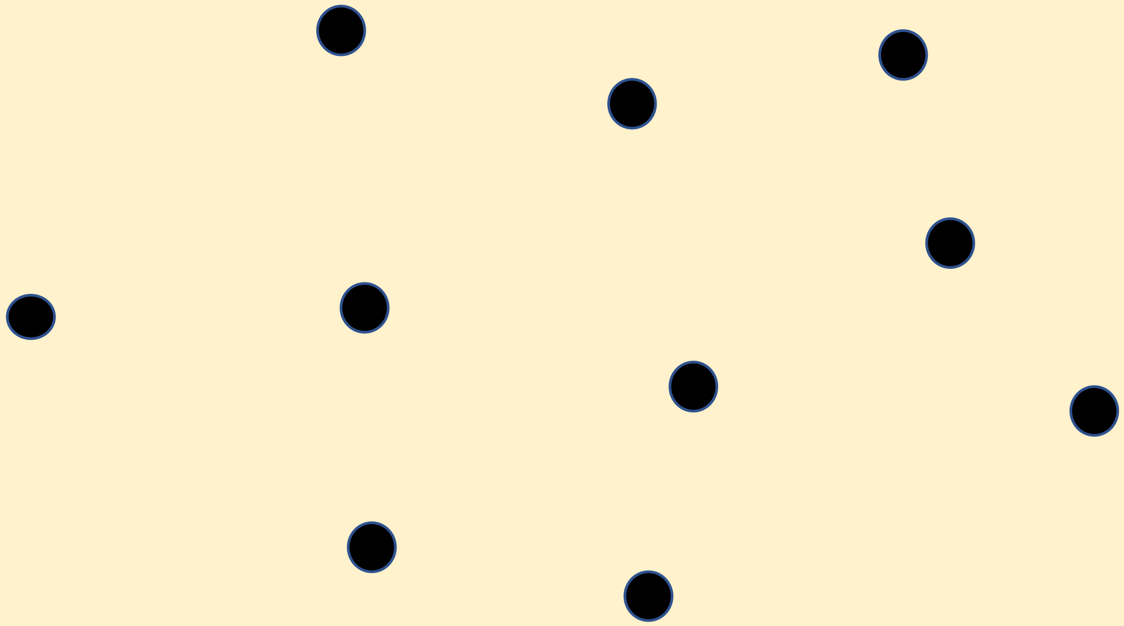
Heap sort

Sort out the following array using heapsort

[1, 6, 16, 13, 7, 3, 86, 30]

Convex hull

Create a convex hull of at least 10 points, e. g.



D&C Max-min

Find max-min using Divide and Conquer for

[1 2 3 4 5 6 7 8 9 10]